

Mémento technique OpenERP en français

Open Source RAD avec OpenERP 7,0

Par **Thierry Godin** 

Date de publication : 19 novembre 2013

Dernière mise à jour : 10 décembre 2013

CONFIRMÉ

Cet article est une traduction française du Memento Technique d'OpenERP v7.0
Version du fichier original : 08/11/2013.

Avec l'aimable autorisation de la société OpenERP SA.

I - Préambule.....	3
II - Installer OpenERP.....	3
II-A - Packages d'installation.....	3
II-B - Installer depuis les sources.....	4
II-C - Création de la base de données.....	4
III - Construire un module OpenERP : Idea (Idée).....	4
III-A - Contexte.....	4
III-B - Composition d'un module.....	4
III-C - Structure typique d'un module.....	5
III-D - Service de mappage objet-relationnel.....	5
III-E - Types de champs de l'ORM.....	7
III-E-1 - Noms de champs spéciaux/réservés.....	9
III-F - Travailler avec l'ORM.....	10
IV - Construire l'interface du module.....	13
IV-A - Structure XML commune.....	13
IV-B - Syntaxe CSV commune.....	14
IV-C - Menus et actions.....	15
IV-C-1 - Déclaration d'une action.....	15
IV-C-2 - Déclaration d'un menu.....	15
V - Vues et héritage.....	16
V-A - Vues formulaires (pour voir/modifier les enregistrements).....	16
V-A-1 - Les éléments de formulaire.....	17
V-B - Vues dynamiques.....	18
V-C - Vues listes et listes d'arborescence hiérarchique.....	18
V-D - Vues fiches Kanban.....	19
V-E - Vues calendrier.....	19
V-F - Diagrammes de Gantt.....	19
V-G - Vues diagrammes (graphes).....	20
V-H - Vues de recherche.....	20
V-I - Héritage des vues.....	21
VI - Rapports.....	21
VI-A - Différents formats de rapports.....	21
VI-B - Les expressions utilisées dans les modèles de rapport OpenERP.....	22
VII - Les Flux de travail (Workflows).....	23
VII-A - Déclaration d'un flux de travail.....	23
VII-B - Activités du flux de travail (nœuds).....	23
VII-C - Transitions du flux de travail (bords).....	24
VIII - Sécurité.....	25
VIII-A - Mécanismes de contrôle d'accès basés sur le groupe.....	25
VIII-B - Roles.....	25
IX - Les assistants (Wizards).....	25
IX-A - Les modèles d'assistant (TransientModel).....	25
IX-B - Vues assistant.....	26
IX-C - Exécution de l'assistant.....	26
X - WebServices - XML-RPC.....	26
XI - Optimisation des performances.....	27
XII - Communauté/contribution.....	28
XIII - Licence.....	28
XIV - Remerciements.....	29



I - Préambule

OpenERP est une suite moderne de d'Applications Métiers, publiée sous la licence AGPL qui comprend les modules CRM, RH, ventes, comptabilité, fabrication, gestion d'entrepôts, gestion de projets, et plus encore. Il est basé sur un système modulaire, une plate-forme Rapid Application Development (RAD) évolutive et intuitive écrite en Python.

OpenERP dispose d'une boîte à outils complète et modulaire pour construire rapidement des applications : Object-Relationship Mapping (ORM) intégré, un patron Modèle-Vue-Contrôleur (MVC), un système de génération de rapport, l'internationalisation automatisée, et bien plus encore.

Python est un langage de programmation dynamique de haut niveau, idéal pour RAD, alliant la puissance avec une syntaxe claire, et un noyau maintenu petit par sa conception.

Astuce : Liens utiles

- Le site Web principal OpenERP et téléchargements :  www.openerp.com
- Documentation fonctionnelle et technique :  doc.openerp.com
- Ressources communautaires :

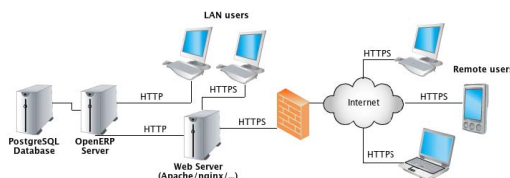


www.openerp.com/community

- Le serveur d'intégration permanent :  runbot.openerp.com
- Apprendre Python :  doc.python.org

II - Installer OpenERP


OpenERP est distribué sous forme de paquets/installeurs pour la plupart des plates-formes, mais peut également être installé à partir des sources sur n'importe quelle plate-forme.



L'architecture d'OpenERP

OpenERP utilise le paradigme client-serveur bien connu : le client s'exécute comme une application JavaScript dans votre navigateur, se connectant au serveur en utilisant le protocole JSON-RPC sur HTTP(S). Des clients peuvent être facilement écrits selon vos besoins et se connecter au serveur en utilisant XML-RPC ou JSON-RPC.


Astuce : Procédure d'installation

- La procédure d'installation OpenERP est susceptible d'évoluer (dépendances et ainsi de suite), alors assurez-vous de toujours consulter la documentation spécifique (fournie avec les packages et sur le site Internet) pour les dernières procédures.
- Voir :  <http://doc.openerp.com/v7.0/install>

II-A - Packages d'installation

- **Windows** : installateur tout-en-un.
- **Linux** : packages tout-en-un pour les systèmes basés sur Debian (*.deb), et Red-Hat (*.rpm).
- **Mac** : pas d'installateur tout-en-un, il faut installer depuis les sources.

II-B - Installer depuis les sources

Il y a deux possibilités : utiliser une archive fournie sur le site, ou obtenir directement les sources à l'aide de  **Bazaar** (système de contrôle de version). Vous devez également installer les dépendances nécessaires (PostgreSQL et quelques bibliothèques Python - voir la documentation sur  **doc.openerp.com**).



Astuce : Compilation

OpenERP étant basé sur Python, aucune compilation n'est nécessaire.

Procédure de commandes Bazaar typique (sur base Debian Linux)

```
1. $ sudo apt-get install bzip2
   # Installer Bazaar (version control software)
2. $ bzip2 cat -d lp:~openerp-dev/openerp-tools/trunk setup.sh | sh # Récupérer l'Installeur
3. $ make init-v70 # Installer OpenERP 7.0
4. $ make server
   # Démarrer OpenERP Server avec l'interface Web
```

II-C - Création de la base de données

Après le démarrage du serveur, ouvrez **http://localhost:8069** dans votre navigateur préféré. Vous verrez l'écran du gestionnaire de bases de données où vous pouvez créer une nouvelle base de données. Chaque base de données possède ses propres modules et sa propre configuration, et peut être créée en mode démo pour tester une base de données préremplie (ne pas utiliser le mode de démonstration pour une véritable base de données !).

III - Construire un module OpenERP : Idea (Idée)

III-A - Contexte

Les exemples de code utilisés dans ce mémento sont pris à partir d'un module hypothétique d'idées. Le but de ce module serait d'aider les esprits créatifs, qui viennent souvent avec des idées qui ne peuvent pas être réalisées immédiatement, et sont trop facilement oubliées si elles ne sont pas notées quelque part. Ce module pourrait être utilisé pour enregistrer ces idées, les trier et les évaluer.

Note : développement modulaire



OpenERP utilise des modules comme des conteneurs de fonctionnalités, afin de favoriser un robuste développement et leur maintenance. Les modules offrent une isolation des fonctions, un niveau approprié d'abstraction et des modèles évidents MVC.

III-B - Composition d'un module


Un module peut contenir n'importe lequel des éléments suivants :

- **objets métier** : déclarés comme des classes Python qui étendent la classe **osv.Model**. La persistance de ces ressources est entièrement gérée par OpenERP ;
- **données** : les fichiers XML/CSV avec des métadonnées (vues et déclarations des flux de travail), les données de configuration (paramétrage des modules) et des données de démonstration (facultatives, mais recommandées pour tester, par exemple, des échantillons d'idées) ;
- **assistants** : formulaires interactifs utilisés pour aider les utilisateurs, souvent disponibles en actions contextuelles sur les ressources ;
- **rapports** : RML (format XML), MAKO ou OpenOffice modèles de rapports, qui seront fusionnés avec n'importe quel type de données de l'entreprise et généreront du HTML, ODT ou des rapports PDF.

III-C - Structure typique d'un module

Chaque module est contenu dans son propre répertoire, dans le répertoire d'installation du serveur **server/bin/addons**.

