easyObject

modern web applications made easy

Manuel Utilisateur

---

version 1.0 - décembre 2012

par Cédric Françoys

http://www.cedricfrancoys.be/easyobject

Table des matières

1. Présentation du fonctionnement

BSUR: Browse, Search, Update, Remove

Il n'y a que quatre fonctions de bases permettant de manipuler les objets, et elles sont les mêmes pour toutes les classes d'objets :

- *browse* (pour obtenir la valeur d'un ou plusieurs champs d'un ou plusieurs objets) ;
- *search* (pour chercher les identifiants des éventuels objets répondant à un ou plusieurs critères) ;
- *update* (pour créer un objet ou modifier un ou plusieurs champs d'un ou plusieurs objets) ;
- *remove* (pour supprimer un ou plusieurs objets).

Exemples :

```
// create a new object
$ids = update('core\User', array(0));

// update an object
update('core\User', $ids, array(
                                    'firstname' => 'John',
                                    'lastname' => 'Doe',
                                    'password' => 'secret'
                                    ));

// create and update 3 objects at once
$ids = update('core\User', array(0, 0, 0), array(
                                    'firstname' => 'John',
                                    'lastname' => 'Doe',
                                    'password' => 'secret'
                                    ));

// update several objects
update('core\User', $ids, array('password'=>md5('test')));

// search for one or more objects
$ids = search('core\User', array(
                                array('lastname', 'like', '%oe%'),
                                array('id', 'in', range(1, 100)
                                ));

// browse one or more objects (obtain firstnames of users whose id is in the list)
$values = &browse('core\User', $ids, array('firstname'));
```

Pour la liste complète des méthodes et de leurs signatures, voir la description des méthodes de base.

Description de la structure arborescente

Emplacement	Description
Dossier racine	L'emplacement contenant l'application easyObject (ex. : *home/easyobject/www*)
fichiers	
.htaccess	fichier de configuration apache utilisé pour empêcher le listing des répertoires et pour traiter les erreurs 404
fc.lib.php	Librairie d'inclusion de classes et de fichiers
index.php	Ce script est également appelé *dispatcher* : il est en charge de l'inclusion des librairies nécessaires et de la définition du contexte. C'est l'unique point d'entrée client.
rpc_server.php	Script serveur en cas d'utilisation en *mode client-server*
url_resolver.php	Script invoqué en cas d'erreur 404 et en charge de l'url rewriting
dossiers	
actions	Dossier contenant les scripts réalisant des opérations sur les objets et retournant éventuellement des réponses au format JSON (requêtes ?do=…)
apps	Dossier contenant les scripts applicatifs générant des pages html (requêtes ?show=…)
data	Dossier contenant les scripts permettant d'obtenir des informations sur les objets en retournant des réponses au format JSON (requêtes ?get=…)
library	Dossier contenant tout ce qui peut être inclus par des scripts php (fichiers et classes)
classes	dossier contenant les classes php
objects	Dossier contenant les définitions d'objets (fichiers .class.php), ainsi que les fichiers de template d'édition (views) et de traduction (i18n)
orm	cœur applicatif : contient les classes object.class.php, objectManager.class.php, IdentificationManager.class.php, I18n.class.php et ErrorHandler.class.php
db	Dossier contenant les classes associées à l'interface avec la base de données. DBConnection.class.php : factory pattern DBManipulator.cass.php et classes héritées : adaptation des méthodes avec les commandes propres au DBMS utilisé
utils	Classes utilitaires utilisées par le cœur applicatif
files	contient *config.inc.php*
[Zend]	Emplacement du framework Zend dans le cas où il est requis Note : il n'est pas utilisé par le noyau de easyObject
html	
js	les librairies javascript utilisées par les scripts applicatifs (/apps)
css	les feuilles de style html utilisées par les scripts applicatifs (/apps)

Définition d'un objet

Pour chaque classe d'objet, la structure est définie dans un fichier *.class.php* à l'emplacement : *library/classes/objects/[package]*.
Toutes les classes d'objet héritent d'une classe *Object* commune déclarée dans le namespace *core* et définie dans : *library/classes/orm/Object.class.php*

Une classe d'objet est toujours renseignée avec le nom du package auquel elle appartient. La syntaxe est 'package_name\class_name' (ex. :'school\Teacher').

