KWO^4

Version 4.0.3

Date 21 octobre 2013

KWO[4] 1

1. PREFACE 8

1.1. CHEMINS 8
1.2. TYPES DE DONNEES 8
1.3. CONVENTIONS TYPOGRAPHIQUES 8

2. INTRODUCTION 9

2.1. NOTION DE FRAMEWORK 9
2.2. ORGANISATION D'UN PROJET 9

3. KWO EN UNE PAGE 10

4. FONCTIONNEMENT 11

4.1. MVC 11
4.2. CONVENTION OVER CONFIGURATION 11
4.3. SCOPES 11
4.4. APPLICATION / APP 12
4.4.1. Fichier de configuration *13*
4.4.2. Paramètres *13*
4.5. EXTENSION 15
4.5.1. Paramétrage *15*
4.6. ACTIONS 15
4.6.1. Cas d'utilisation *16*
4.7. TEMPLATES 17
4.7.1. Actions et templates *17*
4.7.2. Appel d'un template au sein d'un autre template *18*
4.7.3. Class Template *18*
4.7.4. Conventions *18*
4.7.5. Redéfinition d'un template *18*
4.8. LES URLS 19
4.8.1. CLI *19*
4.8.2. Back Office *19*
4.8.3. Front Office *19*
4.9. EXTENSION CORE 19

5. ARBORESCENCE 20

5.1. REPERTOIRES ET FICHIERS A LA RACINE 20
5.2. FICHIER D'AMORÇAGE : INIT.PHP 20
5.3. DETAILS 21
5.3.1. Arborescence /web *21*
5.3.2. Arborescence /lib *22*
5.3.3. Arborescence /var *22*

6. CLASSES DE LA PLATEFORME 23

6.1. PLATFORM 24
6.1.1. Rôle *24*
6.1.2. API *24*
6.2. ERROR 25
6.2.1. Rôle *25*
6.2.2. API *25*
6.3. LOGGER 26
6.3.1. Rôle *26*
6.3.2. API *26*

6.4. REQUEST 27
6.4.1. Rôle *27*
6.4.2. API *27*
6.5. CONTEXT 30
6.5.1. Rôe *30*
6.5.2. API *30*
6.6. EXTENSION 31
6.6.1. Rôle *31*
6.6.2. API *31*
6.7. RESPONSE 32
6.7.1. Rôle *32*
6.7.2. Formats *32*
6.7.3. Templates *32*
6.7.4. API *32*
6.8. DATABASEOBJECT 34
6.8.1. Rôle *34*
6.8.2. Bindings *34*
6.8.3. API *34*
6.8.4. Exemples d'utilisations *36*
6.9. RESULTSET 37
6.9.1. Principe *37*
6.9.2. API *38*
6.9.3. Exemples *39*
6.10. COOKIE 40
6.10.1. Fonction *40*
6.10.2. API *40*
6.11. SESSION 41
6.11.1. Fonction *41*
6.11.2. API *41*

7. ACTIVERECORD 42

7.1. PRINCIPE GENERAL 42
7.2. DEFINITION 42
7.2.1. Création de la table *42*
7.2.2. Création des fichiers classe et paramètres *42*
7.2.3. Modification des paramètres *43*
7.2.4. Reconstruction de la configuration *43*
7.3. INSTANCIATION D'UN ACTIVERECORD 44
7.3.1. getInstance() *44*
7.3.2. Terminologie *44*
7.3.3. Options d'instanciation *44*
7.4. API 45
7.4.1. Attributs *45*
7.4.2. Méthodes *45*
7.4.3. Méthodes et variables statiques *47*
7.5. CREATION DE NOUVELLES METHODES 47
7.6. NAVIGATION 47
7.7. GESTION DU STATUT 47
7.8. RELATIONS ENTRE MODELES 48
7.8.1. Paramètres *48*
7.8.2. Méthodes magiques *48*
7.9. EXCEPTIONS 49
7.9.1. ActiveRecordException *49*
7.9.2. ClassRecordException *49*
7.9.3. UnknownRecordException *49*

7.9.4. DuplicateRecordException 49
7.10. METADONNEES 50
7.10.1. Principe général 50
7.10.2. Types 50
7.10.3. Utilisation 51
7.10.4. Modification 51
7.11. LIMITATIONS 51
7.12. REMARQUES DIVERSES 52
7.13. DESTRUCTION D'OBJET 52
7.14. CLASSES MODEL ET ITEM 52
7.14.1. Classe Model 52
7.14.2. Classe Item 53

8. COLLECTION 55

8.1. ROLE 55
8.1.1. Création 55
8.1.2. Parcours 55
8.1.3. Options : $opts 55
8.1.4. Les filtres : filters 56
8.2. API 57
8.2.1. Méthodes 57
8.3. CAS CONCRETS 59
8.3.1. Chargement de MetaDatas 59
8.3.2. Utilisation de colonnes spéciales dans le filtre 59
8.3.3. Insertion en base de données 59

9. COMPOSANTS 60

9.1. PAGINATION 60
9.1.1. Principe 60
9.1.2. Exemples 60
9.2. ENVOI DE MAIL 61
9.2.1. Paramétrage de l'application 61
9.2.2. API 61
9.2.3. Exemples d'utilisation 62
9.3. CSVREADER 63
9.3.1. Utilisation 63
9.3.2. API 64
9.4. CSVWRITER 64
9.4.1. Utilisation 64
9.4.1. API 64
9.5. FLUX 65

10. ECHANGES HTTP 66

10.1. HTTPCLIENT 66
10.1.1. API 66
10.1.2. Exemples 66
10.2. HTTPREQUEST 66
10.2.1. API 66
10.2.2. Exemples 67
10.3. HTTPRESPONSE 67
10.3.1. API 67
10.3.2. Exemples 68

11. AUTHENTIFICATION 69

11.1. PRINCIPE GENERAL 69

11.2. Informations du contexte 70
11.3. Parametrage 70
11.4. Utilisateurs et Membres 70
11.5. Verrouillage de l'ensemble de l'application 70

12. CACHE 71

12.1. Gestionnaire de cache 71
12.1.1. Cache interne / Cache externe 71
12.1.2. Conseils 71
12.1.3. Vider le cache 71
12.2. Cache client 71

13. API ET WEBSERVICES 72

13.1. Webservices 72
13.1.1. Appel centralisé 72
13.1.2. Appel REST 75
13.1.3. Limitations 75
13.2. Formats des reponses 75
13.2.1. Paramètre kof 75

14. TACHES AUTOMATIQUES 77

14.1. Principe 77
14.2. Blocage 77
14.3. Application 77
14.1. Avertissements 77

15. HOOKS 78

15.1. Hooks de methodes 78
15.2. Hooks d'evenements 78

16. STATISTIQUES ET TRACKING 82

16.1. Tracking 82
16.1.1. Tracking spécifique 82
16.1.2. Tracking d'ActiveRecord 82
16.1.3. Interdiction du tracking 82
16.2. Conversions 82

17. ENTREES / SORTIES 83

17.1. File 83
17.1.1. Utilisation 83
17.1.2. API 83
17.2. Repertoire 84
17.2.1. Utilisation 84
17.2.2. API 84
17.3. Images 85
17.3.1. Utilisation 85
17.3.2. API 85
17.3.3. Exemples d'utilisation 86

18. MOTEUR DE RECHERCHE 87

18.1. Indexation 87
18.2. Recherche 88

19. INTERNATIONALISATION 89

19.1. Parametrage 89

19.2. DICTIONNAIRE 89
19.3. FONCTION L() 89
19.4. APPEL A DES CONTENUS 90
19.4.1. l('snippet:code') 90
19.5. BINDINGS 90
19.6. FORCER UNE LANGUE 90
19.7. ACTIVERECORD ET PROPERTIES 90

20. ROUTAGE 92

21. JAVASCRIPT 93

21.1. KWO.EXEC() 93
21.1.1. Arguments 93
21.1.2. L'utilisation d'un callback 94
21.1.3. Callback + container 96
21.2. KWO.GO() 97
21.3. DEVELOPPEMENT D'UNE FENETRE MODALE 97
21.4. REQUETE AJAX ET FORMULAIRE 98

22. HELPERS 100

22.1. H 100
22.1.1. H ::elt() 100
22.2. XH : XML HELPERS 101
22.2.1. Méthodes 101
22.2.2. Exemples 101
22.3. DH : DATE HELPERS 102
22.4. AH : ARRAY HELPERS 103
22.5. SH : STRING HELPERS 103
22.1. IH : IP HELPERS 104

23. CLASSES STATIQUES 105

23.1. VALID 105
23.2. W : WIDGETS 105
23.2.1. Abus 105
23.2.2. Captcha 105
23.2.3. Choix de langue 106
23.2.4. Commentaire 106
23.2.5. Erreur 106