Note :

 Vous pouvez déclarer votre propre répertoire addons dans le fichier de configuration d'OpenERP en utilisant l'option **addons_path** (transmis au serveur avec l'option **-c**).

```

5. addons/
6. |- idea/                # Le répertoire du module
7.   |- demo/              # Données de démonstration et tests unitaires
8.   |- i18n/              # Fichiers de traduction
9.   |- report/            # Rapports
10.  |- security/           # Déclaration des groupes et droits d'accès
11.  |- view/              # Vues (formulaires, listes), menus et actions
12.  |- wizard/            # Assistants
13.  |- workflow/          # Flux de travail
14.  |- __init__.py         # Fichier d'initialisation Python (requis)
15.  |- __openerp__.py     # Déclaration du module (requis)
16.  |- idea.py             # Classes Python, les objets du module

```

Le fichier **__init__.py** est le descripteur de module Python, car un module OpenERP est aussi un module Python régulier.

```

__init__.py
17. # Importe tous les fichiers et dossiers qui contiennent du code Python
18. import idea, wizard, report

```

Le fichier **__openerp__.py** est le manifeste du module OpenERP et contient un dictionnaire unique Python avec la déclaration du module : son nom, les dépendances, la description et la composition.

```

__openerp__.py
19. {
20.     'name' : 'Idea',
21.     'version' : '1.0',
22.     'author' : 'OpenERP',
23.     'description' : 'Ideas management module',
24.     'category' : 'Enterprise Innovation',
25.     'website' : 'http://www.openerp.com',
26.     'depends' : ['base'], # liste des dépendances conditionnant l'ordre de démarrage
27.     'data' : [ # les fichiers de données à charger lors de l'installation du module
28.         'security/groups.xml', # toujours charger les groupes en premier!
29.         'security/ir.model.access.csv', # charger les droits d'accès après les groupes
30.         'workflow/workflow.xml',
31.         'view/views.xml',
32.         'wizard/wizard.xml',
33.         'report/report.xml',
34.     ],
35.     'demo' : ['demo/demo.xml'], # données de démo (pour les tests unitaires)
36. }

```

III-D - Service de mappage objet-relationnel

Élément-clé d'OpenERP, l'**ORM** est une couche de mappage objet-relationnel complet, qui permet aux développeurs de ne pas avoir à écrire la plomberie SQL de base. Les objets métiers sont déclarés comme les classes Python qui héritent de la classe **osv.Model**, ce qui les rend magiquement persistants par la couche ORM.

Des attributs prédéfinis sont utilisés dans la classe Python pour spécifier les caractéristiques d'un objet métier pour l'ORM :

idea.py

```

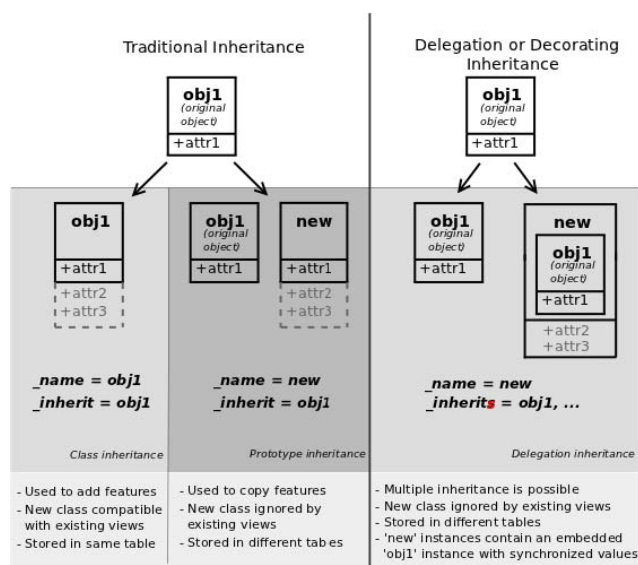
37. from osv import osv, fields
38. class idea(osv.Model):
39.     _name = 'idea.idea'
40.     _columns = {
41.         'name': fields.char('Title', size=64, required=True, translate=True),
42.         'state': fields.selection([('draft', 'Draft'),
43.                                   ('confirmed', 'Confirmed')], 'State', required=True, readonly=True),
44.         # Description est en lecture seule quand elle n est pas en brouillon!
45.         'description': fields.text('Description', readonly=True,
46.                                   states={ 'draft': [ ('readonly', False)] } ),
47.         'active': fields.boolean('Active'),
48.         'invent_date': fields.date('Invent date'),
49.         # par convention, les champs many2one se terminent par '_id'
50.         'inventor_id': fields.many2one('res.partner', 'Inventor'),
51.         'inventor_country_id': fields.related('inventor_id', 'country',
52.                                              readonly=True, type='many2one',
53.                                              relation='res.country', string='Country'),
54.         # par convention, les champs *2many se terminent par '_ids'
55.         'vote_ids': fields.one2many('idea.vote', 'idea_id', 'Votes'),
56.         'sponsor_ids': fields.many2many('res.partner', 'idea_sponsor_rel',
57.                                         'idea_id', 'sponsor_id', 'Sponsors'),
58.         'score': fields.float('Score', digits=(2,1)),
59.         'category_id' = fields.many2one('idea.category', 'Category'),
60.     }
61. _defaults = {
62.     'active': True, # les idées sont actives par défaut
63.     'state': 'draft', # les idées sont à l'état de brouillon par défaut
64. }
65. def _check_name(self, cr, uid, ids):
66.     for idea in self.browse(cr, uid, ids):
67.         if 'spam' in idea.name: return False # On ne peut pas créer une idée avec spam!
68.     return True
69. _sql_constraints = [('name_uniq', 'unique(name)', 'Ideas must be unique!')]
70. _constraints = [(_check_name, 'Please avoid spam in ideas !', ['name'])]

```

Attributs prédéfinis de osv.osv pour les objets métier

_name (requis)	Nom de l'objet métier, en notation pointée (dans le module d'espace de nom)
_columns (requis)	Dictionnaire {nom du champ → déclaration du champ}
_defaults	Dictionnaire : {nom du champ → littéral ou une fonction fournissant la valeur par défaut} <i>_defaults['name'] = lambda self, cr, uid, context: 'eggs'</i>
_auto	Si <i>True</i> (par défaut) l'ORM va créer la table de base de données - Mettre à <i>False</i> pour créer votre propre table/vue dans la méthode <i>init()</i>
_inherit	_name : nom de l'objet métier parent (par héritage)
_inherits	Pour la décoration de l'héritage : dictionnaire mappant le nom (_name) de l'objet(s) métier parent avec les noms des champs de clé étrangère correspondante à utiliser
_constraints	Liste des tuples qui définissent des contraintes Python, sous la forme (<i>func_name, message, champs</i>) (→70)
_sql_constraints	Liste des tuples qui définissent des contraintes SQL, sous la forme (<i>nom, sql_def, message</i>) (→69)
_log_access	Si <i>True</i> (par défaut), 4 champs (<i>create_uid, create_date, write_uid, write_date</i>) seront utilisés pour identifier les opérations au

	niveau des enregistrements, accessible via la fonction <code>perm_read()</code>
<code>_order</code>	Nom du champ utilisé pour trier les enregistrements dans des listes (par défaut : 'id')
<code>_rec_name</code>	Champ alternatif à utiliser comme nom, utilisé par <code>name_get()</code> (par défaut : 'name')
<code>_sql</code>	Code SQL pour créer la table/vue de cet objet (si <code>_auto</code> est <code>False</code>) - Peut être remplacé par l'exécution SQL dans la méthode <code>init()</code>
<code>_table</code>	Nom de la table SQL à utiliser (par défaut : <code>'_' + _name</code> avec des points remplacés par des underscores '_')



Héritage


III-E - Types de champs de l'ORM

Les objets peuvent contenir trois types de champs : simples, relationnels et fonctionnels.

Les types simples sont des nombres entiers, flottants, booléens, chaînes, etc.

Les champs relationnels représentent les relations entre les objets (one2many, many2one, many2many). Les champs fonctionnels ne sont pas stockés dans la base de données, mais calculés à la volée comme des fonctions Python. Des exemples pertinents dans la classe **idea** ci-dessus sont indiqués avec les numéros de ligne correspondants (→ XX, XX).

Types de champs de l'ORM	
<ul style="list-style-type: none"> <code>string</code> : étiquette de champ (obligatoire) <code>required</code> : <code>True</code> si obligatoire <code>readonly</code> : <code>True</code> si non modifiable <code>help</code> : info-bulle <code>select</code> : <code>True</code> pour créer un index de base de données sur cette colonne 	<ul style="list-style-type: none"> <code>context</code> : dictionnaire avec contexte si les paramètres sont obligatoires (pour les champs relationnels) <code>change_default</code> : <code>True</code> si le champ doit être utilisable comme condition pour les valeurs par défaut de clients

	<ul style="list-style-type: none"> states : changements dynamiques aux attributs ordinaires de ce champ sur la base du champ state
Champs simples	
boolean (...) integer (...) date (...) datetime (...) time (...)	'active' : fields. boolean ('Active'), 'priority' : fields. integer ('Priority'), 'start_date' : fields. date ('Start Date'),
char (string,size,translate=False...) text (string, translate=False...) Champs de texte	<ul style="list-style-type: none"> translate : True si les valeurs de champ peuvent être traduites par les utilisateurs, pour les champs de char/text size : taille max facultative pour les champs texte (→ 41,45)
float (string, digits=None...) Valeur décimale	<ul style="list-style-type: none"> digits : tuple (précision, échelle) (→ 58)
selection (values, string...) Champ permettant la sélection entre un ensemble de valeurs prédéfinies	<ul style="list-style-type: none"> values : liste de valeurs (clé tuples) ou fonction retournant une telle liste (obligatoire) (→ 42)
binary (string, filters=None...) Champ pour stocker un fichier ou du contenu binaire.	<ul style="list-style-type: none"> filters : filtres de noms de fichiers en option pour la sélection 'picture' : fields.binary('Picture', filters=*.png,*.gif)
reference (string, selection, size,...) Champ en relation dynamique avec n'importe quel autre objet, associé à un widget assistant	<ul style="list-style-type: none"> selection : nom de modèle (_name) des types d'objets autorisés avec l'étiquette correspondante (même format que le value des champs selection) size : la taille de la colonne de texte utilisée pour stocker (format de stockage est 'model_name, object_id ') 'contact' : fields.reference('Contact', [(('res.partner' , 'Partner'), ('res.partner.contact' , 'Contact'))])
Champs relationnels	
Les attributs communs pris en charge par des champs relationnels	domain : filtre en option sous la forme d'arguments pour la recherche (voir search())
many2one (obj, ondelete='set null'...) (→50) Relation à un objet parent (en utilisant une clé étrangère)	<ul style="list-style-type: none"> obj : _name (nom) de l'objet de destination (obligatoire) ondelete : manipulation de suppression, par exemple 'set null', 'cascade'; consultez la  documentation de PostgreSQL
one2many (obj, field_id...) (→55) Relation virtuelle vers plusieurs objets (inverse de many2one)	<ul style="list-style-type: none"> obj : _name (nom) de l'objet de destination (obligatoire) field_id : nom du champ many2one inverse, c'est-à-dire : clé étrangère correspondante (obligatoire)
many2many (obj, rel, field1, field2...) (→ 56) Relation bidirectionnelle entre plusieurs objets	<ul style="list-style-type: none"> obj : _name (nom) de l'objet de destination (obligatoire) rel : nom optionnel de la table de relation à utiliser (par défaut : autoattribué sur la base des noms de modèles)