Une classe d'objet est composée d'un certain nombre de champs, caractérisés par un nom et un type, ainsi que d'une série de méthodes.
Certaines méthodes sont « système » (leur nom est standard et est utilisé par l'ORM), et d'autres sont propres à une classe d'objet et définies par l'utilisateur (voir plus bas).

Cohérence entre définition .class.php et structure en DB

En parallèle, une table doit être définie en base de données dont la structure correspond à la classe associée. La contrainte principale est que les types doivent être compatibles au sens large (ex. : une colonne *varchar(255)* en DB peut indifféremment représenter un champ *string*, *short_text* ou *text* au sein de la classe associée).

La cohérence entre la définition d'un type et la structure de la table associée en DB doit être maintenue en permanence. Ceci est laissé à la responsabilité du développeur : aucun processus logiciel ne met en place la cohérence ni ne corrige les éventuelles incohérences (voir scripts utilitaires stand-alone).

Types de champs

Les champs se répartissent en trois genres :

1) Types directs
   boolean, integer, string, short_text, text, date, time, datetime, timestamp, selection, binary
2) Types relationnels
   one2many, many2many, many2one, related
3) Types fonctionnels
   function

ou en deux catégories :

1) Les champs simples ou triviaux : la valeur de ces champs est stockée directement dans la table SQL associée à l'objet et leur obtention ne nécessite donc pas de traitement
   boolean, integer, string, short_text, text, date, time, datetime, timestamp, selection, binary, many2one
2) Les champs complexes : champs dont la valeur nécessite un traitement pour être retrouvée
   one2many, many2many, related, function

Champs système

Certains champs sont communs à tous les objets :

Champ	Description
id	identifiant unique de l'instance
created	date et heure de la création de l'instance
modified	date et heure de la dernière modification de l'instance
creator	identifiant de l'utilisateur à l'origine de l'instance
modifier	identifiant de l'utilisateur à l'origine de la dernière modification de l'instance
published	booléen indiquant si l'instance est publiée
deleted	booléen indiquant si l'instance à été supprimée (« soft delete » ou mise en corbeille)

Méthodes communes

Certaines méthodes, définies dans la classe *core\Object*, sont communes à tous les objets :

Méthode	Description
__construct	le constructeur, invoqué lors de la création d'une nouvelle instance
getSpecialFields	renvoie les noms et types des champs systèmes (décrits plus haut)
getColumns	renvoie la description des champs utilisateurs
getSchema	renvoie le schéma complet (champs systèmes + champs défini dans *getColumns*)
getTable	renvoie le nom de la table SQL associée à la classe d'objet en cours
getId	renvoie l'identifiant de l'objet en cours
getModifiedFields	renvoie la liste des champs ayant déjà été chargé
getModifiedFields	renvoie la liste des champs qui ont été modifiés depuis le chargement de l'objet
resetLoadedFields	remet à zéro les flags de chargement des champs
resetModifiedFields	remet à zéro les flags de modification des champs
getUsedLangs	renvoie la liste des langues pour lesquelles au moins un champ est défini
getFieldsNames	renvoie les noms des champs dont le type est repris dans la liste spécifiée
getValues	renvoie un tableau associatif contenant les noms et valeurs des champs spécifiés
setValues	assigne les valeurs spécifiées aux champs de l'objet

Méthode getColumns()

La surcharge de cette méthode est obligatoire pour tout objet et renvoie un tableau associatif définissant la structure de l'objet (les champs ou colonnes).

Exemple issu de la classe *school\Course*:

```
public static function getColumns() {
    return array(
       'label'             => array(
                                  'type' => 'string'),
       'school_id'         => array(
                                  'type'              => 'many2one',
                                  'foreign_object'    => 'school\School'),
       'teachers_ids'      => array(
                                  'type'              => 'many2many',
                                  'foreign_object'    => 'school\Teacher',
                                  'foreign_field'     => 'courses_ids',
                                  'rel_table'         => 'school_rel_course_teacher',
                                  'rel_foreign_key'   => 'teacher_id',
                                  'rel_local_key'     => 'course_id'),
       'classes_ids'       => array(
                                  'type'              => 'one2many',
                                  'foreign_object'    => 'school\_Class',
                                  'foreign_field'     => 'course_id'),
    );
}
```

Base commune

Les attributs communs à tous les champs sont :