24. CONVENTIONS ET NORMES 107

24.1. RECOMMANDATIONS GENERALES 107
24.2. PHP 108
24.2.1. Principes généraux 108
24.2.2. Variables 108
24.2.3. Optimisations 108
24.2.4. Nom des fichiers 108
24.2.5. Standards de notation 108
24.2.6. Log 111
24.2.7. Programmation OO 112
24.3. ACTIONS 115
24.3.1. Structure Standard 115
24.3.2. Nommage 115
24.3.3. Règles essentielles à respecter 115
24.4. BASES DE DONNEES 116
24.4.1. Nom de table 116

24.4.2. Colonnes obligatoires ... 116
24.4.3. Ordonnancement des colonnes ... 116
24.4.4. Ecriture des requêtes ... 116
24.4.5. 2 Colonnes spéciales ... 116
24.4.6. Colonnes courantes ... 117
24.4.7. Indexation ... 118
24.4.8. Autres règles ... 118
24.5. HTML ... 120
24.5.1. En-tête HTML ... 120
24.5.2. Conventions ... 122
24.6. JavaScript ... 123
24.6.1. Namespace Kwo ... 123
24.6.2. Référencement ... 123
24.6.3. Notation des objets ... 123
24.6.4. Déclaration des variables ... 123
24.6.5. Passer par les fonctions Prototype ... 123
24.6.6. Exécution d'un JavaScript ... 123
24.6.7. Standards de notation ... 124
24.7. CSS ... 125
24.7.1. Standards de notation ... 125
24.7.2. Id ou Class ... 125
24.8. Templates ... 126
24.8.1. Standards de notation ... 126
24.8.2. Constantes ... 126
24.8.3. Conventions PHP ... 126

25. INSTALLATION ... 128

25.1. Creation de l'arborescence ... 128
25.2. Environnement ... 128
25.3. Mise en place du Virtualhost ... 129
25.4. Base de donnees ... 129
25.5. Parametrage de l'application ... 129
25.6. Tache automatique ... 130
25.7. Droits ... 130
25.8. Emplacements des fichiers ... 130
25.9. Fichier de log ... 130
25.10. Diagnostique ... 131

26. SECURITE ... 132

26.1. Base de donnees ... 132
26.2. Templates ... 132
26.3. Log ... 132

27. OUTILS DE DEVELOPPEMENT ... 133

27.1. Unicode ... 133
27.2. Editeur ... 133

28. DEFINITIONS ... 134

29. INDEX ... 135

1. PREFACE

1.1. Chemins

La majorité des chemins de type `etc`, `lib`, `var`, `pub`, `web` sont relatifs au `DOCUMENT_ROOT` du projet[1] (ex. `/var/web/projet`).

1.2. Types de données

PHP regroupe sous le type `array` les tableaux associatifs et scalaires.

Pour plus de clarté, cette documentation utilise la terminologie suivante :

`hash`	Les clefs sont obligatoires et de type « chaînes de caractères ». La variable `$h` est souvent utilisée pour stocker un tableau associatif. Il s'agit d'un tableau associatif.	`array('nom'=>'dupont','prenom'=>'paul')`
`array`	Les clefs sont numériques et non significatives.	`array(1,2,array('x', 'y'))`
`set`	Les clefs sont numériques et non significatives ; les valeurs sont de même type.	`array(1,2,3)`

1.3. Conventions typographiques

Chemin	`lib/sys`
URL	`/web`
Code	`$req->x`
Classe	`ActiveRecord`
Les termes anglais	*English text*
Bloc de code	`$topic = Topic::getInstance(1);` `$ctx->topic = $topic;`
Rubrique Espace Admin	***Rubrique > Sous-rubrique***

[1] Ce répertoire doit contenir le script init.php

2. INTRODUCTION

KWO est une plateforme (*framework*) de développement d'applications web.

2.1. Notion de framework

Toute action (*Business Operation*) s'appuie sur un large panel de classes centrales (`Platform`, `Extension`, `AuthManager`, `Request`, `Response`, `DataBaseObject`, `ActiveRecord`, `Collection`, etc.).

2.2. Organisation d'un projet

L'utilisation massive des templates ainsi que la séparation des extensions, modèles et actions permettent l'organisation suivante :

- Chef de projet : Il prend le briefing auprès du commercial et suit l'évolution du projet. Il serait bon que cette personne connaisse les principes de l'Agile Programming.
- Architecte : Il conçoit la base et les tables, identifie les extensions (existantes et/ou à développer), les modèles, les actions. Il développe également le back office. Il réalise des revues de code tous les matins avec ses développeurs.
- Développeur : Il développe les actions en s'appuyant sur les modèles et initialise les templates. Il contrôle régulièrement les templates modifiés par les graphistes.
- Graphiste : Il est en mesure de modifier les templates. Ses retours sont fréquents auprès du développeur et du client. Il doit faire en sorte de customiser au maximum des interfaces normalisées. La nouvelle architecture permet au graphiste (et au client ?) d'avoir un accès FTP aux images et aux templates.

3. KWO EN UNE PAGE

Pour assurer une organisation optimale du code source, un Site basé sur KWO sépare ses différentes vues en « actions ». Toute fonctionnalité (ex. liste de produits, affichage d'un produit, ajout au panier, etc.) dispose d'une action spécifique associée. Techniquement parlant, une action correspond à un script PHP dont l'extension est `.inc`.

Toute action a trois objectifs :

- récupération des paramètres nécessaires (en utilisant des objets tels `Request` `$req`, `Cookie`, `Session`) ;
- réalisation d'une opération (ex. appel à une méthode d'un objet métier, enregistrement en base de données `$dbo`, échange avec un webservice) ;
- configuration de la réponse (objet `Response` `$res`).

Dans la grande majorité des cas, une réponse correspond à une page HTML qui sera affiché directement dans le navigateur à l'origine de la requête. Un des principes fondateurs de KWO consiste à séparer rigoureusement le « fonctionnel » du « visuel ». Pour y parvenir une action fait appel à un fichier extérieur qui contient le code HTML. Un tel fichier est appelé un *template*. Chaque action dispose ainsi de son template associé. Un template correspond également à un script PHP dont l'extension est cette fois `.psp`. Le transfert d'informations entre *action* et *template* repose sur l'utilisation de l'objet `Context` (`$ctx`). Les attributs de `$ctx` (ex. `$ctx->var1`) sont directement accessibles en tant que variables dans le template (`$var1`).

Les adresses sont structurées de la manière suivante : `http://domain/<ACTION>`. Une telle adresse est traitée par l'action : `lib/<APP_EXTENSION>/front/<ACTION>.inc`. Le template associé correspond au script : `web/<APP_EXTENSION>/templates/<ACTION>.psp`.

Réalisons un exemple consistant à additionner deux valeurs. L'extension associée à l'app porte le nom `calc`.

La première action, `formulaire`, propose deux champs textuels et un bouton.

`lib/calc/front/formulaire.inc`

```
<?php

try {
  $res->useTemplate();
}
catch (Exception $e) { $err->add($e); }
```

`web/calc/templates/formulaire.psp`

```
<form action="/calc/addition">
 <input name="x" />
 <input name="y" />
 <input type="submit" />
</form>
```

La seconde action, `addition`, récupère les deux valeurs et affiche le résultat.

`lib/calc/front/addition.inc`

```
<?php

try {
  $resultat = $req->x + $req->y;
  $ctx->resultat = (int) $resultat;
  $res->useTemplate();
}
catch (Exception $e) { $err->add($e); }
```

`web/calc/templates/addition.psp`

```
résultat : <strong><?=$resultat?></strong>
```

Chaque action peut avoir un style et script chargés automatiquement. Il convient pour cela de respecter la convention suivante :

`web/<APP_EXTENSION>/scripts/<ACTION>.js` et `web/<APP_EXTENSION>/styles/<ACTION>.css`.

4. FONCTIONNEMENT

Ce chapitre vise à présenter le mode de fonctionnement d'une application KWO.

4.1. MVC

La plateforme fait un usage intensif du pattern *MVC* avec la séparation du développement en trois couches.

Model	Objets métiers nécessaires au fonctionnement d'une App. L'ensemble de l'intelligence (règles, contraintes, algorithmes, requêtes aux bases de données) doit être contenu dans ces objets. Pour une extension de type e-boutique, les modèles sont `Marque`, `Produit`, `Panier`, `Commande`, etc. Classes du framework mises en œuvre : `ActiveRecord`, `Collection`, `DataBaseObject`.
View	Cette couche prend en charge l'aspect visuel du projet. Classes du framework mises en œuvre : `Response` (`$res`), `Context` (`$ctx`), `Template`, `W`, `H`.
Controller	Cette couche se charge du chargement des paramètres et du routage d'une requête vers l'extension et l'action associées. L'authentification y est également prise en compte. Classes du framework mises en œuvre : `Platform`, `Application`, `Extension`, `Controller`, `AuthManager`, `ApiManager`, `AccountManager`, `Router`, `Request` (`$req`), `Logger` (`$log`), `Error` (`$err`), `Context` (`$ctx`).