	<ul style="list-style-type: none"> field1 : nom du champ dans la table rel stockant l'id de l'objet courant (par défaut : en fonction du modèle) field2 : nom du champ de la table rel stockant l'id de l'objet cible (par défaut : en fonction du modèle)
Champs fonctionnels	
function (fnc, arg=None, fnc_inv=None, fnc_inv_arg=None, type='float', fnc_search=None, obj=None, store=False, multi=False,...) Champ fonctionnel simulant un champ réel, calculé plutôt que stocké <ul style="list-style-type: none"> fnc : fonction pour calculer la valeur du champ (obligatoire) def fnc(self, cr, uid, ids, field_name, arg, context) Retourne un dictionnaire {ids → valeurs} avec les valeurs de type de type fnc_inv : fonction utilisée pour écrire une valeur dans le champ à l'inverse def fnc_inv(obj, cr, uid, id, name, value, fnc_inv_arg, context) type : type de champs simulé (peut être n'importe quel autre type, sauf 'fonction') fnc_search : fonction utilisée pour la recherche dans ce champ def fnc_search (obj, cr, uid, obj, nom, args) Retourne une liste de tuples d'arguments pour search(), par exemple [('id', 'in', [1,3,5])] obj : _name (nom) du modèle du champ simulé s'il s'agit d'un champ relationnel store, multi : mécanismes d'optimisation (voir utilisation dans la section Performance) 	
related (f1, f2,, type='float', ...) Champ raccourci équivalent à la navigation sur des champs reliés <ul style="list-style-type: none"> f1,f2,... : champs reliés pour atteindre la cible (f1 obligatoire) (→51) type : type de champ cible 	
property (obj, type='float', view_load=None, group_name=None, ...) Attribut dynamique avec des droits d'accès spécifiques <ul style="list-style-type: none"> obj : objet (obligatoire) type : type de champ équivalent 	

Astuce : symétrie des champs relationnels



- one2many** ↔ **many2one** sont symétriques
- many2many** ↔ **many2many** sont symétriques lorsque inversés (inverse field1 et field2 si explicite)
- one2many** ↔ **many2one** + **many2one** ↔ **one2many** = **many2many**

III-E-1 - Noms de champs spéciaux/réservés

Quelques noms de champs sont réservés avec un comportement prédéfini dans OpenERP. Certains d'entre eux sont créés automatiquement par le système, et dans ce cas tout champ avec ce nom sera ignoré.

Noms de champs spéciaux/réservés	
id	Identificateur système unique pour l'objet
name	Champ dont la valeur est utilisée pour afficher l'enregistrement dans les listes, etc.

	S'il est manquant, mettre <code>_rec_name</code> pour spécifier un autre champ à utiliser
<code>active</code>	Basculer la visibilité : les enregistrements avec <code>active</code> défini à <code>False</code> sont masqués par défaut
<code>sequence</code>	Définit l'ordre et permet la réorganisation glisser-déposer si visible dans les vues de liste
<code>state</code>	Étapes du cycle de vie de l'objet, utilisée par les attributs <code>states</code>
<code>parent_id</code>	Définit la structure de tableau sur des enregistrements, et permet l'opérateur de <code>child_of</code>
<code>parent_left, parent_right</code>	Utilisé en conjonction avec le signal <code>_parent_store</code> sur l'objet, permet un accès plus rapide à des structures de tableau (voir également section <i>Optimisation Performance</i>)
<code>create_date,</code> <code>create_uid,</code> <code>write_date,</code> <code>write_uid</code>	Utilisé pour enregistrer l'utilisateur créateur, l'utilisateur qui a mis à jour, la date de création et la date de dernière mise à jour de l'enregistrement. Désactivée si le signal <code>_log_access</code> est défini sur <code>False</code> (créé par ORM, ne pas les ajouter)

III-F - Travailler avec l'ORM

Héritant de la classe de `osv.Model`, rend toutes les méthodes de l'ORM disponibles sur des objets métier. Ces méthodes peuvent être appelées sur l'objet lui-même (`self`) au sein de la classe Python (voir les exemples dans le tableau ci-dessous), ou de l'extérieur de la classe en obtenant d'abord une instance via le système `pool` de l'ORM.

Exemple d'utilisation de l'ORM

Exemple d'utilisation de l'ORM

```

72. class idea2(osv.Model):
73.     _inherit = 'idea.idea'
74.     def _score_calc(self, cr, uid, ids, field, arg, context=None):
75.         res = {}
76.         # Cette boucle génère seulement deux requêtes grâce à browse() !
77.         for idea in self.browse(cr, uid, ids, context=context):
78.             sum_vote = sum([v.vote for v in idea.vote_ids])
79.             avg_vote = sum_vote/len(idea.vote_ids)
80.             res[idea.id] = avg_vote
81.         return res
82.     _columns = {
83.         # Remplace le score statique avec une moyenne des scores
84.         'score': fields.function(_score_calc, type='float')
85.     }

```

Méthodes de l'ORM sur les objets <code>osv.Model</code>	
Accesseur générique à OSV	<ul style="list-style-type: none"> <code>self.pool.get('object_name')</code> peut être utilisé pour obtenir un modèle de n'importe quel autre
Les paramètres communs, utilisés par de multiples méthodes	<ul style="list-style-type: none"> <code>uid</code> : identifiant de l'utilisateur qui effectue l'opération <code>ids</code> : <code>ids</code> d'enregistrement sur lesquels on effectue l'opération

	<ul style="list-style-type: none"> context : dictionnaire facultatif de contexte parametres, ex. : { 'lang': 'en_US', ... }
search (cr, uid, domain, offset=0, limit=None, order=None, context=None, count=False) <i>Renvoie : liste des identifiants des enregistrements correspondant aux critères donnés</i>	<ul style="list-style-type: none"> domain : filtre spécifiant des critères de recherche offset : nombre optionnel d'enregistrements à sauter limit : nombre optionnel maximum d'enregistrements à renvoyer order : colonnes optionnelles de tri (par défaut : self_order) count : si True, renvoie seulement le nombre d'enregistrements correspondant aux critères, et non leurs id
	<pre>#Opérateurs: =, ! =, >, >=, <, <=, like, ilike, #in, not in, child_of, parent_left, parent_right #Opérateurs de préfixe: '&' (default), ' ', '!' #Récupère le magasins dont le nom ne contient pas le mot spam #du partenaire 34 ids = self.search(cr, uid, [' ', ('partner_id', '!=', 34), ' ', ('name', 'ilike', 'spam')],, order='partner_id')</pre>
create (cr, uid, values, context=None) <i>Crée un nouvel enregistrement avec la valeur spécifiée</i> <i>Renvoie : id du nouvel enregistrement</i>	<ul style="list-style-type: none"> values : dictionnaire de valeurs de champs
	<pre>idea_id = self.create(cr, uid, { 'name': 'Spam recipe', 'description': 'spam & eggs', 'inventor_id': 45, })</pre>
read (cr, uid, ids, fields=None, context=None) <i>Renvoie : liste des dictionnaires avec des valeurs de champs demandés</i>	<ul style="list-style-type: none"> fields : liste facultative de noms de champs à renvoyer (par défaut : tous les champs)
	<pre>results = self.read(cr, uid, [42,43], ['name', 'inventor_id']) print 'Inventor:', results[0] ['inventor_id']</pre>
read_group (cr, uid, domain, fields, groupby, offset=0, limit=None, orderby=None, context=None) <i>Renvoie : liste des dictionnaires avec des valeurs de champs demandés, regroupés par les champs spécifiés (groupby).</i>	<ul style="list-style-type: none"> domain : filtre de recherche (voir search()) fields : liste de noms de champs à lire groupby : champ ou une liste de champs à grouper offset, limit : voir search() orderby : ordre facultatif pour le résultat
	<pre>> print self.read_group(cr,uid,[], ['score'], ['inventor_id']) [{'inventor_id': (1, 'Administrator'), 'score': 23, # score total 'inventor_id_count': 12, # group count }, {'inventor_id': (3, 'Demo'), 'score': 13, 'inventor_id_count': 7, }]</pre>
write (cr, uid, ids, values, context=None) <i>Met à jour des enregistrements avec les identifiants donnés aux valeurs indiquées.</i>	<ul style="list-style-type: none"> values : dictionnaire de valeurs de champs à mettre à jour
	<pre>self.write(cr, uid, [42,43], { 'name': 'spam & eggs',</pre>