Attribut	Description
type	le type du champ (un type *easyObject* valide)
[*label*]	l'intitulé du champ (qui sera mentionné dans les formulaires d'édition) valeur par défaut = nom du champ
[*help*]	une indication à fournir à l'utilisateur final (style tooltip) concernant le rôle du champ ou la manière de le remplir
[*multilang*]	booléen indiquant si ce champ peut être traduit valeur par défaut = false note : seuls les champs simples peuvent être traduits (cela exclut les champs relationnels et fonctionnels)
[*search*]	booléen indiquant si l'interface utilisateur doit proposer ce champ comme critère de recherche valeur par défaut = false notes : - une recherche est possible sur les champs simples ou sur les champs complexes stockés en DB (voir plus bas) - les recherches présenteront une meilleure performance si les champs correspondants en DB sont indexés
[domain]	attribut pour les champs many2many et one2many permet de limiter les objets associés par la relation sur base d'un critère particulier (format identique de celui utilisé dans la fonction *search*)

Types disponibles

boolean
Utilisé pour un champ destiné à contenir une valeur numérique booléenne (vrai ou faux).
notes : utilisation des constantes PHP : ***true*** et ***false***

integer
Utilisé pour un champ destiné à contenir une valeur numérique signée (négative ou positive).
notes: sous PHP dépend de la plateforme (généralement 32 bits signé), sous SQL dépend du type et de la taille choisis

string
Utilisé pour un champ destiné à stocker une chaine de caractères courte ne contenant aucun formatage ni retour à la ligne (ex. : nom de famille, nom d'un lieu, …)

short_text
Utilisé pour un champ destiné à stocker une chaine de caractères contenant éventuellement des retours à la ligne mais pas de formatage (ex. : une adresse).

text
Utilisé pour un champ destiné à stocker un texte pouvant être assez long et contenir un formatage.
notes : sous SQL, les types TEXT, BLOB ou MEDIUMTEXT, MEDIUMBLOB sont recommandés

binary
Utilisé pour un champ destiné à stocker n'importe quelle valeur binaire (ex. : une image, un document).
notes : sous SQL, le type MEDIUMBLOB est recommandé pour ce type de champ

selection
Utilisé pour un champ destiné à stocker une valeur issue d'une liste prédéfinie.

Exemple issu de la classe *core**Log*:

```
'action'     => array(
                    'type'       => 'selection',
                    'selection'  => array(
                                          'R_CREATE'    => R_CREATE,
                                          'R_READ'      => R_READ,
                                          'R_WRITE'     => R_WRITE,
                                          'R_DELETE'    => R_DELETE,
                                          'R_MANAGE'    => R_MANAGE)),
```

Note : La partie à gauche de la flèche est celle qui sera affichée dans la boite de sélection, la partie à droite est ce qui sera stocké en base de données.

date
Utilisé pour un champ destiné à stocker une date au format : YYYY-mm-dd

time
Utilisé pour un champ destiné à stocker une heure au format : HH:mm:ss

datetime
Utilisé pour un champ destiné à stocker une date au format : YYYY-mm-dd HH:mm:ss

timestamp
Équivalent à datetime, mais mesuré en nombre de secondes depuis l'Epoch Unix (1er janvier 1970 00:00:00 GMT).

many2one
Utilisé pour un champ destiné à stocker une relation plusieurs-à-un (N-1), c'est-à-dire une valeur numérique représentant l'identifiant de l'objet pointé.

Attribut	Description
foreign_object	la classe d'objet vers laquelle pointe le champ courant

Exemple issu de la classe *school**Course*:

```
'school_id'   => array(
                    'type'              => 'many2one',
                    'foreign_object'    => 'school\School'),
```

one2many
Utilisé pour un champ destiné à stocker une relation un-à-plusieurs (1-N).

Attribut	Description
foreign_object	la classe d'objet vers laquelle pointe le champ courant
foreign_field	le nom du champ de la classe pointée qui pointe en retour vers la classe courante
[*foreign_key*]	le champ servant d'identifiant pour les objets pointés par la relation (par défaut, champ *id*)

Exemple issu de la classe *school**Teacher*:

```
'classes_ids'=> array(
                    'type'           => 'one2many',
                    'foreign_object' => 'school\_Class',
                    'foreign_field'  => 'teacher_id'),
```

many2many
Utilisé pour un champ destiné à stocker une relation plusieurs-à-plusieurs (M-N).