4.2. Convention over Configuration

KWO privilégie les conventions aux fichiers de configuration (principe CoC).

Ex. nommage des tables, des champs, emplacement des fichiers, etc.

4.3. Scopes

Différents modes (`scopes`) permettent d'accéder aux fonctionnalités (*Business Operation*) de la plateforme :

Front	*Espace Public* Partie libre de l'application accessible à tous les utilisateurs. Elle contient également les zones à accès restreints telles que la « Gestion de compte ». Ex. `http://domain/products` L'extension correspond toujours au paramètre `app.extension`. Il est possible de préciser une action par défaut avec le paramètre `front.home`.

Account	*Espace Membre* Il s'agit de l'accès restreint réservé aux utilisateurs authentifiés du site. Les URLs commencent par `/account`. L'authentification est automatiquement contrôlée pour chaque action. Ex. `http://domain/account/orders` L'extension correspond toujours au paramètre `app.extension`. Il est possible de préciser une action par défaut avec le paramètre `account.home`.
Middle	*Espace Partenaire* Accès restreint réservé aux partenaires. L'authentification est automatiquement contrôlée pour chaque action. Ex. `http://domain/middle/orders` L'extension correspond toujours au paramètre `app.extension`. Il est possible de préciser une action par défaut avec le paramètre `middle.home`.
Back	*Espace d'administration* Il s'agit de la zone réservée aux administrateurs. Les URLs commencent par `/back`. Tout accès Back nécessite une authentification. Ex. `http://domain/back/forum/topic.home`
CLI	*Command Line Interface* Il s'agit des scripts exécutés en ligne de commande Une action CLI peut être déclenchée de la manière suivante : `> php init.php /tracker/generate`
API	*API* Ex. `http://domain/1/topic.stat` La valeur 1 correspond à la version de l'API. Cette valeur peut être comprise entre 1 et 99. L'extension correspond toujours au paramètre `app.extension`.

4.4. Application / App

KWO est une plateforme multi applications. Une même instance peut regrouper plusieurs sites. Une application est identifiée par son nom de domaine associé.

Dans la version actuelle, les sites partagent toutefois les mêmes données (statistiques, utilisateurs).

L'idée est donc d'utiliser cette dimension multi applications dans les cas suivants :

- incubation de projets au sein de l'entreprise ;
- mise en place d'un site principal avec des sous sites (ex: un blog, une mini-boutique).

4.4.1. Fichier de configuration

Chaque application dispose d'un fichier de configuration qui lui est propre, placé dans `etc/app`.

Il s'agit d'un script PHP formaté de la façon suivante :

```
$parameter['param1'] = 'val1';
$parameter['param2'] = 'val2';
```

4.4.2. Paramètres

`account.home`		Action appelé lorsqu'aucune action n'est précisée (ex. `/account`).
`account.modules`		
`account.template`		Template alternatif pour l'espace membre
`api.key.google.maps`		
`app.domain`		Nom de domaine associé au site.
`app.extension`		Extension principale, elle qualifiée d'*extension projet*.
`app.locale`		Permet de forcer la langue du Site (langue par défaut). Ex. `I18N::LOCALE_FR`
`app.locales`		Locales disponibles sur le site. Ex. `array(I18N::LOCALE_FR, I18N::LOCALE_EN)`
`app.manager`		Nom de la classe statique qui contient les *hooks* généraux.
`app.name`		Code associé à l'application
`app.regexp`		Expression régulière permettant de « matcher » les différents noms de domaine associés au Site.
`app.scopes`	array	Scopes autorisés.
`app.url.port`		
`app.url.secure`		
`back.locales`	array	Locales à proposer en back office (en vue d'une nouvelle traduction du Site).
`back.url`		Permet de préciser une adresse alternative pour l'espace d'administration.
`contact.admin`		Adresse email de l'administrateur principal. Il est susceptible de recevoir des messages d'alerte en cas de dysfonctionnement.
`contact.devel`		Adresse email du développeur principal. Il reçoit les alertes générales ainsi que les alertes techniques.
`extension.search.models`		Model à indexer dans le moteur de recherche. Ex. `array(I('page'), I('topic'))`
`front.breadcrumb`		Indique que le Site dispose d'un fil d'Ariane.
`front.cache.ttl`	Nb sec	Permet la mise en œuvre du cache pour toutes les actions front (en dehors de celles appelées en XHR).
`front.feeds`		Hash des flux RSS

`front.home`		Action par défaut
`front.metas`		Hash de métadonnées
`front.routes`		Gestion du routage.
`front.template`		Template squelette de l'application. Ex. `'test:skeleton.front'`
`front.templates`		Redéfinition de certains templates Ex. `array('shop' => array('catalog' => 'test: list'))`
`host.secure`	bool	Indique si le Site n'est accessible qu'en HTTPS.
`mail.bounce`		Adresse email qui recevra les messages d'erreur (*bounce*)
`mail.from.email`		Adresse email pour le From
`mail.from.name`		Nom pour le From
`mail.smtp.default`		Serveur d'envoi (localhost par défaut)
`mail.template`		Template du mail
`middle.home`		Action appelée par défaut lorsque l'url se limite à `/middle`
`middle.modules`		
`middle.template`		
`profile.customer`		
`profile.user`		

Tout changement de paramètre doit être suivi d'une reconstruction de la configuration[2].

```
php init.php build
```

D'autres paramètres sont évoqués dans les chapitres consacrés à l'authentification, au moteur de recherche.

Deux types de paramètres sont disponibles dans KWO :

- Les paramètres présents dans le fichier de configuration de l'application. Le développeur y accède avec la fonction `P()`. Ces paramètres ne sont pas censés changer pendant l'exploitation du Site.
- Les paramètres gérés dans l'espace d'administration (rubrique Système > Paramètres). Accès : `P()`. Ces paramètres sont susceptibles d'être modifiés par l'administrateur du Site.

Le fichier `etc/platform.conf.inc`, s'il est présent, permet de renseigner des paramètres qui seront partagés par toutes les apps.

[2] Un build génère notamment les deux fichiers suivants : `etc/inc/platform.conf.inc` et `etc/inc/dbcache.conf.inc`

```
<?php
$parameter['app.seed'] = 'XXXXXXXX';
$parameter['app.dsn.host'] = 'localhost';
$parameter['app.dsn.name'] = 'xxx';
$parameter['app.dsn.login'] = 'yyy';
$parameter['app.dsn.password'] = 'zzz';
```

4.5. Extension

Toute action est attachée à une extension. La seule extension obligatoire est `core`.

4.5.1. Paramétrage

Les paramètres de configuration d'une extension sont situés dans un fichier

`etc/ext/<EXT>/parameters.conf.inc`

```
$parameter['id'] = 1250;
$parameter['name'] = 'shop';
$parameter['label'] = 'boutique';
$parameter['table_prefix'] = 'shp_';
$parameter['back'] = array('commandes' => 'order.home',
                           'produits' => 'product.home',
                           'devises' => 'currency.home',
                           'zones' => 'zone.home',
                           'marchand' => 'merchant.edit?id=1',
                           'coupon' => 'coupon.home');
```

4.6. Actions

Tout accès à la plateforme correspond à une action. Une action peut être qualifiée de *Business Operation*.

Une action appartient à une extension (*<EXT>*).

Les actions back office sont placées dans `lib/<EXT>/back/<ACTION>.inc`; les actions front office sont, quant à elles, placées dans `lib/<EXT>/front/<ACTION>.inc`.

Une action dispose d'un accès direct aux variables suivantes :

`$ctx`	*Context*	Le contexte
`$dbo`	*DataBaseObject*	La base de données
`$err`	*Error*	L'erreur
`$log`	*Logger*	Le logger
`$req`	*Request*	La requête
`$res`	*Response*	La réponse

L'action *affiche* de l'extension *test* qui consisterait à afficher la valeur `x` passée en paramètre serait placée dans `lib/test/front/affiche.inc` et serait appelée de la façon suivante : `http://domain/affiche?x=3`

L'affichage du paramètre `x` pourrait être réalisé ainsi :

```
$res->write($req->x);
```

Une action fonctionne généralement de la manière suivante :

1. Bloc `try .. catch` pour récupérer les `Exception` et gérer les erreurs comme il se doit ;
2. Utilisation de `$req->getUser()`, `$req->getUserId()` et récupération éventuelle de l'exception `AuthException` si l'action nécessite une authentification ;
3. Instanciation d'un objet modèle (cf. `ActiveRecord`) le plus souvent à l'aide d'un `id` passé en paramètre ;
4. Passage de paramètre à une des méthodes du modèle ;
5. Initialisation du contexte (`$ctx`) avec les données nécessaires à l'affichage de la réponse ;
6. Utilisation de l'objet réponse `$res` pour initialiser le titre, les éventuels templates ou retourner du code JavaScript (si requête AJAX).