Renvoie : <i>True</i>	<pre>'partner_id': 24, })</pre>
copy (cr, uid, id, defaults, context=None) <i>Duplique l'enregistrement de l'id donné en le mettant à jour avec des valeurs par défaut.</i> Renvoie : <i>True</i>	<ul style="list-style-type: none"> defaults : dictionnaire des valeurs de champs à modifier dans les valeurs copiées lors de la création de l'objet dupliqué
unlink (cr, uid, ids, context=None) <i>Supprime les enregistrements des id spécifiés</i> Renvoie : <i>True</i>	<pre>self.unlink(cr, uid, [42, 43])</pre>
browse (cr, uid, ids, context=None) <i>Récupère les enregistrements comme des objets, ce qui permet d'utiliser la notation à point pour parcourir les champs et les relations.</i> Renvoie : <i>objet ou une liste d'objets demandés</i>	<pre>idea = self.browse(cr, uid, 42) print 'Idea description:', idea.description print 'Inventor country code:', idea.inventor_id.address[0].country_id.code for vote in idea.vote_ids: print 'Vote %2.2f' % vote.vote</pre>
default_get (cr, uid, fields, context=None) <i>Renvoie : dictionnaire des valeurs par défaut pour les champs (définies sur la classe d'objets, par les préférences d'utilisateur, ou par l'intermédiaire du contexte)</i>	<ul style="list-style-type: none"> fields : liste de noms de champs <pre>defs = self.default_get(cr, uid, ['name', 'active']) # active devrait être True par défaut assert defs['active']</pre>
perm_read (cr, uid, ids, details=True) <i>Renvoie : une liste de dictionnaires de propriétés pour chaque enregistrement demandé</i>	<ul style="list-style-type: none"> details : si <i>True</i>, les valeurs des champs <i>*_uid</i> sont remplacées par des paires (id, name_of_user) Les dictionnaires retournés contiennent : id de l'objet (id), id de l'utilisateur créateur (create_uid), date de création (create_date), id de l'utilisateur qui a mis à jour (write_uid), date de mise à jour (write_date) <pre>perms = self.perm_read(cr, uid, [42, 43]) print 'creator:', perms[0].get('create_uid', 'n/a')</pre>
fields_get (cr, uid, fields=None, context=None) <i>Renvoie un dictionnaire de dictionnaires de champs, chacun décrivant un champ de l'objet métier</i>	<ul style="list-style-type: none"> fields : liste de noms de champs <pre>class idea(osv.osv): (...) _columns = { 'name' : fields.char('Name', size=64) (...) def test_fields_get(self, cr, uid): assert(self.fields_get('name') ['size'] == 64)</pre>
fields_view_get (cr, uid, view_id=None, view_type='form', context=None, toolbar=False) <i>Renvoie un dictionnaire décrivant la composition de la vue demandée (y compris les vues héritées)</i>	<ul style="list-style-type: none"> view_type : type de vue à renvoyer si view_id est <i>None</i> ('form', 'tree', ...) toolbar : <i>True</i> pour renvoyer également des actions de contexte <pre>def test_fields_view_get(self, cr, uid): idea_obj = self.pool.get('idea.idea') form_view = idea_obj.fields_view_get(cr, uid)</pre>
name_get (cr, uid, ids, context=None) <i>Renvoie des tuples avec la représentation textuelle des objets demandés pour les relations to-many</i>	<pre># Les idées doivent être présentées avec la date d'inventio def name_get(self, cr, uid, ids): res = [] for r in self.read(cr, uid, ids['name', 'create_date'])</pre>

	<pre>res.append((r['id'], '%s (%s)' (r['name'],year)) return res</pre>
<p>name_search (cr, uid, name="", domain=None, operator='ilike', context=None, limit=80) <i>Renvoie une liste de noms d'objets correspondant aux critères utilisés pour permettre l'achèvement des relations to-many. Équivalent de search() sur le name (nom) + name_get()</i></p>	<ul style="list-style-type: none"> name : nom de l'objet à rechercher operator : l'opérateur pour le critère de noms domain, limit : pareil que pour search() <pre># Les pays peuvent être recherchés par code ou par nom def name_search(self, cr, uid, name='', domain=[], operator='ilike', context=None, limit=80): ids = [] if name and len(name) == 2: ids = self.search(cr, user, [('code', '=', name)] + args, limit=limit, context=context) if not ids: ids = self.search(cr, user, [('name', operator, name)] + args, limit=limit, context=context) return self.name_get(cr, uid, ids)</pre>
<p>export_data(cr, uid, ids, fields, context=None) <i>Exportation des champs des objets sélectionnés, renvoyant un dictionnaire avec une matrice de données. Utilisé lors de l'exportation de données via le menu du client.</i></p>	<ul style="list-style-type: none"> fields : liste de noms de champs context : peut contenir import_comp (par défaut : False) pour rendre les données exportées compatibles avec import_data() (peut empêcher l'exportation de certains champs)
<p>import_data (cr, uid, fields, data, mode='init', current_module="", nouupdate=False, context=None, filename=None) <i>Importe les données spécifiées dans le module spécifié utilisé lors de l'exportation de données via le menu du client</i></p>	<ul style="list-style-type: none"> fields : liste de noms de champs data : données à importer (voir export_data()) mode : 'init' pour créer des enregistrements ou 'update' pour mettre à jour current_module : nom du module nouupdate : signal pour la création des enregistrements filename : fichier optionnel pour stocker l'état des importations partielles pour la récupération



Astuce :

Utilisez **read()** pour des appels via des Web services, mais préférez **browse()** en interne.

IV - Construire l'interface du module

Pour construire un module, le mécanisme principal est d'insérer des enregistrements de données pour déclarer les composants de l'interface du module. Chaque élément du module est un bloc de données standard : les menus, les vues, les actions, les rôles, les droits d'accès, etc.

IV-A - Structure XML commune

Les fichiers XML déclarés dans la section des données d'un module contiennent des déclarations d'enregistrement sous la forme suivante :

Structure XML commune

```

87. <?xml version="1.0" encoding="utf-8"?>
88. <openerp>
89.   <data>
90.     <record model="object_model_name" id="object_xml_id">
91.       <field name="field1">value1</field>
92.       <field name="field2">value2</field>
93.     </record>
94.
95.     <record model="object_model_name2" id="object_xml_id2">
96.       <field name="field1" ref="module.object_xml_id"/>
97.       <field name="field2" eval="ref('module.object_xml_id')"/>
98.     </record>
99.   </data>
100. </openerp>

```

Chaque type d'enregistrement (vue, menu, action) prend en charge un ensemble spécifique d'entités et d'attributs enfants, mais tous partagent les attributs spéciaux suivants :

id	l'identificateur externe unique (par module) de cet enregistrement (xml_id)
ref	peut être utilisé à la place du contenu normal de l'élément pour se référencer à un autre enregistrement (fonctionne entre les modules en faisant précéder par le nom du module parent)
eval	utilisé à la place du contenu d'un élément pour fournir de la valeur comme une expression Python, et qui peut utiliser la méthode ref() pour trouver l'identifiant dans base de données pour un xml_id spécifié

Astuce : Validation XML RelaxNG

OpenERP valide la syntaxe et la structure des fichiers XML, selon la grammaire  RelaxNG,

que l'on peut trouver à :

[server/bin/import_xml.rng](#)

Pour une vérification manuelle, utilisez :

xmllint: `xmllint -relaxng /path/to/import_xml.rng <file>`

IV-B - Syntaxe CSV commune

Les fichiers CSV peuvent également être ajoutés dans la section de données et les enregistrements seront insérés par la méthode `import_data()` de l'OSV, en utilisant le nom du fichier CSV afin de déterminer le modèle de l'objet cible. L'ORM reconnecte automatiquement les relations basées sur les noms des colonnes spéciales suivantes :

id (xml_id)	colonne contenant des identificateurs pour les relations
many2one_field	reconnecte many2one en utilisant name_search()
many2one_field:id	reconnecte many2one basé sur le xml_id de l'objet
many2one_field.id	reconnecte many2one basé sur l' id de l'objet de base de données
many2many_field	reconnecte via name_search() , multiples valeurs séparées par des virgules
many2many_field:id	reconnecte les xml_id des objets, multiples valeurs séparées par des virgules
many2many_field.id	reconnecte les id des objets de base de données, multiples valeurs séparées par des virgules
one2many_field/field	crée un enregistrement one2many de destination et assigne la valeur de champ

ir.model.access.csv

ir.model.access.csv

```

101. "id","name","model_id:id","group_id:id","perm_read","perm_write","perm_create","perm_unlink"
102. "access_idea_idea","idea.idea","model_idea_idea","base.group_user",1,0,0,0

```


ir.model.access.csv

103. "access_idea_vote","idea.vote","model_idea_vote","base.group_user",1,0,0,0

IV-C - Menus et actions

Les actions sont déclarées comme enregistrements réguliers et peuvent être déclenchées de trois façons :

- en cliquant sur les éléments de menu liés à une action spécifique ;
- en cliquant sur les boutons dans les vues, si ceux-ci sont connectés à des actions ;
- comme actions contextuelles sur un objet (visible dans la barre latérale).

