DIAGRAM SOFTWARE

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

PROGRAMACIÓN DE OPENERP/ODOO

1. MÓDULOS EN OPENERP

José Zambudio Bernabeu
jose.zambudio@diagram.net

NubeaERP global cloud solutions

DIAGRAM SOFTWARE

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ÍNDICE

- Estructura de un módulo
  - Modelos
  - Archivos XML
- Fields
  - Parámetros Generales
  - Tipos básicos: boolean, integer, float, char…
  - Tipos relacionales: one2many, many2one…
  - Tipo function
  - Tipo property
- Vistas
  - Menuitems y Actions
  - Tipos básicos: Form, tree, search
  - Vistas avanzadas: Graph, Calendar, Gantt
  - Atributos
- Métodos ORM
  - Create, search, read, write, unlink, browse

ESTRUCTURA DE UN MÓDULO

TIPO ELEMENTOS
ESTRUCTURA MÓDULO
MODELOS
XML
CSV
YAML
VIEWS
OTROS

- Tipos de elementos
  1. Business object
     - Clases de python las cuales extienden de la clase orm.Model. La gestión en base de datos de dichos módulos la mantiene el ORM.
  2. Data
     - Vienen dados por ficheros XML o CSV, y nos proporcionan todos aquellos datos que serán introducidos en base de datos. Entre estos tenemos: la declaración de vistas, workflows, datos demo, datos de configuración…
  3. Reports
     - Ficheros RML, HTML/MAKO o templates de OpenOffice con los que junto con la información de nuestro módulo, utilizaremos para generar informes PDF, HTML o ODT.

TIPOS ELEMENTOS
ESTRUCTURA MÓDULO
MODELOS
XML
CSV
YAML
VIEWS
OTROS

ESTRUCTURA DE UN MÓDULO

En una instalación básica de OpenERP podemos encontrar varios ejemplos de módulos en la carpeta addons.

Dentro de dicha carpeta, cada una de las carpetas corresponde a un módulo OpenERP distinto.

ESTRUCTURA DE UN MÓDULO

TIPOS ELEMENTOS
ESTRUCTURA MÓDULO
MODELOS
XML
CSV
YAML
VIEWS
OTROS

account
account_accountant
account_analytic_analysis
account_analytic_default
account_analytic_plans
account_anglo_saxon
account_asset
account_bank_statement_extensions
account_budget
account_cancel
account_chart
account_check_writing
account_followup
account_payment
account_report_company
account_sequence
account_test
account_voucher
analytic
analytic_contract_hr_expense
analytic_user_function
anonymization

association
audittrail
auth_crypt
auth_ldap
auth_oauth
auth_oauth_signup
auth_signup
base_action_rule
base_iban
claim_from_delivery
contacts
crm
crm_claim
crm_helpdesk
crm_partner_assign
crm_profiling
crm_todo

decimal_precision
delivery
document
document_page
document_webdav
email_template
event
fetchmail
hr_evaluation
hr_expense
hr_holidays
hr_payroll
hr_payroll_account
hr_recruitment
hr_timesheet
hr_timesheet_invoice

hr_timesheet_sheet
idea
knowledge
l10n_ar
l10n_at
l10n_be
l10n_be_coda
l10n_be_hr_payroll
l10n_be_hr_payroll_account
l10n_be_invoice_bba
l10n_bo
l10n_cr
l10n_de
l10n_ec
l10n_es
l10n_et
l10n_fr
l10n_fr_hr_payroll
l10n_fr_rib