4.6.1. Cas d'utilisation

4.6.1.1. Utilisation d'un template

```
try {
  $ctx->x = $req->x;
  $res->setTitle('Hello');
  $res->useTemplate();
}
catch (Exception $e) { $err->add($e); }
```

4.6.1.2. Utilisation de deux templates

```
try {
  $ctx->x = $req->x;
  $res->setTitle('Hello');
  $res->useTemplates('template1', 'template2');
}
catch (Exception $e) { $err->add($e); }
```

4.6.1.3. Action sécurisée

```
try {
  $user = $req->getUser();
  $ctx->x = $req->x;
  $res->setTitle('Hello');
  $res->useTemplate();
}
catch (AuthException $e) { $res->sendAuthenticationForm(); }
catch (Exception $e) { $err->add($e); }
```

4.6.1.4. Action appelée en AJAX

```
try {
  $res->sendMessage('Opération réalisée avec succès !');
}
catch (Exception $e) { $err->add($e); }
```

4.6.1.5. Action appelée en AJAX + Callback

```
try {
  $ctx->x = $req->x;
}
catch (Exception $e) { $err->add($e); }
```

4.7. Templates

4.7.1. Actions et templates

Une action peut faire appel à un template à l'aide des fonctions suivantes :

`$res->useTemplate()`	Le template appelé porte le nom de l'action et se trouve dans le répertoire templates (`web/<EXT>/templates`) de l'extension en cours.
`$res->useTemplates('template1', 'template2')`	Les différents *templates* vont être appelés les uns derrière les autres. Deux valeurs spéciales sont disponibles : `null` : annule tous les *templates* précédents `true` : correspond au nom de l'action

Les variables placées dans le `Context` (`$ctx`) sont alors directement disponibles au sein d'un template.

4.7.2. Appel d'un template au sein d'un autre template

Cela repose sur la méthode `H::inc()`.

- Premier argument : un nom de template ou un tableau de noms de templates.
- Second argument : un *hash* de valeurs (`array` ou `object`) ou `true` pour propager le `Context` (`$ctx`). La valeur `true` est celle par défaut.

```
<div><?=H::inc('product.panel')?></div>
```

4.7.3. Class Template

La méthode `Template::render()` fonctionne comme `H::inc()`. Elle sera utilisée dans les classes ou les actions (les méthodes statiques de la classe `H` étant en effet réservées aux templates).

4.7.4. Conventions

`templateX`	`web/<EXT>/templates/templateX.psp`
`extensionX:templateX`	`web/<EXT>/templates/templateX.psp`
`//templateX`	`lib/core/templates/templateX.psp`
`//extensionX:templateX`	`lib/<EXT>/templates/templateX.psp`
`null`	Remise à zéro de la liste des templates.
`true`	`web/<EXT>/templates/<ACTION>.psp`

Plusieurs *templates* peuvent être transmis, l'argument devient alors un tableau de *templates*.

```
<?=H::inc(array('template1', 'extension1:template2'), $bindings)?>
```

4.7.5. Redéfinition d'un template

Il est possible de redéfinir le template utilisé par défaut par la méthode `$res->useTemplate()`. Le paramétrage est réalisé au sein du fichier de configuration de l'application avec l'élément `templates`.

```
$parameter['templates'] = array('content' => array('page.view' => 'agri:page.view'));
```

L'exemple précédent fera en sorte que l'action `/page.view/-/id/N` passe par le template `web/agri/templates/page.view.psp` plutôt que par le template par défaut : `web/content/templates/page.view.psp`.

Il également possible d'utiliser un tableau de templates. Cette technique est particulièrement utile pour ajouter une décoration autour du template de base.

```
$parameter['templates'] = array('cms' => array('forum' => array('myapp:roundbox', 'forum'));
```

4.8. Les URLs

4.8.1. CLI

L'appel d'une action en ligne de commande nécessite de se placer à la racine du projet (cf. `DocumentRoot`).

La commande est ensuite :

```
php init.php extension/action
```

La partie `extension/` peut être évitée s'il s'agit de l'extension `core`.

```
php init.php cache.clean
```

Les paramètres sont passés de la manière suivante : `param1=val1 param2=val2`

```
php init.php extension/action param1=val1 param2=val2
```

4.8.2. Back Office

L'accès au *back office* se fait par l'URL suivante : `http://domain/back`.[3]

Les actions *back office* sont toujours préfixées par `/back` suivi du nom de l'extension et du nom de l'action (`/back/extension/action`).

Ex. `http://domain/back/forum/topic.view?id=1`

4.8.3. Front Office

Une URL *front office* est constituée du nom de l'extension suivi du nom de l'action.

Ex. `http://domain/topic.view?id=1`

4.9. Extension CORE

Cette extension contient les classes du framework dans son répertoire `lib/core/class`.

Il s'agit de la seule extension obligatoire du framework.

[3] Il reste possible de définir une URL alternative de connexion au Back Office à l'aide du paramètre :

$parameter['back.url']

5. ARBORESCENCE

5.1. Répertoires et fichiers à la racine

`doc`	Les fichiers déposés par les administrateurs. Lorsqu'un projet est initialisé au sein de l'arborescence *master*, les fichiers correspondant à l'application doivent être regroupés dans un répertoire portant le nom du projet (ex. `doc/credit` pour l'application « credit »). Cela assure que les fichiers seront bien conservés au moment de la duplication d'instance. Ce répertoire ainsi que les sous-répertoires doivent appartenir à `apache:apache`.
`etc`	Ce répertoire contient les fichiers de configuration. Le répertoire `etc/app` contient un fichier de configuration par application. Le répertoire `etc/ext` contient un répertoire de fichiers de configuration par extension. Ce répertoire ainsi que les sous-répertoires doivent appartenir à `apache:apache`. `etc/inc` contient quant à lui une version optimisée des fichiers de configuration liés aux modèles.
`init.php`	Script d'amorçage de la plateforme par lequel passe toute requête. Il peut également être appelé *bootstrap*.
`lib`	Il s'agit du cœur de KWO. Ce répertoire contient toutes les extensions de l'application. Chaque extension est composée d'une dimension code situé dans `lib` et d'une dimension visuelle située dans `web` (images, templates). L'objectif est de limiter les modifications à l'ajout de nouvelles extensions.
`pub`	Fichiers publics spécifiques à l'applicatif et devant être accessibles en téléchargement. (ex. PDF pour un site de téléchargement de « dossiers de presse »).
`usr`	Ce répertoire contient les fichiers déposés par les utilisateurs. Chaque utilisateur dispose d'un répertoire. Un fonctionnement reposant sur du *hashing* permet d'envisager des sites disposant de plusieurs millions d'utilisateurs. Ce répertoire ainsi que les sous-répertoires doivent appartenir à `apache:apache`.
`var`	Ce répertoire contient les éléments susceptibles de changer durant la vie du site : sessions, cache, logs, traductions (`locales`). Ce répertoire ainsi que les sous-répertoires doivent appartenir à `apache:apache`.
`web`	Ce répertoire contient les différents styles, éléments graphiques (images, fonts, flash), scripts et templates liés à chaque extension. Un objectif de KWO est de limiter l'adaptation d'une extension pour une application donnée à des modifications dans ce répertoire. Ce répertoire pourra, à terme, être rendu accessible aux clients en FTP. Sous-répertoires standards : `web/<EXT>/files`, `web/<EXT>/fonts`, `web/<EXT>/images`, `web/<EXT>/scripts`, `web/<EXT>/styles`, `web/<EXT>/templates`.

5.2. Fichier d'amorçage : init.php

Il charge le fichier de configuration placé dans `etc/inc/platform.conf.inc`.

Il contient également les fonctions globales :

- `c()` : équivalent de ucfirst() ;
- `l()` : traduction des mots ;
- `h()` : pour l'échappement de caractères dangereux ;

- `S()` : accès à un singleton de la plateforme ;
- `P()` : accès à la valeur d'un paramètre ;
- `T()` : table associée à un modèle.

Le rôle de ce script est avant tout de charger la classe `Platform` qui initialise quant à elle l'ensemble du framework.