IV-C-1 - Déclaration d'une action

Déclaration d'une action

```

104. <record model="ir.actions.act_window" id="action_id">
105.   <field name="name">action.name</field>
106.   <field name="view_id" ref="view_id"/>
107.   <field name="domain">[list of 3-tuples (max 250 characters)]</field>
108.   <field name="context">{context dictionary (max 250 characters)}</field>
109.   <field name="res_model">object.model.name</field>
110.   <field name="view_type">form|tree</field>
111.   <field name="view_mode">form,tree,calendar,graph</field>
112.   <field name="target">new</field>
113.   <field name="search_view_id" ref="search_view_id"/>
114. </record>
  
```

id	identificateur de l'action dans la table <code>ir.actions.act_window</code> doit être unique
name	nom de l'action (obligatoire)
view_id	vue spécifique pour ouvrir (si manquant, la vue avec la plus haute priorité du type spécifié est utilisée)
domain	tuple (voir les paramètres de <code>search()</code>) pour filtrer le contenu de la vue
context	dictionnaire de contexte à passer à la vue
res_model	modèle d'objet sur lequel la vue à ouvrir est définie
view_type	mettre à <i>form</i> pour ouvrir les enregistrements en mode édition, mettre à <i>tree</i> pour une vue hiérarchique uniquement
view_mode	si view_type est <i>form</i> , la liste des modes d'affichage autorisés pour voir les enregistrements (<i>form</i> , <i>tree</i> , ...)
target	mettre à <i>new</i> pour ouvrir la vue dans une nouvelle fenêtre/pop-up
search_view_id	identificateur de la vue de recherche pour remplacer le formulaire de recherche par défaut

IV-C-2 - Déclaration d'un menu

L'élément `menuitem` est un raccourci pour déclarer un enregistrement de `ir.ui.menu` et le connecte à une action correspondante à un enregistrement de `ir.model.data`.

Déclaration d'un menu

```

115. <menuitem id="menu_id" parent="parent_menu_id" name="label"
116.   action="action_id" groups="groupname1,groupname2" sequence="10"/>
  
```


id	identificateur du menuitem , doit être unique
parent	id externe (xml_id) du menu parent dans la hiérarchie
name	étiquette de menu optionnelle (par défaut : nom de l'action)
action	identificateur de l'action à exécuter, le cas échéant
group	liste des groupes qui peuvent voir ce menu (si manquant, tous les groupes peuvent le voir)
sequence	indice entier pour ordonner les menuitems du même parent (10,20,30..)

V - Vues et héritage

Les vues forment une hiérarchie. Plusieurs vues d'un même type peuvent être déclarées sur le même objet, et seront utilisées en fonction de leurs priorités. En déclarant une vue héritée, il est possible d'ajouter/supprimer des fonctions dans une vue.

Déclaration d'une vue générique

Déclaration d'une vue générique

```

117. <record model="ir.ui.view" id="view_id">
118.   <field name="name">view.name</field>
119.   <field name="model">object_name</field>
120.   <!-- types: tree,form,calendar,search,graph,gantt,kanban-->
121.   <field name="type">form</field>
122.   <field name="priority" eval="16"/>
123.   <field name="arch" type="xml">
124.     <!-- contenu de la vue: <form>, <tree>, <graph>, ... -->
125.   </field>
126. </record>

```

name	nom de la vue
model	modèle d'objet sur lequel la vue est définie (comme res_model dans les actions)
type	form , tree , graph , calendar , search , gantt , kanban
priority	priorité de la vue, la plus petite est la plus élevée (par défaut : 16)
arch	architecture de la vue, voir différents types de vue ci-dessous

V-A - Vues formulaires (pour voir/modifier les enregistrements)

Éléments autorisés	Tous (voir les éléments du formulaire ci-dessous)
--------------------	--

Vue formulaire (form view)

```

127. <form string="Idea form">
128.   <group col="6" colspan="4">
129.     <group colspan="5" col="6">
130.       <field name="name" colspan="6"/>
131.       <field name="inventor_id"/>
132.       <field name="inventor_country_id" />
133.       <field name="score"/>
134.     </group>
135.     <group colspan="1" col="2">
136.       <field name="active"/><field name="invent_date"/>
137.     </group>
138.   </group>
139.   <notebook colspan="4">
140.     <page string="General">
141.       <separator string="Description"/>
142.       <field colspan="4" name="description" nolabel="1"/>
143.     </page>
144.     <page string="Votes">
145.       <field colspan="4" name="vote_ids" nolabel="1">

```

Vue formulaire (form view)

```

146.         <tree>
147.             <field name="partner_id"/>
148.             <field name="vote"/>
149.         </tree>
150.     </field>
151. </page>
152. <page string="Sponsors">
153.     <field colspan="4" name="sponsor_ids" nolabel="1"/>
154. </page>
155. </notebook>
156. <field name="state"/>
157. <button name="do_confirm" string="Confirm" type="object"/>
158. </form>
  
```

Nouveau : l'API de formulaire v7.0

Une nouvelle API de vue formulaire a été introduite dans OpenERP 7.0. Elle peut être activée en ajoutant `version="7.0"` à l'élément `<form>`.

Cette nouvelle API de formulaire permet de mélanger le code XHTML arbitraire avec des éléments de formulaire réguliers d'OpenERP.



Il introduit également quelques éléments propres à produire des formulaires plus beaux, comme `<sheet>`, `<header>`, `<footer>`, et un ensemble de classes CSS génériques pour personnaliser l'apparence et le comportement des éléments de formulaire.

Les meilleures pratiques et des exemples pour la nouvelle API de formulaire sont disponibles dans la documentation technique :

 <http://doc.openerp.com/trunk/developers/server/form-view-guidelines>

V-A-1 - Les éléments de formulaire

Les attributs communs à tous les éléments :

- `string` : label de l'élément ;
- `nolabel` : mettre à `1` pour cacher l'étiquette du champ ;
- `colspan` : nombre de colonnes sur lesquelles le champ doit s'étendre ;
- `rowspan` : nombre de lignes sur lesquelles le champ doit s'étendre ;
- `col` : nombre de colonnes que cet élément doit allouer à ses éléments enfants ;
- `invisible` : mettre à `1` pour cacher cet élément complètement ;
- `eval` : évaluer ce code Python comme contenu d'un élément (le contenu est une chaîne par défaut) ;
- `attrs` : carte Python définissant les conditions dynamiques sur ces attributs : `readonly`, `invisible`, `required`, en fonction de tuples de recherche sur d'autres valeurs de champs.

field : widgets automatiques en fonction du type de champ correspondant.

Attributs :

- `string` : étiquette du champ pour cette vue particulière ;
- `nolabel` : mettre à `1` pour cacher l'étiquette du champ ;
- `required` : substitue l'attribut `required` du champ du modèle pour cette vue ;
- `readonly` : substitue l'attribut `readonly` du champ du modèle pour cette vue ;
- `password` : mettre à `True` pour masquer les caractères entrés dans ce champ ;
- `context` : code Python déclarant un dictionnaire contextuel ;
- `domain` : code Python déclarant une liste de tuples pour restreindre les valeurs ;
- `on_change` : méthode Python à appeler lorsque la valeur du champ est modifiée ;
- `groups` : liste de groupes (`id`) séparés par des virgules groupe (`id`) qui ont la permission de voir ce champ ;
- `widget` : possibilités du widget de sélection (`url`, `email`, `image`, `float_time`, `reference`, `html`, `progressbar`, `statusbar`, `handle`, etc.).

properties : widget dynamique montrant toutes les propriétés disponibles (pas d'attribut).

button : widget cliquable associé à des actions.

Attributs :

- **type** : type de bouton : *workflow* (par défaut), *object* ou *action* ;
- **name** : signal de *workflow*, nom de la fonction (sans les parenthèses) ou action à appeler (dépend de **type**) ;
- **confirm** : texte du message de confirmation lorsque vous cliquez dessus ;
- **states** : liste des états séparés par des virgules dans lesquels ce bouton s'affiche.

separator : ligne de séparation horizontale pour structurer les vues, avec étiquette facultative.

newline : espace réservé pour l'achèvement de la ligne actuelle de la vue.

label : légende en texte libre ou une légende dans le formulaire.

group : utilisé pour organiser les champs en groupes avec étiquette facultative (rajoute des cadres).

notebook : les éléments d'un notebook sont des onglets pour des éléments **page**.

Attributs :

- **page**
- **name** : étiquette de l'onglet/page ;
- **position** : position des onglets dans le notebook (*inside*, *top*, *bottom*, *left*, *right*).

V-B - Vues dynamiques

En plus de ce qui peut être fait avec les attributs **states** et **attrs**, des fonctions peuvent être appelées par des éléments de la vue (via les boutons de type **object**, ou par les déclencheurs **on_change** sur les champs) pour obtenir un comportement dynamique.

Ces fonctions peuvent modifier l'interface de la vue en renvoyant une carte Python avec les entrées suivantes :

value	un dictionnaire de noms de champs et leurs nouvelles valeurs
domain	un dictionnaire de noms de champs et les valeurs actualisées du domaine
warning	un dictionnaire avec un titre et un message à afficher dans une boîte de dialogue d'avertissement

V-C - Vues listes et listes d'arborescence hiérarchique

Les vues liste qui incluent les éléments **field**, sont créées avec le type **tree**, et ont un élément parent **<tree>**. Elles sont utilisées pour définir des listes plates (modifiables ou non) et les listes hiérarchiques.