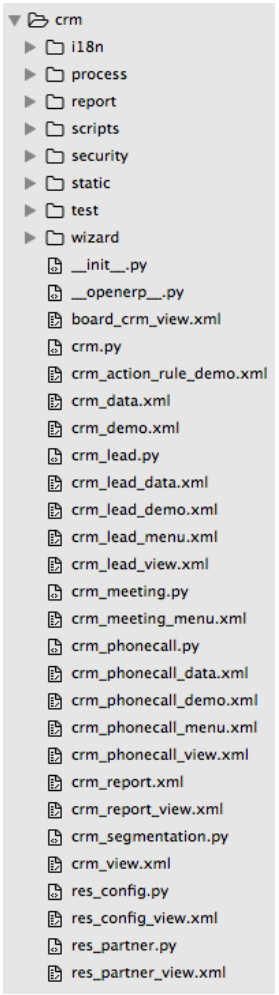
Por convención

- Nombres en minúsculas
- Separados por _
- Empezamos por los nombres de mayor relevancia, por ejemplo, aquellos de los que dependamos/ modificamos

TIPOS ELEMENTOS
ESTRUCTURA MÓDULO
MODELOS
XML
CSV
YAML
VIEWS
OTROS

ESTRUCTURA DE UN MÓDULO

Estructura interna:

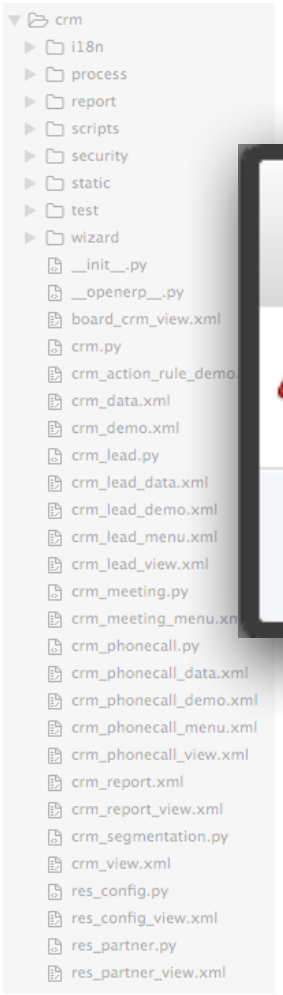


- **crm**: Directorio del módulo
  - **i18n**: Archivos de traducciones
  - **demo**: Registros de datos demo.
  - **report**: Declaración de informes, y archivos.
  - **security**: Declaración de grupos y ACL's
  - **view**: Vistas y menuitems.
  - **wizard**: Declaración de wizards.
  - **workflow:** Declaración de workflows.
  - **static**: Ficheros estáticos del servidor web.
    - **lib**: Third-party libraries
    - **src**: CSS/JS/XML/IMG
    - **tests**: JS Tests
  - **test**: Tests unitarios/funcionales
  - **data**: Registros de datos necesarios para la aplicación
  - **controllers**: RPC-JSON endpoints
  - **doc**: Posible documentación a aportar.

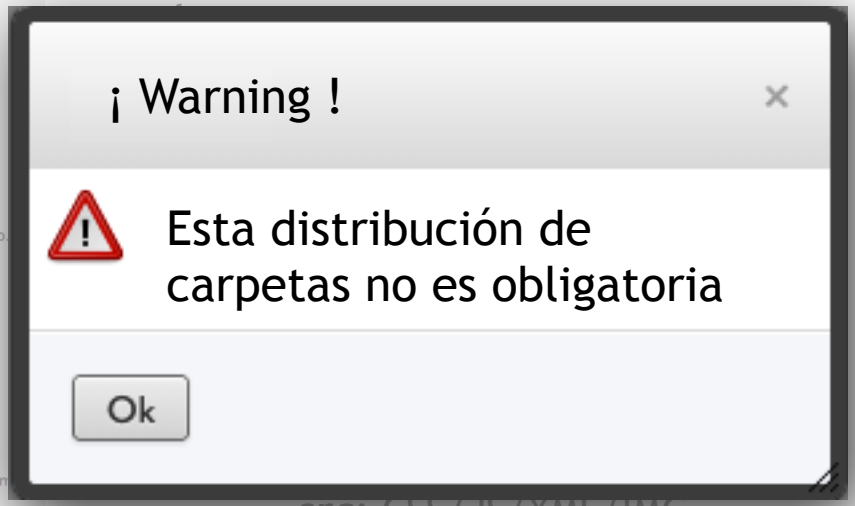
TIPOS ELEMENTOS
ESTRUCTURA MÓDULO
MODELOS
XML
CSV
YAML
VIEWS
OTROS

ESTRUCTURA DE UN MÓDULO

Estructura interna:



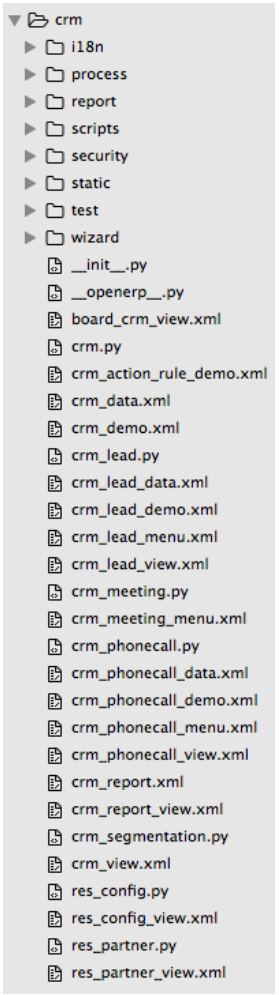
- **crm**: Directorio del módulo
  - **i18n**
  - **src**: CSS/JS/XML/IMG
  - **tests**
  - **test**
  - **data**
  - **controllers**
  - **doc**: Posible documentación a aportar.



TIPOS ELEMENTOS
ESTRUCTURA MÓDULO
MODELOS
XML
CSV
YAML
VIEWS
OTROS

ESTRUCTURA DE UN MÓDULO

Estructura interna:

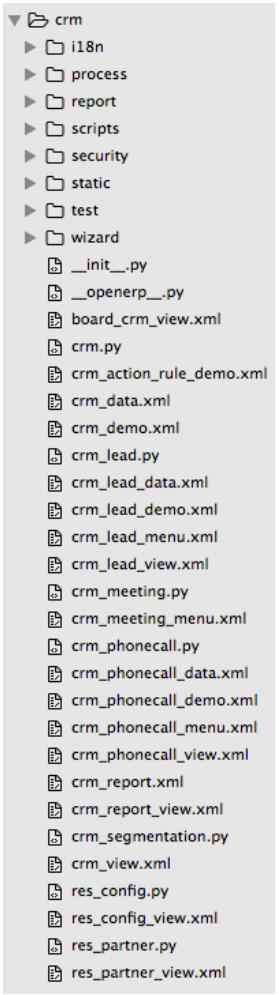


- **__init__.py**:
  - En python cada archivo *.py se denomina módulo.
  - Estos módulos, a la vez, pueden formar parte de paquetes.
  - Un paquete, es una carpeta que contiene archivos *.py. Pero, para que una carpeta pueda ser considerada un paquete, debe contener un archivo de inicio llamado **__init__.py**.
  - En OpenERP consideramos un módulo de OpenERP como un paquete python. Por tanto debemos incluir dicho fichero para importar lo módulos y/o subpaquetes.
  - Si dentro de nuestro módulo OpenERP contenemos carpetas que son subpaquetes (contienen archivos python) debemos añadir este fichero, he importar el subpaquete en el __init__.py del paquete.

ESTRUCTURA DE UN MÓDULO

TIPOS ELEMENTOS
ESTRUCTURA MÓDULO
MODELOS
XML
CSV
YAML
VIEWS
OTROS

Estructura interna:



- **__openerp__.py**:
  - Manifest file
  - Cada módulo de OpenERP debe contener dicho archivo en la raíz del paquete python.
  - Este archivo debe contener un diccionario de python, y es responsable de:
    - ➡ Determinar los archivos a cargar.
    - ➡ Especificar dependencias del módulo.
    - ➡ Incluir metadatos adicionales.
- (Ver __openerp__.py de ejemplo del repositorio)

