Data Files

Odoo is greatly data-driven, and a big part of modules definition is thus the definition of the various records it manages: UI (menus and views), security (access rights and access rules), reports and plain data are all defined via records.

Structure

The main way to define data in Odoo is via XML data files: The broad structure of an XML data file is the following:

Any number of operation elements within the root element `odoo`

```
<!-- the root elements of the data file -->
<?xml version="1.0" encoding="UTF-8"?>
<odoo>
  <operation/>
  ...
</odoo>
```

Data files are executed sequentially, operations can only refer to the result of operations defined previously

> If the content of the data file is expected to be applied only once, you can specify the odoo flag `noupdate` set to 1. If part of the data in the file is expected to be applied once, you can place this part of the file in a <data noupdate="1"> domain.
>
> ```
> <odoo>
>     <data noupdate="1">
>         <!-- Only loaded when installing the module (odoo-bin -i module) -->
>         <operation/>
>     </data>
>
>     <!-- (Re)Loaded at install and update (odoo-bin -i/-u) -->
>     <operation/>
> </odoo>
> ```

Core operations

`record`

`record` appropriately defines or updates a database record, it has the following attributes:

`model` (required)

name of the model to create (or update)

id

the external identifier (../glossary.html#term-external-identifier) for this record. It is strongly recommended to provide one

for record creation, allows subsequent definitions to either modify or refer to this record

for record modification, the record to modify

context

context to use when creating the record

forcecreate

in update mode whether the record should be created if it doesn't exist

Requires an external id (../glossary.html#term-external-id), defaults to `True`.

field

Each record can be composed of `field` tags, defining values to set when creating the record.

A `record` with no `field` will use all default values (creation) or do nothing (update).

A `field` has a mandatory `name` attribute, the name of the field to set, and various methods to define the value itself:

Nothing

if no value is provided for the field, an implicit `False` will be set on the field. Can be used to clear a field, or avoid using a default value for the field.

search

for relational fields (orm.html#reference-fields-relational), should be a domain (orm.html#reference-orm-domains) on the field's model.

Will evaluate the domain, search the field's model using it and set the search's result as the field's value. Will only use the first result if the field is a `Many2one` (orm.html#odoo.fields.Many2one)

ref

if a `ref` attribute is provided, its value must be a valid external id (../glossary.html#term-external-id), which will be looked up and set as the field's value.

Mostly for `Many2one` (orm.html#odoo.fields.Many2one) and `Reference` (orm.html#odoo.fields.Reference) fields

type

if a `type` attribute is provided, it is used to interpret and convert the field's content. The field's content can be provided through an external file using the `file` attribute, or through the node's body.

Available types are:

xml , html
extracts the **field** 's children as a single document, evaluates any external id (../glossary.html#term-external-id) specified with the form **%(external_id)s** . **%%** can be used to output actual % signs.

file
ensures that the field content is a valid file path in the current model, saves the pair ***module,path*** as the field value

char
sets the field content directly as the field's value without alterations

base64
base64 (https://tools.ietf.org/html/rfc3548.html#section-3)-encodes the field's content, useful combined with the **file** *attribute* to load e.g. image data into attachments

int
converts the field's content to an integer and sets it as the field's value

float
converts the field's content to a float and sets it as the field's value

list , tuple
should contain any number of **value** elements with the same properties as **field** , each element resolves to an item of a generated tuple or list, and the generated collection is set as the field's value

eval
for cases where the previous methods are unsuitable, the **eval** attributes simply evaluates whatever Python expression it is provided and sets the result as the field's value.

The evaluation context contains various modules (**time** , **datetime** , **timedelta** , **relativedelta**), a function to resolve external identifiers (../glossary.html#term-external-identifiers) (**ref**) and the model object for the current field if applicable (**obj**)

delete

The **delete** tag can remove any number of records previously defined. It has the following attributes:

model (required)
the model in which a specified record should be deleted

id
the external id (../glossary.html#term-external-id) of a record to remove

search
a domain (orm.html#reference-orm-domains) to find records of the model to remove

`id` and `search` are exclusive

`function`

The `function` tag calls a method on a model, with provided parameters. It has two mandatory parameters `model` and `name` specifying respectively the model and the name of the method to call.