Attributs	<ul style="list-style-type: none"> • colors : liste des couleurs ou des codes de couleur HTML mappés à des conditions Python • editable : <i>top</i> ou <i>bottom</i> pour permettre l'édition en place • toolbar : mettre à <i>True</i> pour afficher le plus haut niveau de la hiérarchie d'objets comme une barre d'outils latérale (uniquement pour les listes hiérarchiques, c'est-à-dire ouvertes avec des actions qui ont view_type à <i>"tree"</i> au lieu de <i>"mode"</i>)
Éléments autorisés	• field , group , separator , tree , button , filter , newline


Vue liste (tree view)

```
159. <tree string="Idea Categories" toolbar="1" colors="blue:state==draft">
160.     <field name="name"/>
161.     <field name="state"/>
162. </tree>
```

V-D - Vues fiches Kanban

Note du traducteur :

Définition Wikipédia de « Kanban » :

 Un kanban (#### ou ##, terme japonais signifiant « regarder le tableau »?) est une simple fiche cartonnée que l'on fixe sur les bacs ou les conteneurs de pièces dans une ligne d'assemblage ou une zone de stockage.

Voir Kanban sur Wikipédia : <http://fr.wikipedia.org/wiki/Kanban>

Depuis OpenERP 6.1, un nouveau type polyvalent de vue, dans laquelle chaque enregistrement est rendu comme un petit « **kanban** » (fiche) est apparu. Il supporte le glisser-déposer pour gérer le cycle de vie des fiches kanban basées sur des dimensions configurables. Les vues Kanban sont introduites dans les notes de version d'OpenERP 6.1 et définies en utilisant le langage de templates **QWeb**, documenté dans la documentation technique :

Voir : <http://bit.ly/18usDXt>

Et : <http://doc.openerp.com/trunk/developers/web/qweb>

V-E - Vues calendrier

Vues utilisées pour afficher les champs de date comme des événements de calendrier (élément parent : `<calendar>`)

Attributs	<ul style="list-style-type: none"> <code>color</code> : nom du champ pour la segmentation de la couleur <code>date_start</code> : nom du champ contenant le début de l'événement date/heure <code>day_length</code> : durée d'une journée [de travail] en heures (par défaut : 8) <code>date_stop</code> : nom du champ contenant l'arrêt de l'événement date/heure ou <code>date_delay</code> : nom du champ contenant la durée de l'événement
Éléments autorisés	<ul style="list-style-type: none"> <code>field</code> (pour définir l'étiquette de chaque événement du calendrier)

Vue calendrier (calendar view)

```
163. <calendar string="Ideas" date_start="invent_date" color="inventor_id">
164.     <field name="name"/>
165. </calendar>
```

V-F - Diagrammes de Gantt

Diagramme à barres généralement utilisé pour afficher le calendrier du projet (élément parent : `<gantt>`).

Attributs	<ul style="list-style-type: none"> les mêmes que <code><calendar></code>
Éléments autorisés	<ul style="list-style-type: none"> <code>field</code> : champ <code>level</code> : éléments qui sont utilisés pour définir les niveaux du diagramme de Gantt, avec le champ fermé utilisé comme étiquette pour le niveau de profondeur


Vue diagramme de Gantt (gant view)

```
163. <ganttt string="Ideas" date_start="invent_date" color="inventor_id">
164.     <level object="idea.idea" link="id" domain="[]">
165.         <field name="inventor_id"/>
166.     </level>
167. </ganttt>
```

V-G - Vues diagrammes (graphes)

Vues utilisées pour afficher les tableaux de statistiques (élément parent : `<graph>`).

Astuce :

 Les graphiques sont particulièrement utiles avec des vues personnalisées qui permettent d'extraire des statistiques prêtes à l'emploi.

Attributs	<ul style="list-style-type: none"> type : type de graphique : <i>bar</i>, <i>pie</i> (par défaut) orientation : <i>horizontal</i>, <i>vertical</i>
Éléments autorisés	<ul style="list-style-type: none"> field : avec un comportement spécifique : le premier champ dans la vue est l'axe X, le 2e est Y, le 3e est Z ; 2 champs sont requis, le 3e est en option ; group : attribut qui définit le champ GROUP BY (mis à 1) ; operator : attribut qui définit l'opérateur d'agrégation à utiliser pour d'autres domaines quand un champ est groupé (+, *, **, <i>min</i>, <i>max</i>)

Vue graphe (graph view)

```
171. <graph string="Total idea score by Inventor" type="bar">
172.     <field name="inventor_id" />
173.     <field name="score" operator="+"/>
174. </graph>
```

V-H - Vues de recherche

Les vues de recherche personnalisent le panneau de recherche en haut des autres vues.

Éléments autorisés	field , group , separator , label , search , filter , newline , propriétés : <ul style="list-style-type: none"> filter : éléments filtres permettant de définir le bouton des filtres de domaine ; l'ajout d'un attribut context aux champs permet aux widgets de modifier le contexte de recherche (utile pour les champs contextuels, par exemple, les prix des listes de prix dépendantes)
--------------------	--

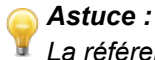
Vue recherche (search view)

```
175. <search string="Search Ideas">
176.     <group col="6" colspan="4">
177.         <filter string="My Ideas"
178.             domain="['inventor_id','=',uid]"
179.             help="My own ideas"/>
180.         <field name="name"/>
181.         <field name="description"/>
182.         <field name="inventor_id"/>
183.         <!-- le champ de contexte suivant est juste pour illustration -->
184.         <field name="inventor_country_id" widget="selection"
185.             context="{ 'inventor_country': self }"/>
186.     </group>
187. </search>
```

V-I - Héritage des vues

Les vues existantes devraient être modifiables à travers des vues héritées, jamais directement.

Une vue héritée se référence à sa vue parent en utilisant le champ `inherit_id`, et peut ajouter ou modifier des éléments existants dans la vue par leur référencement ou par des expressions XPath, et en spécifiant la position appropriée.



Astuce :

La référence à **XPath** peut être trouvée à l'adresse www.w3.org/TR/xpath

position	<ul style="list-style-type: none"> inside : mettre à l'intérieur de la correspondance (par défaut) replace : remplacer la correspondance before : mettre avant la correspondance after : mettre après la correspondance
-----------------	---

XPath

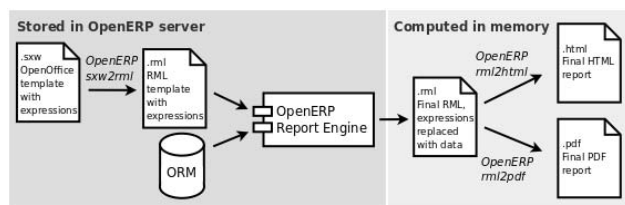
```

188. <!-- amélioration de la liste des catégories d'idée -->
189. <record id="idea_category_list2" model="ir.ui.view">
190.   <field name="name">id.category.list2</field>
191.   <field name="model">ir.ui.view</field>
192.   <field name="inherit_id" ref="id_category_list"/>
193.   <field name="arch" type="xml">
194.     <xpath expr="/tree/field[@name='description']" position="after">
195.       <field name="idea_ids" string="Number of ideas"/>
196.     </xpath>
197.   </field>
198. </record>

```

VI - Rapports

Il existe plusieurs moteurs de rapport dans OpenERP, pour produire des rapports à partir de différentes sources et dans de nombreux formats.



Processus de génération de rapport

Les expressions spéciales utilisées à l'intérieur des modèles de rapport produisent des données dynamiques et/ou modifient la structure du rapport au moment du rendu.

Des analyseurs de rapport personnalisés peuvent être écrits pour supporter d'autres expressions.

VI-A - Différents formats de rapports

sxw2rml	Modèles OpenOffice 1.0 (.sxw) convertis en RML avec l'outil sxw2rml , puis le RML rendu au format HTML ou PDF
rml	Modèles RML rendus directement au format HTML ou PDF
xml,xsl:rml	Données XML + feuilles de styles XSL:RML pour générer le RML
odt2odt	Modèles OpenOffice (.odt) utilisés pour produire directement des documents OpenOffice (.odt)

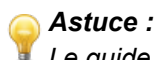
VI-B - Les expressions utilisées dans les modèles de rapport OpenERP

Expressions utilisées dans les modèles de rapport OpenERP	
<code>[[<contenu>]]</code>	Le contenu à l'intérieur des doubles crochets est évalué comme une expression Python sur la base des expressions suivantes
Expressions prédéfinies : <ul style="list-style-type: none"> objects : les objets contiennent la liste des documents à imprimer ; data : les données proviennent de l'assistant qui lance le rapport ; user : contient l'utilisateur courant (<code>browse_record</code>, comme renvoyé par <code>browse()</code>) ; time : le temps donne accès au module de temps Python ; repeatIn(list, 'var', 'tag') répète l'élément parent actuel nommé tag pour chaque objet dans la list, ce qui rend l'objet disponible comme var lors de chaque boucle ; setTag('tag1', 'tag2') remplace le parent RML tag1 avec tag2 ; removeParentNode('tag') supprime le parent RML de l'élément tag ; formatLang(value, digits=2, date=False, date_time=False, grouping=True, monetary=False) peut être utilisé pour formater une date, l'heure ou le montant selon la localisation ; setLang('lang_code') définit le langage courant et la localisation pour les traductions. 	