ESTRUCTURA DE UN MÓDULO

TIPOS ELEMENTOS
ESTRUCTURA MÓDULO
MODELOS
XML
CSV
YAML
VIEWS
OTROS

- Modelos:
  - Todo objeto de OpenERP es una instancia de un modelo:
    - ➡ invoices, partners, menuitems, actions, reports…
  - En OpenERP, un modelo lo consideramos una clase de python en la que heredaremos de:
    - ➡ orm.Model: regular database-persisted models.
    - ➡ orm.TransientModel: for temporary data, stored in the database but automatically vaccuumed every so often
    - ➡ orm.AbstractModel: for abstract super classes meant to be shared by multiple _inheriting classes (usually Models or TransientModels)

ESTRUCTURA DE UN MÓDULO

TIPOS ELEMENTOS
ESTRUCTURA MÓDULO
MODELOS
XML
CSV
YAML
VIEWS
OTROS

- Modelos
  - Todo objeto de OpenERP es una instancia de un modelo:
    - invoice, partner, actions, reports...
  - En OpenERP consideramos una clase de python en la que heredaremos de:
    - orm.Model: for persistent models.
    - orm.TransientModel: for temporary data, stored but automatically vaccuumed every so often
    - orm.AbstractModel: can be shared by multiple _inheriting classes (usually Models or TransientModels)

Por convención

- Nombres en minúsculas
- Separados por _
- Deben comenzar por el nombre del módulo al que pertenecen

account_chart_template

account_tax_code_template

account_fiscalyear

account_fiscal_position_template

ESTRUCTURA DE UN MÓDULO

TIPOS ELEMENTOS
ESTRUCTURA MÓDULO
MODELOS
XML
CSV
YAML
VIEWS
OTROS

- Archivos XML:
  - Son utilizados para inicializar o actualizar datos en la base de datos cuando el módulo es instalado o actualizado.
  - Podemos encontrar XML de varios tipos:
    - *_data.xml
    - *_demo.xml
    - *_workflow.xml
    - *_report.xml
    - *_view.xml
    - *_menu.xml
    - *_security.xml

TIPOS ELEMENTOS
ESTRUCTURA MÓDULO
MODELOS
XML
CSV
YAML
VIEWS
OTROS

ESTRUCTURA DE UN MÓDULO

- Archivos XML:
  - La estructura básica de un XML en OpenERP:

```xml
<?xml version="1.0" encoding="utf-8"?>
<openerp>
    <data>
        <record model="object_model_name" id="object_xml_id">
            <field name="field1">value1</field>
            <field name="field2">value2</field>
        </record>
        <record model="object_model_name2" id="object_xml_id2">
            <field name="field1" ref="module.object_xml_id"/>
            <field name="field2" eval="ref('module.object_xml_id')"/>
        </record>
    </data>
</openerp>
```

  - Define un nuevo registro en un modelo de OpenERP específico.
  - Para ello debemos especificarle:
    - @model (**mandatory**): Nombre del modelo al que pertenece al ser creado/insertado.
    - @id (**opcional…pero recomendable**): ID **ÚNICO (por módulo)** del registro, permite referenciar a dicho registro para el resto de archivos, ya sean del mismo módulo o de otro distinto.
      - Para referenciar el id de otro registro lo haremos mediante el field/@ref o por la función ref()
      - En caso de referenciar a un id de otro módulo anteponemos el nombre del módulo seguido de un punto y el id.