Attribut	Description
foreign_object	la classe d'objet vers laquelle pointe le champ courant
foreign_field	le nom du champ de la classe pointée qui pointe en retour vers la classe courante
rel_table	le nom de la table SQL dédiée à la relation m2m (syntaxe recommandée : *package*_rel_*class1*_*class2*)
rel_local_key	le nom de la colonne de *rel_table* contenant l'identifiant d'un objet de la classe courante
rel_foreign_key	le nom de la colonne de *rel_table* contenant l'identifiant d'un objet de la classe pointée

Exemple issu de la classe *school**Teacher*:

```
'courses_ids'=> array(
                    'type'            => 'many2many',
                    'foreign_object'  => 'school\Course',
                    'foreign_field'   => 'teachers_ids',
                    'rel_table'       => 'school_rel_course_teacher',
                    'rel_foreign_key' => 'course_id',
                    'rel_local_key'   => 'teacher_id'
                    ),
```

related

Ce type de champ permet de spécifier une relation indirecte de type *many2one* ou *one2many*.
Le principe est de renseigner le champ d'un autre objet, accessible par relations successives.

Attribut	Description
result_type	le type résultant de la relation indirecte (i.e. le type du champ pointé final)
foreign_object	le nom du champ pointé final
path	tableau associatif reprenant les noms des champs successifs permettant d'atteindre le champ final (note : le premier champ doit appartenir à la classe courante)

Exemple du champ *school_id* de la classe *school**Lesson* :

```
'school_id'   => array(
                     'type'           => 'related',
                     'result_type'    => 'many2one',
                     'foreign_object' => 'school\School',
                     'path'           => array('class_id','course_id','school_id')),
```

function

Les champs calculés (ou « fonctionnels ») permettent de doter une classe de champs dont la valeur dépend d'un ou plusieurs autres champs de l'objet en cours ou d'un autre objet.
Notes : l'utilisation de l'attribut *store* requiert, la plupart du temps, que les champs dont dépend la valeur du champ calculé aient un événement *onchange* déclenchant la mise à jour du champ calculé (voir exemple).

Attribut	Description
result_type	le type de la valeur résultant de la fonction invoquée
store	un booléen spécifiant si le résultat de la fonction doit être stocké en base de données (dans ce cas, la table associée doit contenir le champ correspondant)
function	une chaîne représentant le nom de la méthode devant être invoquée, au format : *package\Class::method* (note : cette méthode sera appelée à l'aide de la fonction PHP *call_user_func*)

Exemple du champ *rights_txt* de la classe *core\Permission* :

Extrait de la méthode *getColumns()* :

```
'rights'     => array(
                    'type' => 'integer',
                    'onchange' => 'core\Permission::onchange_Rights'),
'rights_txt' => array(
                    'type' => 'function',
                    'store' => true,
                    'result_type' => 'string',
                    'function' => 'core\Permission::callable_getRightsTxt'),
```

Méthodes additionnelles :

```
public static function callable_getRightsTxt($om, $uid, $oid, $lang) {
    $rights_txt = array();
    $res = $om->browse($uid, 'core\Permission', array($oid), array('rights'), $lang);
    $rights = $res[$oid]['rights'];
    if($rights & R_CREATE)    $rights_txt[] = 'create';
    if($rights & R_READ)      $rights_txt[] = 'read';
    if($rights & R_WRITE)     $rights_txt[] = 'write';
    if($rights & R_DELETE)    $rights_txt[] = 'delete';
    if($rights & R_MANAGE)    $rights_txt[] = 'manage';
    return implode(',', $rights_txt);
}

public static function onchange_Rights($om, $uid, $oid, $lang) {
    $om->update($uid,
                'core\Permission',
                array($oid),
                array('rights_txt' =>
                      Permission::callable_getRightsTxt($om, $uid,$oid, $lang)),
                $lang);
}
```

Méthode getDefaults()

Les valeurs de retour sont le résultat soit d'une fonction lamda (« closure ») soit d'une fonction statique (de la classe courante ou d'une autre classe).

Exemple classe *school\Student* :

Extrait de la méthode *getColumns()* :

```
       'birthdate'         => array('type' => 'date'),
       'subscription'      => array('type' => 'date'),
```

Méthodes additionnelles :

```
public static function getDefaults() {
       return array(
              'subscription'      => 'school\Student::default_subscription',
              'birthdate'         => function() { return '2000-01-01'; }
       );
}

public static function default_subscription() {
       return date("Y-m-d");
}
```

Méthode getTable()

Il est également possible de surcharger la méthode *getTable()*. Ceci peut être utile dans le cas où le nom par défaut n'est pas judicieux ou, pour une raison ou pour une autre, ne convient pas au développeur.