Un certains nombre de constantes y sont également définis :

```
APP_PATH
CORE
DOC_PATH
DOC_URL
LIB_PATH
PIX_URL
TMP_PATH
USR_PATH
USR_URL
VAR_PATH
WEB_PATH
WEB_URL
```

5.3. Détails

5.3.1. Arborescence /web

Chaque extension contient différents fichiers et répertoires.

`web/<EXT>/fonts`	Typos.
`web/<EXT>/images`	Les images utilisées au sein des templates de l'extension. Le répertoire `images` de l'extension d'application, contient les fichiers `favicon.png` et `favicon.ico`.
`web/<EXT>/scripts`	Ce fichier contient l'ensemble du code JavaScript lié à une extension. Les différentes fonctions sont regroupées dans des objets au sein du *namespace* Kwo. Il est probable qu'un objet par modèle soit nécessaire. Ce script peut être inclus dans la page à partir du header ou au sein d'un template.
`web/<EXT>/styles`	Les styles liés à l'extension.
`web/<EXT>/templates`	Les templates (`.psp`) liés à l'extension. Il s'agit de fichier HTML pouvant contenir du PHP. Il contient les templates des scopes front, account, et middle.

web/<EXT>/files	Fichiers « textuels » accessibles depuis l'extérieur (ex. crossdomain.xml, robots.txt). Une redirection automatique est réalisée par Apache sur les accès aux fichiers .txt ou .xml placés à la racine du site. Ex. un accès à http://domain/robots.txt charge automatiquement le fichier web/<EXT>/files/robots.txt.

L'extension core contient les images utilisées dans le back office ainsi que les éléments communs à l'ensemble des extensions (templates, JavaScripts).

5.3.2. Arborescence /lib

lib/<EXT>/account	Actions « Espace Membre »
lib/<EXT>/back	Actions « Espace d'administration »
lib/<EXT>/class	Classes modèles, classes statiques
lib/<EXT>/cli	Actions en ligne de commandes + tâches automatiques.
lib/<EXT>/infos	Ce répertoire contient les documentations ainsi que les fichiers temporaires ou de sauvegarde. Il est en effet interdit de laisser des fichiers de travail (old, safe, backup, etc.) dans un autre répertoire.
lib/<EXT>/front	Actions de l'« Espace Public »
lib/<EXT>/middle	Actions du Middle Office
lib/<EXT>/templates	Templates Espace d'administration (rarement utilisés).
lib/<EXT>/xml	Les interfaces pour l'espace d'administration.

5.3.3. Arborescence /var

Tous les sous-répertoires et fichiers contenu dans /var doivent appartenir à apache:apache.

var/cache	Les fichiers de cache.
var/locales	Dictionnaires de mots de l'application. Générés à partir des éléments créés en BO dans *Cms > Phrases*
var/log	Différents fichiers d'erreur.
var/run	PID du TaskServer.
var/session	L'utilisation du *hashing* impose le nettoyage manuel du répertoire. La tâche session.clean (lib/core/cli) exécutée toutes les heures se charge de cela.
var/tmp	Répertoire utilisé dès que nous avons besoin d'un fichier temporaire. Les fichiers temporaires d'upload y sont par exemple déposés (paramétrage dans le virtualhost).

6. CLASSES DE LA PLATEFORME

Le framework peut être assimilé à un ensemble de classes de haut niveau interagissant les unes avec les autres.

Les méthodes publiques offertes par ces classes constituent l'API de KWO qu'il conviendra, dans la mesure du possible, de maintenir[4] afin de pouvoir remplacer l'intégralité du framework sans risquer de bloquer les extensions. L'idée est également de pouvoir venir greffer des extensions à n'importe quel étape de la vie d'un site (du moment que l'API est compatible).

Le but est maintenant :

- d'éviter de toucher au cœur du système (framework) afin de maintenir l'API ;
- d'ajouter de nouvelles fonctionnalités en créant des extensions et des modèles réutilisables.
-

[4] Les méthodes JavaScript proposées dans kwo.js font également parties de cette API.

6.1. Platform

6.1.1. Rôle

L'instanciation de cette classe entraîne :

1. la création des objets : `$err`, `$log`, `$app`, `$req`, `$ctx`, `$res`, `$dbo` ;
2. le chargement de l'extension (`Extension`) associée à la requête `$req` et l'exécution de l'action appelée (*dispatch*) ;
3. l'affichage de la réponse `$res`.

Le singleton `Platform` est ensuite utilisé comme support pour le registre d'objets (cf. `S()` qui est un alias de `Platform::getSingleton()`).

6.1.2. API

6.1.2.1. Méthodes et variables statiques

`getSingleton()`	$name	La fonction globale `S()` est un alias de `getSingleton()` et doit être préférée.
`hasSingleton()`	$name	
`setSingleton()`	$name $obj	

6.2. Error

6.2.1. Rôle

Cette classe est utilisée comme réceptacle des messages d'erreur restitués dans la réponse.

L'objet `$res` se charge de présenter les messages d'erreur dans le bon format de réponse (XML-RPC, JSON-RPC, etc.).

Le template `web/core/templates/error.psp` est utilisé dans le cadre d'une réponse HTML en front.

6.2.2. API

6.2.2.1. Méthodes

`add()`	$msg $code=null	Ajoute une erreur. `$msg` peut être un objet `Exception`.
`count()`		Retourne le nombre d'erreurs.
`get()`		Retourne le tableau des erreurs.
`getCode()`		Retourne le code de l'erreur (>=0).
`setCode()`	$value	La valeur doit être supérieure ou égale à 1.

```
try {

}
catch (Exception $e) { $err->add($e); }
```

6.3. Logger

6.3.1. Rôle

Cette classe est utilisée pour tracer des messages dans le fichier de logs (`var/logs/error`).

L'action CLI log permet de visualiser les erreurs.

```
> php init.php log
```

Cinq niveaux de traçage sont disponibles :

- `DEBUG` ;
- `TRACE` ;
- `WARN` ;
- `ERROR` ;
- `FATAL`.

Par convention, une erreur tracée au sein d'une classe doit être préfixée avec le nom de la classe et de la méthode. L'utilisation de la constante `__METHOD__` est vivement conseillée.

```
if ($rs === false || $rs->numRows() < 1) {
  $this->log->error('invalid shipping_id ('.$shipping_id.')', __METHOD__);
  throw new Exception('invalid shipping_id');
}
```

Les erreurs d'URL (fichiers inexistants) sont quant à elles enregistrées dans le fichier `var/logs/url`.

6.3.2. API

6.3.2.1. Méthodes

`backtrace()`		
`bench()`		Permet d'afficher l'état de la mémoire et de la charge machine.
`debug()`	$arg, $prefix=null	Accepte n'importe quel format en argument et en affiche, dans les logs, une version lisible.
`error()`	$msg, $prefix=null	Trace le message en mode `ERROR`.
`fatal()`	$msg, $prefix=null	Trace le message en mode `FATAL`.
`trace()`	$msg, $prefix=null	Trace le message en mode `TRACE`.
`warn()`	$msg, $prefix=null	Trace le message en mode `WARN`.

6.4. Request

6.4.1. Rôle

Au moment de son instanciation, cet objet se charge de :

- récupérer les en-têtes (*headers*) ;
- prendre en compte les routes ;
- détecter l'extension et l'action ;
- initialiser les paramètres passés à l'action.

Cet objet permet avant toute chose d'accéder aux paramètres transmis avec la requête. En virtualisant leur accès, le code reste inchangé quel que soit le mode d'appel :

- web (méthode *GET* ou *POST*) ;
- ligne de commande (*CLI*) ;
- web service (*XML-RPC*, *JSON-RPC* ou *SOAP*).

Les paramètres passés à une requête deviennent les attributs de l'objet `$req`.

Pour le web deux méthodes permettent de transmettre les paramètres :

- avec la QueryString : `http://domain/action?x=12&y=2`
- au sein de l'URL : `http://domain/action/-/x/12/y/2`

Ce second mode est particulièrement intéressant dans le cadre du référencement.

Dans le cas d'une exécution en ligne de commande (CLI) les paramètres sont passés de la façon suivante :

```
php init.php /extension/action param1=val1 param2=val2
```

6.4.2. API

6.4.2.1. Attributs

Ils correspondent aux paramètres transmis à l'action.

Avec une url du type `http://domain/divide/-/x/4?y=2`, `$req->x` contient `4` et `$req->y` contient `2`.