ESTRUCTURA DE UN MÓDULO

TIPOS ELEMENTOS
ESTRUCTURA MÓDULO
MODELOS
XML
CSV
YAML
VIEWS
OTROS

- Archivos XML:
  - Por lo general un registro contiene múltiples “fields” tags, especificando los valores a guardar cuando se crea/modifica.
  - Los fields que no especifiquemos se guardarán en el registro con el valor por defecto especificado en el Modelo de la instancia a crear.
    - En caso de ser un field required, y no especificarse ni tener default, la aplicación lanzará una excepción.
  - La declaración de un field es muy sencilla:
    - Declaramos el tag field seguido del atributo @name que corresponde al nombre del field a introducir.
    - El valor del field lo indicamos en el cuerpo del mismo.

```
<field name=“res_model”>res_partner</field>
```

```
<field name=“partner_id” />
```

TIPOS ELEMENTOS
ESTRUCTURA MÓDULO
MODELOS
XML
CSV
YAML
VIEWS
OTROS

ESTRUCTURA DE UN MÓDULO

- Archivos XML:
  - Además del atributo name podemos encontrar otros que pueden procesar el cuerpo del field, o modificar este por completo:
    - @name (**mandatory**)
      - Nombre del field que contiene el record del model.
    - @type (optional)
      - Puede ser:
        - char, int, float, list, tuple, xml, html, file or base64.
      - Convierte el cuerpo del field en el tipo especificado.

```
    <record id="view_partner_bank_form_inherit" model="ir.ui.view">
        <field name="name">Partner Bank Accounts - Journal</field>
        <field name="model">res.partner.bank</field>
        <field name="inherit_id" ref="base.view_partner_bank_form"/>
        <field name="arch" type="xml">
            <group name="bank" position="after">
                <group name="accounting" col="2" colspan="2" attrs="{'invisible': [('company_id','=', False)]}">
                    <separator string="Accounting Information" colspan="2"/>
                    <field name="journal_id"/>
                    <field name="currency_id" groups="base.group_multi_currency"/>
                </group>
            </group>
        </field>
    </record>
```

TIPOS ELEMENTOS
ESTRUCTURA MÓDULO
MODELOS
XML
CSV
YAML
VIEWS
OTROS

ESTRUCTURA DE UN MÓDULO

- @model
  - Model al que hacemos referencia cuando especificamos @ref o @search.
  - En caso de @search este atributo es obligatorio.
- @eval
  - Expresión python a evaluar para obtener el valor a guardar en el record.
- @ref
  - Link a otro record mediante el ID. En caso de hacer referencia a otro módulo lo debemos especificar indicando “nombre_modulo”.”ID”
- @search
  - Expresión python a evaluar para obtener un domain dentro del @model.

- data -> noupdate=“1” && noupdate=“0”
  - Se utiliza para indicar que los records del xml definido deben ser actualizados cuando el módulo se actualiza.
- record -> forcecreate=“True”
  - Por defecto es True y determina si un record que ha sido borrado debe ser re-creado durante la actualización del módulo **(incluso si hablamos de un update=“0”)**

TIPOS ELEMENTOS
ESTRUCTURA MÓDULO
MODELOS
XML
CSV
YAML
VIEWS
OTROS

ESTRUCTURA DE UN MÓDULO

- Archivos XML:
  - Ejemplos: addons/account/data/account_data.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<openerp>
    <data noupdate="1">
```

```xml
<record id="account_payment_term_immediate" model="account.payment.term">
    <field name="name">Immediate Payment</field>
    <field name="note">Immediate Payment</field>
</record>

<record id="account_payment_term_line_immediate" model="account.payment.term.line">
    <field name="value">balance</field>
    <field eval="0" name="days"/>
    <field eval="0" name="days2"/>
    <field eval="account_payment_term_immediate" name="payment_id"/>
</record>
```

```xml
<record forcecreate="True" id="decimal_payment" model="decimal.precision">
    <field name="name">Payment Term</field>
    <field name="digits">6</field>
</record>
```

ESTRUCTURA DE UN MÓDULO

TIPOS ELEMENTOS
ESTRUCTURA MÓDULO
MODELOS
XML
CSV
YAML
VIEWS
OTROS

- Archivos XML:
  - Ejemplos: addons/account/account_bank_view.xml