Exemple : classe *school_Class*

```
public function getTable() { return 'school_class'; }
```

Mécanisme d'héritage

Il est possible de surcharger une classe (dans certaines situations, la surcharge est même obligatoire).
Pour mettre en place un héritage :

- la classe doit être définie non pas comme dérivant de la classe *core**Object*, mais bien de celle dont elle hérite (en utilisant le mot-clé *extends*);
- la méthode *getTable()* doit être utilisée pour redéfinir le nom de la table associée en base de données (par défaut le nom de la table associée sera celle de la classe parente) ;
- la méthode *getColumns()* doit retourner un tableau associatif contenant tous les champs de la nouvelle classe (pour ne pas réécrire les champs communs avec la classe parente, on utilise la fonction PHP *array_merge* avec *parent::getColumns()* comme premier paramètre).

Exemple :
Classe **knine\User**

```
namespace knine {
	class User extends \core\User {

		public function getTable() { return 'core_user'; }

		public static function getColumns() {
			return array_merge(parent::getColumns(),
				array(
				'articles_ids' => array(
					'type'			=> 'many2many',
					'foreign_object'	=> 'knine\Article',
					'foreign_field'		=> 'authors_ids',
					'rel_table'		=> 'knine_rel_article_user',
					'rel_foreign_key'	=> 'article_id',
					'rel_local_key'		=> 'user_id')
				));
		}
	}
}
```

Système de gestion des droits

La classe 'Permission' est dédiée à la gestion des droits: pour chaque classe d'objet (y compris pour la classe 'Permission' elle-même), des permissions peuvent être définies pour un ou plusieurs groupes d'utilisateurs.

Les classes impliquées sont :

Classe **core\User** (library/classes/objects/core/User.class.php)

```
public static function getColumns() {
	return array(
		'firstname'		=> array('type' => 'string'),
		'lastname'		=> array('type' => 'string'),
		'login'			=> array('type' => 'string', 'label' => 'Username'),
		'password'		=> array('type' => 'string', 'label' => 'Password'),
		'language'		=> array('type' => 'string'),
		'groups_ids'		=> array('type' => 'many2many',
						'foreign_object'  => 'core\Group',
						'foreign_field'   => 'users_ids',
						'rel_table'       => 'core_rel_group_user',
						'rel_foreign_key' => 'group_id',
						'rel_local_key'   => 'user_id')
	);
}
```

Classe **core\Group** (library/classes/objects/core/Group.class.php)

```
public static function getColumns() {
	return array(
		'name'			=> array('type' => 'string'),
		'users_ids'		=> array('type' => 'many2many',
						'foreign_object'  => 'core\User',
						'foreign_field'   => 'groups_ids',
						'rel_table'       => 'core_rel_group_user',
						'rel_foreign_key' => 'user_id',
						'rel_local_key'   => 'group_id'),
		'permissions_ids'	=> array('type' => 'one2many',
					       'foreign_object'    => 'core\Permission',
					       'foreign_field'     => 'group_id')
	);
}
```

Classe **core\Permission** (library/classes/objects/core/Permission.class.php)

```
public static function getColumns() {
	return array(
		'class_name'		=> array('type' => 'string'),
		'group_id'		=> array(
					       'type'              => 'many2one',
					       'foreign_object'    => 'core\Group',
					       'foreign_field'     => 'permissions_ids'
					   ),
		'rights'		=> array('type' => 'integer')
	);
}
```

Les droits qu'il est possible d'attribuer sont définis dans le fichier config.inc.php :

```
define('R_CREATE',	1);	// autorisation de création
define('R_READ',	2);	// autorisation de consultation
define('R_WRITE',	4);	// autorisation de modification
define('R_DELETE',	8);	// autorisation de suppression
define('R_MANAGE',	16);	// autorisation de gestion des permissions
```

Le champ 'rights' des objets Permission, est une combinaison binaire (OU logiques) des différents droits concédés au groupe concerné (qu'il est possible d'isoler avec un masque binaire).

Si aucune permission n'est définie pour une classe donnée pour aucun des groupes auxquels appartient un utilisateur, celui-ci se voit attribuer les permissions par défaut, définies dans le fichier de configuration (constante *DEFAULT_RIGHTS*).