Déclaration d'un rapport

```
1. <!-- Ce qui suit crée un enregistrement dans le modèle ir.actions.report.xml -->
2. <report id="idea_report" string="Print Ideas" model="idea.idea"
3.     name="idea_report" rml="idea/report/idea.rml" >
4. <!-- Utilisez addons/base_report_designer/wizard/tiny_sxw2rml/
tiny_sxw2rml.py pour générer le modèle RML depuis un modèle .sxw -->
```

id	identifiant de rapport unique
name	nom du rapport (obligatoire)
model	modèle d'objet sur lequel le rapport est défini (obligatoire)
rml, sxw, xml, xsl	chemin vers les sources de modèles (à partir de addons), selon le rapport
auto	mettre à False pour utiliser un interpréteur personnalisé, en contournant <code>report_sxw.rml_parse</code> et en déclarant le rapport comme suit : <code>report_sxw.report_sxw(report_name, object_model, rml_path, parser=customClass)</code>
header	mettre à False pour supprimer l'en-tête du rapport (par défaut : True)
groups	liste de groupes, séparés par des virgules, autorisés à voir ce rapport
menu	mettre à True pour afficher le rapport dans le menu d'impression (par défaut : True)
keywords	précise le type de mot-clé du rapport (par défaut : <code>client_print_multi</code>)



Astuce :

Le guide de l'utilisateur RML: www.reportlab.com/docs/rml2pdf-userguide.pdf

Extrait d'un exemple de rapport

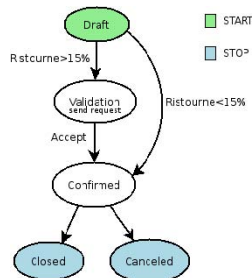
```
204. <story>
205.     <blockTable style="Table">
206.         <tr>
207.             <td><para style="Title">Idea name</para> </td>
208.             <td><para style="Title">Score</para> </td>
209.         </tr>
210.         <tr>
211.             <td><para>[[ repeatIn(objects,'o','tr') ]] [[ o.name ]]</para></td>
212.             <td><para>[[ o.score ]]</para></td>
213.         </tr>
214.     </blockTable>
215. </story>
```


VII - Les Flux de travail (Workflows)

Les flux de travail peuvent être associés à n'importe quel objet dans OpenERP, et sont entièrement personnalisables.

Les flux de travail sont utilisés pour structurer et gérer les cycles de vie des objets et documents commerciaux, et pour définir des transitions, des déclencheurs, etc. avec des outils graphiques.

Les flux de travail, les activités (nœuds ou les actions) et les transitions (conditions) sont déclarés comme des enregistrements XML, comme d'habitude. Les jetons qui naviguent dans les *workflows* sont appelés *workitems*.



Flux de travail - Workflows

VII-A - Déclaration d'un flux de travail

Les flux de travail sont déclarés sur des objets qui possèdent un champ d'état (voir l'exemple de la classe *idea* dans la section ORM).

Déclaration d'un flux de travail

```

216. <record id="wkf_idea" model="workflow">
217.   <field name="name">idea.basic</field>
218.   <field name="osv">idea.idea</field>
219.   <field name="on_create" eval="1"/>
220. </record>

```

id	identificateur unique d'enregistrement du flux de travail
name	nom du flux de travail (obligatoire)
osv	modèle d'objet sur lequel le flux de travail est défini (obligatoire)
on_create	si True , un élément de travail est instancié automatiquement pour chaque nouvel enregistrement osv

VII-B - Activités du flux de travail (nœuds)

Activité du flux de travail

```

221. <record id="act_confirmed" model="workflow.activity">
222.   <field name="name">confirmed</field>
223.   <field name="wkf_id" ref="wkf_idea"/>
224.   <field name="kind">function</field>
225.   <field name="action">action_confirmed()</field>
226. </record>

```

id	identificateur unique de l'activité
wkf_id	identificateur du flux de travail parent
name	étiquette du nœud de l'activité
flow_start	<i>True</i> pour faire un nœud <i>'begin'</i> , recevant un workitem (élément de travail pour chaque instance de workflow (flux d'activité))
flow_stop	<i>True</i> pour faire un nœud <i>'end'</i> , terminant le flux de travail lorsque tous les éléments l'ont atteint
join_mode	comportement logique de ce nœud en ce qui concerne les transitions entrantes : <ul style="list-style-type: none"> XOR : activer sur la première transition d'entrée (par défaut) AND : attend que toutes les transitions entrantes deviennent valides
split_mode	comportement logique de ce nœud en ce qui concerne les transitions sortantes : <ul style="list-style-type: none"> XOR : une transition valide est nécessaire, envoyant le workitem sur elle (par défaut) OR : envoyer les workitem sur toutes les transitions valides (0 ou plus), de manière séquentielle AND : envoyer un workitem sur toutes les transitions valides en une seule fois (fork)
kind	type de l'action à exécuter lorsque le nœud est activé par une transition : <ul style="list-style-type: none"> dummy : pour n'effectuer aucune opération lorsqu'il est activé (par défaut) function : pour invoquer une fonction déterminée par l'action subflow : sous-flux à exécuter avec subflow_id, en invoquant des mesures pour déterminer l'id d'enregistrement de l'enregistrement pour lequel le sous-flux doit être instancié. Si l'action ne renvoie aucun résultat, l'élément de travail est supprimé. stopall : mettre fin au flux de travail lors de l'activation
subflow_id	en cas de sous-flux, id du subflow à exécuter (utiliser l'attribut ref ou search avec un tuple)
action	appel à la méthode de l'objet, utilisé si kind est function ou subflow . Cette fonction doit également mettre à jour le champ d'état de l'objet, par exemple pour un type de fonction : <pre> 1. def action_confirmed(self, cr, uid, ids): 2. self.write(cr, uid, ids, { 'state' : 'confirmed' }) 3. # ... effectue d'autres tâches 4. return True </pre>

VII-C - Transitions du flux de travail (bords)

Les conditions sont évaluées dans cet ordre : **role_id**, **signal**, **condition**, **expression**

Transitions du flux de travail

```

227. <record id="trans_idea_draft_confirmed" model="workflow.transition">
228.     <field name="act_from" ref="act_draft"/>
229.     <field name="act_to" ref="act_confirmed"/>
230.     <field name="signal">button_confirm</field>
231.     <field name="role_id" ref="idea_manager"/>
232.     <field name="condition">1 == 1</field>
233. </record>

```

act_from, act_to	identifiants des activités de source et de destination
signal	nom d'un bouton de type workflow qui déclenche cette transition
role_id	référence au rôle que l'utilisateur doit avoir pour déclencher la transition (voir Rôles)
condition	expression Python qui doit évaluer la valeur True pour que la transition soit déclenchée

Astuce :



OpenERP dispose d'un éditeur de flux de travail graphique, disponible en passant à la vue diagramme tout en affichant un flux de travail dans : Paramètres > Technique > Workflows.

VIII - Sécurité

Des mécanismes de contrôle d'accès doivent être combinés pour aboutir à une politique de sécurité cohérente.

VIII-A - Mécanismes de contrôle d'accès basés sur le groupe

Les groupes sont créés comme des enregistrements normaux sur le modèle [res.groups](#), et bénéficient d'accès aux menus par des définitions de menu.

Cependant, même sans menu, les objets peuvent encore être accessibles indirectement, donc les autorisations actuelles au niveau de l'objet ([create](#), [read](#), [write](#), [unlink](#)) doivent être définies pour les groupes.

Elles sont généralement insérées via des fichiers CSV à l'intérieur des modules. Il est également possible de restreindre l'accès à des champs spécifiques sur une vue ou sur un objet en utilisant l'attribut [groups](#) du champ.

ir.model.access.csv

```
234. "id", "name", "model_id:id", "group_id:id", "perm_read", "perm_write", "perm_create", "perm_unlink"
235. "access_idea_idea", "idea.idea", "model_idea_idea", "base.group_user", 1, 1, 1, 0
236. "access_idea_vote", "idea.vote", "model_idea_vote", "base.group_user", 1, 1, 1, 0
```

VIII-B - Rôles

Les rôles sont créés comme des enregistrements normaux sur le modèle [res.roles](#) et sont utilisés uniquement pour conditionner les transitions de [workflow](#) à travers l'attribut [role_id](#) des transitions.

IX - Les assistants (Wizards)

Les assistants décrivent des sessions d'états interactifs avec l'utilisateur à travers des formulaires dynamiques. Ils sont construits sur la base de la classe [osv.TransientModel](#) et sont automatiquement détruits après usage. Ils sont définis en utilisant la même API et les vues sont des objets [osv.Model](#) réguliers.

IX-A - Les modèles d'assistant (TransientModel)

Modèle d'un assistant

```
237. from osv import fields, osv
238. import datetime
239. class cleanup_wizard(osv.TransientModel):
240.     _name = 'idea.cleanup.wizard'
241.     _columns = {
242.         'idea_age': fields.integer('Age (in days)'),
243.     }
244.     def cleanup(self, cr, uid, ids, context=None):
245.         idea_obj = self.pool.get('idea.idea')
```

Modèle d'un assistant

```

246.         for wiz in self.browse(cr,uid,ids):
247.             if wiz.idea_age <= 3:
248.                 raise osv.exception('UserError','Please select a larger age')
249.             limit = datetime.date.today()-datetime.timedelta(days=wiz.idea_age)
250.             ids_to_del = idea_obj.search(cr,uid, [('create_date', '<',
251.                 limit.strftime('%Y-%m-%d 00:00:00'))],context=context)
252.             idea_obj.unlink(cr,uid,ids_to_del)
253.         return {}

```

IX-B - Vues assistant

Les assistants utilisent des vues régulières et leurs boutons peuvent utiliser un attribut spécial **cancel** pour fermer la fenêtre de l'assistant lorsque vous cliquez dessus.