```
<record id="view_partner_bank_form_inherit" model="ir.ui.view">
    <field name="name">Partner Bank Accounts - Journal</field>
    <field name="model">res.partner.bank</field>
    <field name="inherit_id" ref="base.view_partner_bank_form"/>
    <field name="arch" type="xml">
        <group name="bank" position="after">
            <group name="accounting" col="2" colspan="2" attrs="{'invisible': [('company_id','=', False)]}">
                <separator string="Accounting Information" colspan="2"/>
                <field name="journal_id"/>
                <field name="currency_id" groups="base.group_multi_currency"/>
            </group>
        </group>
    </field>
</record>
```

TIPOS ELEMENTOS
ESTRUCTURA MÓDULO
MODELOS
XML
CSV
YAML
VIEWS
OTROS

ESTRUCTURA DE UN MÓDULO

- Archivos XML:
  - Ejemplos: addons/account/demo/account_demo.xml

```
<record id="data_fiscalyear" model="account.fiscalyear">
    <field eval="'Fiscal Year X '+time.strftime('%Y')" name="name"/>
    <field eval="'FY'+time.strftime('%Y')" name="code"/>
    <field eval="time.strftime('%Y')+'-01-01'" name="date_start"/>
    <field eval="time.strftime('%Y')+'-12-31'" name="date_stop"/>
    <field name="company_id" ref="base.main_company"/>
</record>
```

```
<record id="base.user_demo" model="res.users">
    <field name="groups_id" eval="[(4,ref('account.group_account_user'))]"/>
</record>
```

ESTRUCTURA DE UN MÓDULO

TIPOS ELEMENTOS
ESTRUCTURA MÓDULO
MODELOS
XML
CSV
YAML
VIEWS
OTROS

- Archivos XML:
  - Ejemplos: addons/account/demo/account_minimal.xml

```
<record forcecreate="True" id="property_account_receivable" model="ir.property">
    <field name="name">property_account_receivable</field>
    <field name="fields_id" search="[('model','=','res.partner'),('name','=','property_account_receivable')]"/>
    <field eval="'account.account,'+str(a_recv)" name="value"/>
    <field name="company_id" ref="base.main_company"/>
</record>
<record forcecreate="True" id="property_account_payable" model="ir.property">
    <field name="name">property_account_payable</field>
    <field name="fields_id" search="[('model','=','res.partner'),('name','=','property_account_payable')]"/>
    <field eval="'account.account,'+str(a_pay)" name="value"/>
    <field name="company_id" ref="base.main_company"/>
</record>
<record forcecreate="True" id="property_account_position" model="ir.property">
    <field name="name">property_account_position</field>
    <field name="fields_id" search="[('model','=','res.partner'),('name','=','property_account_position')]"/>
    <field eval="False" name="value"/>
    <field name="company_id" ref="base.main_company"/>
</record>
```

ESTRUCTURA DE UN MÓDULO

TIPOS ELEMENTOS
ESTRUCTURA MÓDULO
MODELOS
XML
CSV
YAML
VIEWS
OTROS

- Archivos XML:
  - function tag:
    - Se utiliza para llamar a una función de la aplicación **DESPUÉS** instalar/actualizar un módulo.
    - Podemos pasarle argumentos:
    - Ejemplo: addons/account/account_unit_test.xml

```
<function model="account.invoice" name="pay_and_reconcile">
    <!-- ids -->                    <value eval="[ref('test_invoice_1')]"/>
    <!-- pay_amount -->             <value eval="1850"/>
    <!-- pay_account_id -->         <value eval="ref('cash')"/>
    <!-- period_id -->              <value eval="ref('account.period_' + str(int(time.strftime('%m'))))"/>
    <!-- pay_journal_id -->         <value eval="ref('bank_journal')"/>
    <!-- writeoff_acc_id -->        <value eval="ref('cash')"/>
    <!-- writeoff_period_id -->     <value eval="ref('account.period_' + str(int(time.strftime('%m'))))"/>
    <!-- writeoff_journal_id -->    <value eval="ref('bank_journal')"/>
    <!-- context -->                <value eval="{}"/>
    <!-- name -->                   <value eval="str('Payment from ASUStek')"/>
</function>
```

ESTRUCTURA DE UN MÓDULO

TIPOS ELEMENTOS
ESTRUCTURA MÓDULO
MODELOS
XML
CSV
YAML
VIEWS
OTROS

- Archivos CSV:
  - La estructura básica de un CSV en OpenERP:

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
access_crm_segmentation_user,crm.segmentation user,model_crm_segmentation,base.group_sale_salesman,1,0,0,0
access_crm_segmentation_line_user,crm.segmentation.line user,model_crm_segmentation_line,base.group_sale_salesman,1,0,0,0
access_crm_segmentation,crm.segmentation,model_crm_segmentation,base.group_sale_manager,1,1,1,1
```