Systèmes de traduction

Deux mécanismes distincts sont à l'œuvre selon qu'il s'agisse :

1) de la traduction de termes liés à une classe d'objets ;
2) de la traduction de la valeur des champs au sein d'un objet (instance).

Traduction de termes liés à une classe d'objet

Chaque package possède un dossier intitulé 'i18n' lui-même contenant, pour chaque langue de traduction, un dossier dont le nom correspond au code IS0 639 de la langue (ex. : fr_BE ou zh_CN). Au sein de ces dossiers, pour chaque classe, est défini un fichier *.json* dont le préfixe est le même que le nom de la classe à laquelle il se rapporte (ex. : Student.json)
Ces fichiers de traduction sont au format UTF-8 et en notation JSON, et contiennent les traductions des termes de tous les éléments qui peuvent être traduits (attributs 'label' et 'help').
Ce système n'est pas le plus léger en terme d'overhead mais offre l'avantage d'être utilisable tel quel par l'interface graphique (pas de traitement côté serveur).

En effet, le fichier de traduction est accessible directement via requête HTTP :
'library/classes/objects/'+package_name+'/i18n/'+lang+'/'+object_name+'.json',

Par ailleurs, pour obtenir le fichier de traduction via script, il faut invoquer *?get=core_i18n_lang*.
Emplacement : `data/core/i18n/lang.php`
Exemple d'URL : http://localhost/easyobject/?get=core_i18n_lang&class=school\Student&lang=fr

Traduction de la valeur des champs d'un objet

Tous les champs de type simple peuvent être traduits (voir types de champs).
Pour permettre cela, une table dédiée aux traductions est définie en DB.
Lorsqu'un champ est marqué comme 'multilang', les valeurs de ses différentes traductions s'obtiennent à l'aide d'une requête SQL.

Objet 'Translation'
champs:
string lang (code ISO 639-1)
string object_class
string object_field
integer object_id
mediumblob value

Pour le champ value, le type SQL MEDIUMBLOB est utilisé (overhead de 3 bytes, taille max de 16,7 Mo).
En effet, la taille de la colonne 'value' peut très fort varier d'un champ à un autre (le type binary peut représenter un document, une image, une vidéo, …). Et, la plupart du temps, ce sont des textes qui nécessitent une traduction (string, short_text ou text). Dans tous les cas, un overhead de 3 caractères est acceptable (et imposer une limite à 16 mégaoctets ne semble pas préjudiciable).
Note : puisque le type SQL est libre du moment qu'il permette la compatibilité des types easyObject, il faut veiller, si l'on souhaite qu'un champ binaire puisse être traduit (par exemple un document pdf disponible en plusieurs langues), à ce que la valeur de ce champ n'ait jamais une taille supérieur à celle du type SQL MEDIUMBLOB (par exemple : BLOB, LONGBLOB, …).

2. Utilisation

Méthodes de base

Les méthodes suivantes peuvent être invoquées telles quelles à tout moment.
Note : ces méthodes sont définies aussi bien en PHP (librairie easyobject.api.php) qu'en Javascript (easyobject.api.js), et leurs signatures sont rigoureusement identiques dans les deux langages.

En cas d'erreur, la plupart de ces méthodes retournent un entier contenant un ou plusieurs codes d'erreur (qu'il est possible d'isoler grâce à un masque binaire), qui sont définis dans le fichier config.inc.php :

Constante	Valeur	Description
UNKNOWN_ERROR	0	something went wrong (that requires to check the logs)
INVALID_PARAM	1	one or more parameters have invalid or incompatible value
SQL_ERROR	2	error while building SQL query or processing it
UNKNOWN_OBJECT	3	unknown class or object
NOT_ALLOWED	4	action violates some rule or user don't have required permissions

Méthode user_id()

integer **user_id**()

Renvoie l'identifiant de l'utilisateur courant (sur base de l'identifiant de la session PHP en cours).

Méthode user_lang()

string **user_lang**()

Renvoie la langue (au format ISO 639) de l'utilisateur courant (sur base de l'identifiant de la session PHP en cours).

Méthode login()

boolean **login**(string *$login*, string *$password*)

La méthode login tente de valider l'identification d'un utilisateur. Si l'identification est réussie, la méthode renvoie TRUE et la session courante est alors liée à l'identifiant de l'objet utilisateur associé.

Méthode validate()

mixed **validate**(string *$object_class* , array *$values*)

Vérifie que les valeurs données pour le champ spécifié sont valides.