6.4.2.2. Méthodes

`comesFrom()`	$regexp	Indique si la requête provient d'une URL correspondant à `$regexp`.
`debug()`		Affiche dans les logs l'ensemble des attributs de la requête.
`getAction()`		Retourne le nom de l'action.
`getAttribute()`	$key	
`getAttributes()`	$regexp=null	La *regular expression* permet de filtrer les attributs retournés en fonction de leur nom.
`getBody()`		Retourne le corps de la requête HTTP. (cf. `php://input`)

`getChecksum()`	$type='CRC32'	Retourne une « empreinte » de la requête. Cette empreinte prend en compte l'URL ainsi que les paramètres (triés). Autre type disponible : MD5.
`getClient()`	$key=null	Permet d'obtenir des informations sur le client. $key : `ip` (address au format nul), `referer`, `address`, `host`, `agent`, `country` (country_id), `login`, `passord`
`getContentType()`		
`getCookie()`	$name=Cookie::NAME $ttl=null	Retourne un objet `Cookie`.
`getExtension()`		Retourne le nom de l'extension.
`getHeader()`	$key	Retourne un en-tête HTTP.
`getHeaders()`		Retourne un hash contenant les différents en-têtes HTTP de la requête.
`getLocale()`		
`getLocation()`		Ex. `/action`
`getMethod()`		Retourne `GET` ou `POST`.
`getParameter()`	$key	
`getParameters()`		
`getPath()`		Ex. `/action`
`getQueryString()`		
`getScope()`		Retourne le nom du scope (front, back, cli ou member).
`getServer()`	$key=null	$key : host, port
`getSession()`		Retourne la session courante.
`getUri()`		Ex. `/action?param=val`
`getUrl()`		Ex. `http://domain/action`
`getUser()`		Cf. chapitre Authentification
`getUserId()`		Cf. chapitre Authentification
`getVisitId()`		Il s'agit de la `visit_id`.
`hasAttribute()`	$key $has_content=false	Indique si la requête contient le paramètre `$key`. En passant `true` en second argument, une vérification est également faite pour savoir si le paramètre est non vide.
`hasAttributes()`	$key1, $key2, …	Si le dernier argument `true`, une vérification est faite sur le caractère non vide des paramètres.
`hasCookie()`	$name=Cookie::NAME	
`hasHeader()`	$key	
`hasParameter()`	$key	
`hasSession()`		Indique si une session a déjà été associée à cet utilisateur.
`hasValidCaptcha()`		

`isAccount()`		Indique si la requête est faite sur le scope « account ».
`isAuthenticated()`		Cf. chapitre Authentification
`isBack()`		Indique si la requête est faite sur le scope « back ».
`isCli()`		Indique si la requête est faite sur le scope « cli ».
`isFront()`		Indique si la requête est faite sur le scope « front ».
`isHttp()`		Indique si la requête est de type HTTP.
`isIe()`		Indique si la requête provient du navigateur Internet Explorer.
`isMiddle()`		
`isNewVisit()`		
`isNewVisitor()`		
`isRpc()`		Indique si la requête est de type XML-RPC ou SOAP.
`isXhr()`	$type=null	Indique s'il s'agit d'une requête de type `XmlHttpRequest`. Un type (cf. constantes) peut être transmis pour savoir s'il s'agit d'un `exec` ou d'un `update`.
`match`	$regexp	Indique si l'URL de la page appelée correspond à l'expression régulière.
`removeCookies()`		
`setAttribute()`	$key, $value	
`setParameter()`	$key, $value	
`track()`		Cf. chapitre Statistiques et Tracking

6.4.2.3. Constantes

`XHR_EXEC`	Requête `XmlHttpRequest` de type *exec*.
`XHR_UPDATE`	Requête `XmlHttpRequest` de type *update*.

```
if ($req->isXhr(Request::XHR_UPDATE)) {

}
```

6.5. Context

6.5.1. Rôe

Le contexte `$ctx` a pour rôle de collecter les données qui seront nécessaires à l'affichage de la réponse (`$res`).

6.5.2. API

6.5.2.1. Attributs

Ils correspondent aux données qui sont enregistrées dans le contexte.

Le contexte contient par défaut les variables suivantes :

`_extension`	Ex. forum
`_action`	Ex. topic.view
`_app`	Ex. kwo
`_locale`	Ex. fr

6.5.2.2. Méthodes

`debug()`		Affichage du contenu du contexte dans les logs.
`getAttribute()`	$key	Identique à `$ctx->x`.
`getAttributes()`	$filter_flag=false	Si la valeur `true` est passée, les attributs commençant par _ ne sont pas retournés.
`hasAttribute()`	$key	
`setAttribute()`	$key $value	Identique à `$ctx-> x = 1`.
`setAttributes()`	$arg	Importe argument de type objet, hash, ActiveRecord etc.

6.6. Extension

6.6.1. Rôle

Essentiellement utilisée pour tester si une extension est installée :

```
if (Extension::exists('shop')) {
}
```

6.6.2. API

6.6.2.1. Attributs

Ils contiennent les paramètres associés à l'extension.

6.6.2.2. Méthodes

`getId()`		Affichage du contenu du contexte dans les logs.
`getModels()`		
`getTables()`		

6.6.2.1. Méthodes statiques

`all()`		Retourne un hash contenant l'ensemble des extensions de l'application. La clef correspond au nom de l'extension et la valeur à un hash contenant les différents paramètres associés à l'extension.
`enumerate()`		Identique à `$ctx->x`.
`exists()`	$name1, $name2, …	Retourne `true` si toutes les extensions passées en arguments existent.
`getInstance()`	$extension_name	
`id()`	$extension_name	
`name()`	$extension_name	
`parameter()`	$extension_name $key	
`parameters()`	$extension_name	

6.7. Response

6.7.1. Rôle

Cet objet est responsable de l'affichage de la réponse.

6.7.2. Formats

Deux méthodes permettent de forcer le format de la réponse :

1. passer le paramètre `kof` dans l'url de la requête,
2. ajouter l'entrée `X-Knx-Kof`.

Ces formats sont décrits plus précisément dans le chapitre consacré aux WebServices.

6.7.3. Templates

L'objet `$res` peut faire appel à un template en utilisant les méthodes `useTemplate()` ou `useTemplates(template1, template2, etc.)`.

Le chapitre consacré aux templates entre plus largement dans les détails.

6.7.4. API

6.7.4.1. Méthodes

`cache()`	$opts	Cf. chapitre Cache
`close()`		
`debug()`		
`getLocale()`		Retourne la locale associée à la réponse.
`isCached()`	$key, $ttl	Cf. chapitre Cache
`isClosed()`		
`render()`		
`reset()`		Réinitialise le contenu et les templates.
`resetHeader()`	$key	
`resetTemplates()`		
`sendAuthenticationForm()`		
`sendData()`	$input, $opts=array()	Permet de retourner un flux de données (et de forcer un téléchargement). `$input` peut être un objet de type `File`. Il peut également correspondre à une `Collection` ou un `ResultSet`. Le flux de données correspondra alors à un fichier CSV.
`sendHttpNotModified()`	$etag=null	
`sendMessage()`	$msg	

`sendRedirect()`	$url='/' $args=null	`$url` peut également être un ActiveRecord. La méthode `asURL()` sera alors mise en œuvre. La méthode fonctionne dans le cadre d'une requête AJAX.
`setBreadcrumb()`	array $h	
`setFormat()`	$format	Voir les constantes ci-dessous.
`setLocale()`	$locale	
`setMenu()`	$arg1, $arg2,	Permet d'initialiser la variable `$ctx->_menu`. `$_menu` peut être utilisé au sein d'un skeleton pour initialiser la navigation (`$_menu[0]`).
`setHead()`	$arg	L'argument peut prendre 4 types de valeur : String : initialise le « title » Hash : clefs : title, description, keywords ActiveRecord : les données seo sont utilisées. Vide : les données seo de l'action sont utilisées.
`setSpace()`	$name=null	Permet d'initialiser l'emplacement associé à l'action. La notion d'emplacement est liée au module de régie publicitaire.
`setTitle()`	$title	Initialise `$ctx->_title` qui est utilisé pour le titre de la page (par convention entre balises <h1>).
`useTemplate()`		
`useTemplates()`		
`useXml()`	$file_name=null	
`write()`	$content $bindings =null	

6.7.4.2. Constantes

`FORMAT_HTTP`	La réponse correspond à une commande HTTP (404, 200).
`FORMAT_JSON`	Le contenu du `Context` `$ctx` est converti en JSON.
`FORMAT_JSONRPC`	Le contenu correspond à du code JavaScript.
`FORMAT_PHP`	Le contenu du `$ctx` est converti tableau PHP sérialisé.
`FORMAT_SOAP`	
`FORMAT_TEXT`	Contenu HTML ou texte.
`FORMAT_XML`	`$ctx` en XML
`FORMAT_XMLRPC`	`$ctx` en XML (format *XML-RPC*)

Des informations supplémentaires sont disponibles dans le chapitre consacré aux Web Services.