  - El OSV se encarga de introducir los records definidos en el CSV mediante el procedimiento import_data()
  - El ORM se encargará de realizar la conexión entre las relaciones dependiendo del nombre definido en las columnas:
    - **id** identificador para las relaciones
    - **many2one_field** conexión utilizando el método name_search()
    - **many2one_field:id** conexión utilizando el id de los XML
    - **many2one_field.id** conexión utilizando el id introducido en ddbb.
    - **many2many_field** conexión utilizando el método name_seach(). Para varios valores, los separamos por comas.
    - **many2many_field:id** conexión utilizando el id de los XML. Para varios valores, los separamos por comas.
    - **many2many_field.id** conexión utilizando el id introducido en ddbb. Para varios valores, los separamos por comas.
    - **one2many_field/field** Crea una relación one2many y guarda el valor de field

ESTRUCTURA DE UN MÓDULO

TIPOS ELEMENTOS
ESTRUCTURA MÓDULO
MODELOS
XML
CSV
YAML
VIEWS
OTROS

- Archivos YAML:

Wikipedia

YAML es un formato de serialización de datos legible por humanos inspirado en lenguajes como XML, C, Python, Perl, así como el formato para correos electrónicos especificado por el RFC 2822. YAML fue propuesto por Clark Evans en 2001, quien lo diseñó junto a Ingy döt Net y Oren Ben-Kiki.

YAML es un acrónimo recursivo que significa "YAML Ain't Another Markup Language (en castellano, "YAML no es otro lenguaje de marcado").

- Normalmente se utiliza para pruebas funcionales.
- Carpeta tests/…

ESTRUCTURA DE UN MÓDULO

TIPOS ELEMENTOS
ESTRUCTURA MÓDULO
MODELOS
XML
CSV
YAML
VIEWS
OTROS

- Views:
  - Las vistas son la forma en la que se presentan los datos de los modelos en la parte del cliente. En estas, indicamos al navegador el diseño con el que vamos a mostrar los datos.
  - Forman una jerarquía. Un mismo modelo puede tener varias vistas (sean o no del mismo tipo). La vista a utilizarse se determinará en función de la prioridad con la que las diseñemos.
  - Por defecto, la primera vista definida de cada tipo será utilizada por defecto para dicho tipo
  - Las vistas se diseñan en XML.
  - En caso de no definir una vista para un modelo, OpenERP nos generará una vista a corde a como hemos creado las _columns de dicho modelo.

TIPOS ELEMENTOS
ESTRUCTURA MÓDULO
MODELOS
XML
CSV
YAML
VIEWS
OTROS

ESTRUCTURA DE UN MÓDULO

- Views:
  - Heredar. Podemos modificar/añadir/borrar cualquier campo de una vista creada por otro módulo mediante el id declarado en esta.
  - Las vistas son lanzadas por actions (más adelante…)
  - Tipos de vistas:
    - Form
    - Tree
      - list: Caso particular de las vistas tree.
    - Kanban
    - Calendar
    - Gantt
    - Graph
    - Search

ESTRUCTURA DE UN MÓDULO

TIPOS ELEMENTOS
ESTRUCTURA MÓDULO
MODELOS
XML
CSV
YAML
VIEWS
OTROS

- Otros…
  - Reports
  - Workflows
  - i18n