Parameters can be provided using `eval` (should evaluate to a sequence of parameters to call the method with) or `value` elements (see `list` values).

```
<odoo>
    <data noupdate="1">
        <record name="partner_1" model="res.partner">
            <field name="name">Odude</field>
        </record>

        <function model="res.partner" name="send_inscription_notice"
            eval="[[ref('partner_1'), ref('partner_2')]]"/>

        <function model="res.users" name="send_vip_inscription_notice">
            <function eval="[[('vip','=',True)]]" model="res.partner" name="search"/>
        </function>
    </data>

    <record id="model_form_view" model="ir.ui.view">

    </record>
</odoo>
```

Shortcuts

Because some important structural models of Odoo are complex and involved, data files provide shorter alternatives to defining them using record tags:

`menuitem`

Defines an `ir.ui.menu` record with a number of defaults and fallbacks:

`parent`

If a `parent` attribute is set, it should be the external id (../glossary.html#term-external-id) of an other menu item, used as the new item's parent

If no `parent` is provided, tries to interpret the `name` attribute as a `/`-separated sequence of menu names and find a place in the menu hierarchy. In that interpretation, intermediate menus are automatically created

Otherwise the menu is defined as a "top-level" menu item (*not* a menu with no parent)

name

If no **name** attribute is specified, tries to get the menu name from a linked action if any. Otherwise uses the record's **id**

groups

A **groups** attribute is interpreted as a comma-separated sequence of external identifiers (../glossary.html#term-external-identifiers) for **res.groups** models. If an external identifier (../glossary.html#term-external-identifier) is prefixed with a minus (**-**), the group is *removed* from the menu's groups

action

if specified, the **action** attribute should be the external id (../glossary.html#term-external-id) of an action to execute when the menu is open

id

the menu item's external id (../glossary.html#term-external-id)

template

Creates a QWeb view (views.html#reference-views-qweb) requiring only the **arch** section of the view, and allowing a few *optional* attributes:

id

the view's external identifier (../glossary.html#term-external-identifier)

name , inherit_id , priority

same as the corresponding field on **ir.ui.view** (nb: **inherit_id** should be an external identifier (../glossary.html#term-external-identifier))

primary

if set to **True** and combined with a **inherit_id** , defines the view as a primary

groups

comma-separated list of group external identifiers (../glossary.html#term-external-identifiers)

page

if set to **"True"** , the template is a website page (linkable to, deletable)

optional

enabled or **disabled** , whether the view can be disabled (in the website interface) and its default status. If unset, the view is always enabled.

CSV data files

XML data files are flexible and self-descriptive, but very verbose when creating a number of simple records of the same model in bulk.

For this case, data files can also use csv (https://en.wikipedia.org/wiki/Comma-separated_values), this is often the case for access rights (security.html#reference-security-acl):

the file name is ***model_name*.csv**

the first row lists the fields to write, with the special field **id** for external identifiers (../glossary.html#term-external-identifiers) (used for creation or update)

each row thereafter creates a new record

Here's the first lines of the data file defining US states **res.country.state.csv**

```
"id","country_id:id","name","code"
state_au_1,au,"Australian Capital Territory","ACT"
state_au_2,au,"New South Wales","NSW"
state_au_3,au,"Northern Territory","NT"
state_au_4,au,"Queensland","QLD"
state_au_5,au,"South Australia","SA"
state_au_6,au,"Tasmania","TAS"
state_au_7,au,"Victoria","VIC"
state_au_8,au,"Western Australia","WA"
state_us_1,us,"Alabama","AL"
state_us_2,us,"Alaska","AK"
state_us_3,us,"Arizona","AZ"
state_us_4,us,"Arkansas","AR"
state_us_5,us,"California","CA"
state_us_6,us,"Colorado","CO"
```

rendered in a more readable format:

id	country_id:id	name	code
state_au_1	au	Australian Capital Territory	ACT
state_au_2	au	New South Wales	NSW
state_au_3	au	Northern Territory	NT
state_au_4	au	Queensland	QLD
state_au_5	au	South Australia	SA
state_au_6	au	Tasmania	TAS
state_au_7	au	Victoria	VIC
state_au_8	au	Western Australia	WA
state_us_1	us	Alabama	AL
state_us_2	us	Alaska	AK
state_us_3	us	Arizona	AZ
state_us_4	us	Arkansas	AR
state_us_5	us	California	CA

id	country_id:id	name	code
state_us_6	us	Colorado	CO

For each row (record):

the first column is the external id (../glossary.html#term-external-id) of the record to create or update

the second column is the external id (../glossary.html#term-external-id) of the country object to link to (country objects must have been defined beforehand)

the third column is the **`name`** field for **`res.country.state`**

the fourth column is the **`code`** field for **`res.country.state`**