6.8. DataBaseObject

6.8.1. Rôle

L'objet `$dbo` donne accès à la base de données associée au projet.

6.8.2. Bindings

Les *bindings* correspondent à un *hash* permettant le formatage rapide et sécurisé d'une requête SQL.

```
$dbo->query('SELECT * FROM page WHERE code=:code:',
            array(':code:' => "bla'bla"));
```

Différents types de *bindings* existent :

`':col:' => "bla'bla"`	'bla\'bla'
`'#col#' => "bla'bla"`	bla\'bla
`'[cols]' => array('a', "bla'bla")`	'a', 'bla\'bla'
`'{cols}' => array(1, 2)`	1, 2

```
$rs = $dbo->query('SELECT * FROM page WHERE code IN ([codes])',
                  array('[codes]' => array('faq', 'help')));
```

6.8.3. API

6.8.3.1. Méthodes

`affectedRows()`		Retourne le nombre de lignes affectées. Il est souvent préférable d'utiliser la valeur retournée par `exec()`, `insert()` ou `update()`.
`asArray()`	$sql $bindings=null $field=null	Retourne dans un tableau les différentes lignes de la requête (*hash*). Si l'argument `$field` est transmis, le tableau est aplati en utilisant ce champ. Retourne un tableau vide en cas d'erreur.
`asHash()`	$sql $bindings=null $key_field=null $value_field=null	La requête liée est mise en cache. Si `$key_field` contient la valeur `null`, les deux premières colonnes sont utilisées pour construire le *hash*. Si `$value_field` contient la valeur `null`, la méthode retourne un tableau de *hash*.
`asSet()`	$sql $bindings=null $key_field=null	
`debug()`		Affiche dans les logs la dernière requête SQL exécutée.

`exec()`	$sql, $bindings=null	Utilisée pour les requêtes `DELETE`, `UPDATE`, `INSERT`. Il préférable d'utiliser `update()` et `insert()` pour les deux dernières. Retourne le nombre de lignes modifiées (`affected_rows`).
`fetchArray()`	$sql, $bindings=null	Retourne un tableau correspondant à la première ligne de la requête. Les méthodes `fetchArray/Hash/Objet/Value()` ont l'avantage de passer en interne par des requêtes non bufferisées et sont par là même plus optimisées que leurs cousines liées à l'objet `ResultSet`. Retourne `false` en cas d'erreur.
`fetchHash()`	$sql, $bindings=null	Retourne un *hash* correspondant à la première ligne de la requête. Return `false` en cas d'erreur.
`fetchObject()`	$sql, $bindings=null	Retourne un objet correspondant à la première ligne de la requête. Retourne `false` en cas d'erreur.
`fetchValue()`	$sql, $bindings=null	Retourne la valeur de la première colonne de la première ligne de la requête. Retourne `false` en cas d'erreur.
`hasTable()`	$name $user_cache=true	
`insert()`	$table $record $options=null	Retourne l'id du nouvel élément inséré si la table contient une colonne en *auto increment*. Retourne `-1` dans le cas d'une erreur de doublon (cf. clef unique). `$record` correspondre à un *hash*. Options est un hash dont une clef possible est `modifiers` `[LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]`
`massInsert()`	$table $records $options=null	Permet de réaliser plusieurs insertions en une seule requête. $records peut correspondre à un array de tableaux associatifs, à une `Collection`, à un Csv ou à un `ResultSet`.
`matchedRows()`		Retourne le nombre de lignes impactées par la requête. Cette méthode doit être préférée à `affectedRows()` dans le cadre d'`UPDATE`. En effet `affectedRows()` retourne `0` lorsque la requête ne modifie aucune des colonnes.
`monitor()`		Permet de tracer toutes les requêtes. Un double appel annule le *monitoring*.
`query()`	$sql, $bindings=null, $options=null	Utilisée pour un SELECT. Retourne un `ResultSet` ou `false` en cas d'erreur. Les options sont les suivantes : `[HIGH_PRIORITY] [STRAIGHT_JOIN] [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT] [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]`
`replace()`	$table, $record, $options=null	Retourne le nombre de lignes modifiées. `LOW_PRIORITY | IGNORE`
`update()`	$table, $record, $where=null, $bindings=null, $options =null	Retourne le nombre de lignes modifiées. `LOW_PRIORITY | IGNORE`

6.8.3.2. Exceptions

Les méthodes `query()`, `exec()`, `insert()`, `massInsert()`, et `update()` émettent des exceptions `DataBaseException` en cas d'erreur au sein de la requête. Le code associé à l'exception correspond au code d'erreur MySQL.

http://dev.mysql.com/doc/refman/5.0/en/error-messages-server.html

6.8.4. Exemples d'utilisations

6.8.4.1. Insertion

```
$record = array('name' => 'x', 'value' => 2);
$id = $dbo->insert('ma_table', $record);
$res->write('id = '.$id);
```

6.8.4.2. Insertion multiple

```
$records = array(array('name' => 'x', 'value' => 2),
                 array('name' => 'x', 'value' => 2));
$n = $dbo->insert('ma_table', $records);
$res->write($n.' lignes ont été insérées');
```

6.8.4.3. Mise à jour

```
$n = $dbo->update('ma_table', array('value' => 1, 'updated_at' => Date::now()));

$n = $dbo->update('ma_table', array('value' => 1, '`updated_at`=NOW()'));
```

6.9. ResultSet

6.9.1. Principe

Un `ResultSet` est l'objet retourné par la méthode `query()`. Il donne accès aux lignes retournées par un `SELECT`.

Cette classe implémente l'interface `Iterator`. Il est ainsi possible de parcourir les différents enregistrements avec un `foreach`. La valeur retournée à chaque itération est un *hash*.

```
foreach ($dbo->query('select * from abuse') as $row) {
  print_r($row);
}
```

Par convention un `ResultSet` est écrit `$rs`.

```
$rs = $dbo->query('select * from abuse');
foreach ($rs as $row) {
  print_r($row);
}
```

L'utilisation d'un `foreach` a l'avantage de replacer automatiquement le pointeur interne du `ResultSet` au début. L'utilisation d'un `getNext()` impliquera l'utilisation de la méthode `goFirst()` si deux parcours successifs sont réalisés.

L'utilisation de la méthode statique `ResultSet::getInstance()` est préférable à `$dbo->query()`. Le premier paramètre correspond au nom de la table.

```
foreach (ResultSet::getInstance('abuse') as $row) {
  print_r($row);
}
```

Le second paramètre, optionnel, correspond à un hash d'option permettant de modifier la query.

Les clefs disponibles sont les suivantes :

- filters ;
- limit ;
- map ;
- order.

```
$opts = array('limit' => 4,
              'map' => array('first_name' => 'nom',
                             'last_name' => 'prenom',
                             'email'),
              'filters' => array('status' => 1),
              'order' => array('last_name', 'first_name'));

foreach (ResultSet::getInstance(T('user'), $opts) as $row) {
  print_r($row);
}
```

6.9.2. API

6.9.2.1. Méthodes

`asArray()`	$field=null	Cf. `DataBaseObject`
`asCollection()`	$model	Lorsqu'un ResultSet contient toutes les colonnes d'une table d'un ActiveRecord, cette méthode peut être utilisée pour le transformer en Collection.
`asHash()`	$key_field=null $value_field=null	Cf. `DataBaseObject`
`asSet()`	$field=null	L'aplatissage est réalisé par défaut sur la première colonne. Le paramètre optionnel `$field` permet de spécifier une autre colonne.
`fetchArray()`		Retourne la ligne courante en tant que tableau. Cette fonction fait automatiquement avancer le pointeur interne d'un enregistrement.
`fetchFieldName()`		
`fetchHash()`		Retourne la ligne courante en tant que *hash*. Cette fonction fait automatiquement avancer le pointeur interne d'un enregistrement.
`fetchObject()`		Retourne la ligne courante en tant qu'objet. Cette fonction fait automatiquement avancer le pointeur interne d'un enregistrement.
`fetchValue()`		Retourne la valeur de la première colonne de la ligne courante. Cette fonction fait automatiquement avancer le pointeur interne d'un enregistrement.
`foundRows()`		Retourne le nombre de lignes trouvées s'il n'y avait pas de `LIMIT` dans la requête. Le `SELECT` doit comporter l'option `SQL_CALC_FOUND_ROWS`.
`getFields()`		Retourne un tableau des noms des colonnes.
`getNext()`	$type	Retourne la ligne courante. Le type peut être `MYSQL_NUM` ou `MYSQL_ASSOC`. Cette fonction fait automatiquement avancer le pointeur interne d'un enregistrement.
`goFirst()`		Déplace le pointeur interne au niveau du premier enregistrement.
`goLast()`		Déplace le pointeur interne au niveau du dernier enregistrement.
`isEmpty()`		Indique si le ResultSet contient des enregistrements.
`numFields()`		Retourne le nombre de colonnes.
`map()`	$h	Permet de définir un *mapping* de colonnes. Retourne `$this`.
`numRows()`		Retourne le nombre de lignes.
`result()`	$row $col	Retourne une valeur localisée par son numéro de ligne et de colonne.