La valeur renvoyée est soir FALSE soit un tableau associant, pour chaque champ invlaide, le nom du champ associé à l'identifiant de l'erreur correspondante.

Méthode browse()

mixed &**browse**(string *$object_class* [, *$ids*=null [, *$fields*=null [, *$lang*=DEFAULT_LANG]]])

En cas de succès, cette méthode retourne, pour la classe d'objet donnée, un tableau associant la liste des valeurs de chaque champ spécifié pour chaque objet dont l'identifiant est renseigné.

En cas d'erreur, un entier contenant un ou plusieurs codes d'erreur est retourné.

Note : Cette fonction est susceptible de générer une requête SQL pour chacun des identifiants indiqués (pour les objets non encore chargés dans le manager). Une bonne pratique est donc de veiller à ne pas l'appeler consécutivement pour chacun des champs d'un objet, mais bien en regroupant tous les champs nécessaires dans le tableau *$fields*.
Exemple :

```
$values = &browse('core\User', array(3), array('login', 'firstname', 'lastname'));
echo "{$values[3]['login']}, {$values[3]['firstname']}, {$values[3]['lastname']}" ;
```

Méthode search()

mixed **search**(string *$object_class*, array *$domain*=null, string *$order*='', string *$sort*='asc', integer *$start*=0, *$limit*='', *$lang*=DEFAULT_LANG)

Retourne la liste des identifiants des objets correspondant à un domaine donné, trié sur un champ donné, éventuellement limitée à une certaine section.

Méthode update()

mixed **update**(string *$object_class*, array *$ids*, array *$values* [, *$lang*=DEFAULT_LANG])

Cette méthode permet à la fois de créer et de modifier des objets.
Lorsqu'un élément de la liste *ids* est mis à zéro, un nouvel objet est crée et son identifiant est placé dans le tableau retourné par la méthode. Pour des exemples, voir BSUR: Browse, Search, Update, Remove.

Méthode remove()

mixed **remove**(string *$object_class*, array *$ids*, boolean *$permanent*=false)

Cette méthode permet de supprimer un ou plusieurs objets. Si le paramètre permanent est laissé à FALSE, le champ *delete* de l'objet est mis à TRUE et l'objet n'apparaît plus dans les listes mais la suppression peut être annulée. Si le paramètre permanent est mis à TRUE, l'objet est supprimé de la base de données (et ne peut plus être récupéré à moins de la base de données n'ait été sauvegardée).

Méthode get()

mixed &**get**(string *$object_class* , integer *$object_id*)

Il est également possible d'obtenir une instance d'objet.
Pour obtenir ou modifier la valeur des champs déclarés dans la méthode *getColumns*, il est alors possible d'utiliser les méthodes get et set au format OO standard (implémentés avec la magic method PHP *__call*)
Exemple :

```
$User = &get('core\User', 3);
$first_name = $User->getFirstname();
$User->setFirstname('Lulu');
```

Notes : Cette méthode nécessite le chargement de tous les champs de l'objet (qui peuvent être nombreux et dont la valeur peut être complexe à obtenir). Il est donc recommandé de ne l'utiliser que lorsqu'on a effectivement besoin de l'instance de l'objet, et non pour obtenir un champ particulier (auquel cas il est préférable d'utiliser la méthode *browse*).

Méthode getStatic()

mixed &**getStatic**(string *$object_class*)

Dans certains cas, il est nécessaire d'obtenir un objet 'vide' pour une classe donnée (par exemple, dans le cadre de la validation du schéma d'une classe).

Édition des objets

Pour définir la mise en page, la liste des champs et les éventuelles interactions entre champs d'un formulaire d'édition, nous utilisons un système de vues comparable à des templates.

Chaque package possède un dossier intitulé 'views' contenant, pour chaque classe d'objet, un ou plusieurs fichiers HTML.
Ces fichiers sont au format HTML5, et contiennent des indications sur les champs et labels à afficher ainsi que sur leur disposition pour l'édition de la classe d'objets à laquelle ils se rapportent.
A nouveau, ce système n'est pas le plus léger en terme d'overhead mais offre l'avantage d'être utilisable tel quel par le client Javascript (pas de traitement côté serveur).