Vue de l'assistant

```

254. <record id="wizard_idea_cleanup" model="ir.ui.view">
255.     <field name="name">idea.cleanup.wizard.form</field>
256.     <field name="model">idea.cleanup.wizard</field>
257.     <field name="type">form</field>
258.     <field name="arch" type="xml">
259.         <form string="Idea Cleanup Wizard">
260.             <label colspan="4" string="Select the age of ideas to cleanup"/>
261.             <field name="idea_age" string="Age (days)"/>
262.             <group colspan="4">
263.                 <button string="Cancel" special="cancel"/>
264.                 <button string="Cleanup" name="cleanup" type="object"/>
265.             </group>
266.         </form>
267.     </field>
268. </record>

```

IX-C - Exécution de l'assistant

Ces assistants sont lancés via les enregistrements d'action réguliers, avec un champ cible spécial utilisé pour ouvrir la vue de l'assistant dans une nouvelle fenêtre.

Exécution de l'assistant

```

269. <record id="action_idea_cleanup_wizard" model="ir.actions.act_window">
270.     <field name="name">Cleanup</field>
271.     <field name="type">ir.actions.act_window</field>
272.     <field name="res_model">idea.cleanup.wizard</field>
273.     <field name="view_type">form</field>
274.     <field name="view_mode">form</field>
275.     <field name="target">new</field>
276. </record>

```

X - WebServices - XML-RPC

OpenERP est accessible à travers des interfaces XML-RPC, pour lesquels les bibliothèques existent dans de nombreux langages.

Exemple Python

```

277. import xmlrpclib
278. # ... définir HOST, PORT, DB, USER, PASS
279. url = 'http://%s:%d/xmlrpc/common' % (HOST,PORT)
280. sock = xmlrpclib.ServerProxy(url)
281. uid = sock.login(DB,USER,PASS)
282. print "Logged in as %s (uid:%d)" % (USER,uid)
283. # Crée une nouvelle idée
284. url = 'http://%s:%d/xmlrpc/object' % (HOST,PORT)
285. sock = xmlrpclib.ServerProxy(url)
286. args = {

```


Exemple Python

```
287.     'name' : 'Another idea',
288.     'description' : 'This is another idea of mine',
289.     'inventor_id': uid,
290. }
291. idea_id = sock.execute(DB,uid,PASS,'idea.idea','create',args)
```

Exemple PHP

```
293. <?
294. include('xmlrpc.inc'); // Use phpxmlrpc library, available on sourceforge
295. // ... définir $HOST, $PORT, $DB, $USER, $PASS
296. $client = new xmlrpc_client("http://$HOST:$PORT/xmlrpc/common");
297. $msg = new xmlrpcmsg("login");
298. $msg->addParam(new xmlrpcval($DB, "string"));
299. $msg->addParam(new xmlrpcval($USER, "string"));
300. $msg->addParam(new xmlrpcval($PASS, "string"));
301. resp = $client->send($msg);
302. uid = $resp->value()->scalarval();
303. echo "Logged in as $USER (uid:$uid)"
304. // Crée une nouvelle idée
305. $arrayVal = array(
306.     'name'=>new xmlrpcval("Another Idea", "string") ,
307.     'description'=>new xmlrpcval("This is another idea of mine" , "string"),
308.     'inventor_id'=>new xmlrpcval($uid, "int"),
309. );
```

Note du traducteur :

 Il semblerait que l'exemple PHP ne soit pas complet. Il n'est fait mention nulle part du modèle `idea.idea`, donc le code ci-dessus ne devrait pas pouvoir utiliser la table.

XI - Optimisation des performances

En tant que logiciel de gestion d'entreprise qui a généralement à faire face à de grandes quantités d'enregistrements, vous voudriez peut-être faire attention aux conseils suivants, pour obtenir des performances constantes :

- ne placez pas d'appels à `browse()` à l'intérieur des boucles, mettez-les avant et n'accédez qu'aux objets parcourus à l'intérieur de la boucle. L'ORM optimisera le nombre de requêtes de base de données basé sur les attributs parcourus ;
- évitiez la récursivité sur les hiérarchies des objets (objets avec une relation `parent_id`), en ajoutant des champs d'entiers `parent_left` et `parent_right` sur votre objet, et en mettant `_parent_store` sur `True` dans votre classe d'objets. L'ORM va utiliser une hiérarchie de précommande transversale modifiée pour être en mesure d'effectuer des opérations récursives (par exemple `child_of`) avec des requêtes de base de données en temps(1) au lieu de temps(n) ;
- ne pas utiliser les champs de fonction à la légère, surtout si vous les incluez dans la vue liste.
Pour optimiser les fonctions des champs, deux mécanismes sont disponibles :
multi : tous les champs partageant la même valeur d'attribut **multi** seront calculés avec un seul appel à la fonction, ce qui devrait alors retourner un dictionnaire des valeurs dans son plan de valeurs,
store : les champs de fonction avec un attribut **store** seront stockés dans la base de données et recalculés à la demande lorsque les objets de déclenchement applicables sont modifiés. Le format de la spécification de déclenchement est le suivant :
store = { 'model': (_ref_fnct, fields, priority) } (voir l'exemple ci-dessous)

Exemple de code

```
311. def _get_idea_from_vote(self, cr, uid, ids, context=None):
312.     res = {}
313.     vote_ids = self.pool.get('idea.vote').browse(cr, uid, ids, context=context)
314.     for v in vote_ids:
315.         res[v.idea_id.id] = True # Store the idea identifiers in a set
316.     return res.keys()
317. def _compute(self, cr, uid, ids, field_name, arg, context=None):
318.     res = {}
319.     for idea in self.browse(cr, uid, ids, context=context):
```

Exemple de code

```

320.         vote_num = len(idea.vote_ids)
321.         vote_sum = sum([v.vote for v in idea.vote_ids])
322.         res[idea.id] = {
323.             'vote_sum': vote_sum,
324.             'vote_avg': (vote_sum/vote_num) if vote_num else 0.0,
325.         }
326.         return res
327.     _columns = {
328.         # Ces champs sont recalculés à chaque fois que l'un des votes change
329.         'vote_avg': fields.function(_compute, string='Votes Average',
330.             store = {'idea.vote': (_get_idea_from_vote, ['vote'], 10)}, multi='votes'),
331.         'vote_sum': fields.function(_compute, string='Votes Sum',
332.             store = {'idea.vote': (_get_idea_from_vote, ['vote'], 10)}, multi='votes'),
333.     }

```

XII - Communauté/contribution

Les projets OpenERP sont hébergés sur Launchpad (LP), où toutes les ressources du projet peuvent être trouvées : les branches Bazaar, suivi des bogues, des plans, des FAQ, etc.

Créez un compte gratuit sur  launchpad.net pour être en mesure de contribuer.

Les groupes Launchpad		
Groupes *	Membres	Restrictions Bazaar/ Launchpad
OpenERP Quality Team (~openerp)	OpenERP Core Team	Peut fusionner et intégrer sur des branches officielles
OpenERP Drivers (~openerp-drivers)	Membres actifs de la communauté sélectionnés	Peut confirmer des bogues et poser des jalons sur les bogues
OpenERP Community (~openerp-community)	Groupe ouvert, n'importe qui peut se joindre	Possibilité de créer des branches communautaires où chacun peut contribuer

**Les membres des groupes supérieurs sont également membres des groupes inférieurs*

XIII - Licence

Copyright © 2010-2013 Open Object Press. Tous droits réservés.

Vous pouvez récupérer une copie électronique de ce travail et le distribuer si vous ne modifiez pas le contenu. Vous pouvez également imprimer une copie pour être lue par vous seul.

Nous avons des contrats avec différents éditeurs de différents pays pour vendre et distribuer des versions papier ou électroniques de ce travail (traduites ou non) dans les librairies. Cela permet de distribuer et de promouvoir le produit OpenERP. Il nous aide aussi à créer des incitations pour payer les contributeurs et auteurs avec les redevances.

Pour cette raison, l'accord de traduire, modifier ou vendre ce travail est strictement interdit, sauf si OpenERP S.A. (représentant Open Object Press) vous donne une autorisation écrite pour cela.

Bien que toutes les précautions aient été prises dans la préparation de cet ouvrage, l'éditeur et les auteurs n'assument aucune responsabilité pour les erreurs ou omissions, ou pour les dommages résultant de l'utilisation de l'information contenue dans ce document.

Publié par Open Object Press, Grand-Rosière, Belgique.

XIV - Remerciements

Remerciements spéciaux à l'équipe d'OpenERP (OpenERP SA) pour avoir permis la traduction et la publication de cet article.

Le site officiel OpenERP :  <http://www.openerp.com/>

Un grand Merci aux membres de l'équipe de la rédaction de Developpez.com pour leurs conseils et corrections

-  **Claude Leloup**