Le format de leur nom est : object_name.(list | form).view_name.html

Une série de tags HTML5 ont été choisis pour la rédaction des vues. Les tags utilisés ont été retenus sur base de l'usage qui en est habituellement fait :

- réservés aux inputs (puisque cette partie est générée automatiquement)
- qui n'ont, par défaut, aucun impact visuel

Tags et attributs pour les vues :

Tag	Attributs	Description
form	*action*	l'action à exécuter lors de la soumission du formulaire (bouton 'save')
section	*name*	permet de regrouper différents éléments au sein de sections accessibles par sélection d'onglets
fieldset	*title*	génère l'affichage d'un cadre permettant de regrouper plusieurs éléments
span	*width*	largeur du *span*, en % de la largeur du parent
div	*width*	largeur du *div*, en % de la largeur du parent
label	*[for]*	le cas échéant, le nom du champ auquel le libellé est associé
	[name]	le cas échéant, l'identifiant du libellé indépendant
var	*id*	le nom du champ
	[*onchange*]	action à exécuter en cas de modification de la valeur du champ par l'utilisateur (écrit en javascript, utilisant la syntaxe jQuery)
	[*readonly*]	booléen indiquant si l'interface utilisateur doit permettre la modification de ce champ (valeur par défaut = false) note : ce mécanisme est limité à la couche de présentation et ne doit pas être utilisé pour une gestion des permissions
	[*required*]	booléen indiquant si la saisie d'une valeur est indispensable pour ce champ lors de l'édition d'un objet (valeur par défaut = false)
	[*domain*]	chaîne de caractères représentant le domaine auquel on souhaite limiter les éléments à afficher (applicable dans le cas de listes m2m ou o2m)
	[*view*]	le nom de la vue à utiliser (si cet attribut n'est pas spécifié, c'est le fichier **.list.default.html* associé qui est utilisé)

Exemple d'un formulaire de la classe *school**Student* (Student.form.default.html) :

```
<form action="core_objects_update">
    <section name="identification">
        <fieldset title="identification">
            <span width="50%">
                <label for="firstname"></label>
                <var id="firstname" required="true"></var>
                <br />
                <label for="lastname"></label>
                <var id="lastname" required="true"></var>
            </span>
            <span width="50%">
                <label for="birthdate"></label>
                <var id="birthdate" required="true"></var>
                <br />
                <label for="subscription"></label>
                <var id="subscription" required="false"></var>
            </span>
        </fieldset>
    </section>
    <section name="data">
        <fieldset title="data">
            <span width="100%">
                <label name="classes_ids"></label>
                <br />
                <var id="classes_ids" view="list.default"></var>
            </span>
        </fieldset>
    </section>
</form>
```

Exemple d'une liste de la classe *school**Student* (Student.list.default.html) :

```
<ul>
	<li id="id" width="10%"></li>
	<li id="firstname" width="23%"></li>
	<li id="lastname" width="23%"></li>
	<li id="birthdate" width="22%"></li>
	<li id="subscription" width="22%"></li>
</ul>
```

Applications de base

Certaines applications (requêtes ?show=) sont incluses dans le noyau applicatif afin de faciliter la gestion et le développement des packages :

- *core_manage* : gestion des objets (liste, création, édition, suppression) par package
  (Exemple d'URL : http://localhost/easyobject/?show=core_manage)
- *core_utils* : (voir le point Scripts utilitaires stand-alone)
  (Exemple d'URL : http://localhost/easyobject/?show=core_utils)
- *core_setup* : permet de valider une nouvelle installation easyObject.

Scripts utilitaires stand-alone

Des scripts utilitaires permettent de réaliser certaines opérations non prises en charge par le noyau applicatif, comme par exemple valider la cohérence d'un nouveau package.
Il s'agit en quelque sorte de plugins, écrits en PHP et regroupés dans le dossier : *data/utils/*

Les scripts prévus, en cours de développement ou déjà disponibles sont :
Disponible (beta):

- Validation du noyau
- Validation d'un package : vérification de la cohérence entre DB et classes & de la syntaxe des classes (PHP), des vues (HTML) et des fichiers de traduction (json)

Prévus :

- Créer une base de données compatible sur base d'un schéma SQL
- Générer une classe PHP à partir d'une table existante
- Générer des fichiers vues par défaut (list.default.html et form.default.html)
- Import / Export de données

Pour consulter la liste des plugins et les appliquer à un ou plusieurs package, il est possible d'utiliser l'application *core_utils* (exemple d'URL : http://localhost/easyobject/index.php?show=core_utils).

Note : Bien entendu, ces scripts peuvent être écrits par le développeur-utilisateur lui-même ou adaptés en fonction de ses besoins particuliers.