`unmap()`	Supprime le *mapping*. Retourne `$this`.

6.9.3. Exemples

6.9.3.1. Téléchargement au format CSV

```
$rs = $dbo->query('select * from blog')->map(array('id' => 'blog_id', 'name'));
$res->sendData($rs);
```

6.10. Cookie

6.10.1. Fonction

Cette classe donne accès aux cookies. KWO fonctionne cependant avec un seul et même cookie contenant un *hash* de valeurs. Ce cookie est appelé `tracker`.

La méthode `getCookie()` (sans paramètre) de la classe `Request $req` retourne une instance de ce cookie. Cette méthode seule doit être utilisée pour créer ou accéder à un cookie.

Les attributs de l'objet cookie correspondent aux valeurs du *hash*.

```
$cookie = $req->getCookie();
$cookie->setAttribute('attr1', array(1, 2));
$cookie->setAttribute('attr2', "test");
$res->write($cookie->attr2);
```

Les cookies sont par défaut partagés par tous les sous-domaines.

Ex. `www.projet.com`, `blog.projet.com`.

6.10.2. API

6.10.2.1. Attributs

Valeurs du *hash*.

6.10.2.2. Méthodes

`debug()`		Affiche le contenu du cookie dans les logs.
`getAttribute()`		
`getAttributes()`		
`getName()`		Retourne le nom du cookie.
`hasAttribute()`		
`hasAttributes()`		
`remove()`		
`removeAttribute()`		
`setAttribute()`	$key	
`setAttributes()`		
`setPath()`		
`setValue()`		

6.11. Session

6.11.1. Fonction

Seule la méthode `getSession()` de `$req` doit être utilisée pour accéder à une session.

Les attributs de l'objet correspondent aux données de la session.

```
$session = $req->getSession();
echo $session->user_id;
```

L'accès direct à un attribut est **interdit** en écriture; la méthode `setAttribute()` doit systématiquement être utilisée.

```
$session = $req->getSession();
$session->setAttribute('number', 1);
```

6.11.2. API

6.11.2.1. Attributs

Les données de la session.

6.11.2.2. Méthodes

`debug()`		Affiche le contenu de la session dans les logs.
`destroy()`		Pour détruire une session.
`getAttribute()`		
`getAttributes()`		
`getId()`		
`hasAttribute()`		
`hasAttributes()`		
`increment()`	$key	
`invalidate()`		Détruit la session.
`removeAttribute()`		
`setAttributes()`		
`setAttribute()`		

7. ACTIVERECORD

7.1. Principe général

Tout modèle de KWO hérite de la classe `ActiveRecord`. Ce *pattern* permet de faire l'association entre les attributs d'un objet et les colonnes d'une table.

Le principe est ici de factoriser toutes les méthodes qui sont généralement partagées par un modèle : création, mise à jour, destruction, manipulation des attributs, des métadonnées, de l'indexation, des traductions, etc.

Les modèles sont par conséquent vides la plupart du temps.

```
class Machin extends ActiveRecord {
}
```

Les méthodes à développer sont :

- spécifiques au métier ;
- des surcharges ;
- des hooks.

Les modèles contiennent également souvent des constantes (ex. `STATUS_*`, `TYPE_*`, `FLAG_*`) et des tableaux statiques[5].

```
class Order extends ActiveRecord
{
  const FLAG_DIGITAL = 1;
  const FLAG_EXPORTED = 2;
  const FLAG_DISCOUNT = 4;
  const FLAG_ITEM_RETURNED = 8;
  const FLAG_WRAPPING = 16;

  public static $flags = array(self::FLAG_DIGITAL => 'digitale',
                               self::FLAG_EXPORTED => 'exportée',
                               self::FLAG_DISCOUNT => 'réduction',
                               self::FLAG_ITEM_RETURNED => 'retour demandé',
                               self::FLAG_WRAPPING => 'papier cadeau');
```

7.2. Définition

Quatre étapes sont nécessaires pour la création d'un ActiveRecord.

7.2.1. Création de la table

Cette table doit absolument disposer d'une colonne `id`. Voir le chapitre sur la norme pour les autres colonnes.

Les colonnes ne doivent pas être redondantes par rapport au nom du modèle. Le titre d'un forum est `title` plutôt que `forum_title`.

7.2.2. Création des fichiers classe et paramètres

[5] Les tableaux statiques publics $statuses et $flags sont souvent utilisés pour construire les interfaces du backoffice.

```
php init.php scaffold extension=<EXT> model=<MODEL>
```

7.2.3. Modification des paramètres

Les paramètres d'un modèle sont situés dans le fichier `etc/ext/<EXT>/models/<MODEL>.conf.inc`.

Liste des paramètres :

`belongs_to`		Relation d'appartenance. La table doit contenir la `foreign_key` du modèle lié. La table associée au modèle Topic doit contenir `forum_id`.
`has_many`		Relation de possession. La table du modèle lié doit contenir la `foreign_key`.
`has_one`		Relation de possession. La table du modèle lié doit contenir la `foreign_key`.
`id`	obligatoire	Valeur unique supérieure à `20000`.
`label`		Intitulé français pouvant qualifier le modèle
`name`	obligatoire	Nom du modèle.
`metas`	booléen	Indique si le modèle dispose de métadonnées.

Le fichier de configuration du modèle *Topic* situé `etc/ext/cms/models/topic.conf.inc` est proposé ci-dessous à titre d'exemple.

```
$parameter['name'] = 'topic';
$parameter['id'] = 71;
$parameter['belongs_to'] = array('forum');
```

Il convient de regrouper les `id` par extension (ex. 200, 201, 202 pour une extension et 500, 501 pour une autre extension).

Les extensions spécifiques à un projet qui ne feront jamais partie de la distribution standard KWO doivent avoir des `id` supérieurs à 10000.

Les tables sont préfixées avec la valeur du paramètre `$parameter['table_prefix']` de l'extension.

Ex. `boutique_produit`, `boutique_panier` pour l'extension Boutique.

Tous les noms de modèles doivent être différents. Un rôle essentiel de l'architecte est de choisir des noms de modèles précis qui ne risquent pas d'entrer en collision avec d'autres modèles d'autres extensions.

7.2.4. Reconstruction de la configuration

```
php init.php build
```

7.3. Instanciation d'un ActiveRecord

7.3.1. getInstance()

Cette méthode statique prend deux paramètres.

Arguments	int[6] ou hash ou true optionnel	
Options	optionnel	Voir ci-dessous.

Création d'un Post qui ne sera véritablement créé qu'après l'utilisation de `commit()`. Avant de le `commit()`, `$post` correspond à un ClassRecord, un modèle *Post* non initialisé.

```
$post = Post::getIntance();
$post->commit(array('title' => 'bonjour',
                    'content' => 'monde'));
$res->write('id du nouveau post : '.$post->getId());
```

Chargement d'un sujet ayant pour id la valeur 1.

```
$topic = Topic::getInstance(1);
```

Chargement d'une page ayant pour code la valeur `faq`.

```
$page = Page::getInstance(array('code' => 'faq'));
```

Afin de pouvoir écrire une action de création/mise à jour le plus naturellement possible, la création d'un ClassRecord peut également être réalisée en passant la valeur `true` en deuxième argument.

```
try {
  $arg = $req->hasAttribute('id', true) ? (int) $req->id : true ;
  $post = Post::getInstance($arg);
  $post->commit($req->record);
}
catch (Exception $e) { $err->add($e); }
```

7.3.2. Terminologie

Un ActiveRecord instancié et existant est qualifié de : item.

7.3.3. Options d'instanciation

Il s'agit d'un *hash* pouvant contenir les clefs suivantes :

`ignore`	booléen	Un échec de chargement de l'objet ne sera pas reporté dans les logs.

[6] Lorsque cela est opportun, forcer un argument en numérique (via un cast de type `intval()` ou `(int)`) est une optimisation.

include	array	Liste de modèles liés (belongs_to) qui seront chargés dans la même requête.
metas	array	Liste de métadonnées. Elles sont accessibles via des clefs de la forme comment, display, etc.

7.4. API

7.4.1. Attributs

Ils correspondent aux colonnes de la table. Si l'option include est passée les informations du modèle lié sont également disponibles, préfixées par le nom du modèle (ex. user_first_name).

7.4.2. Méthodes

asDocumentHash()		Se référer au chapitre consacré aux moteurs de recherche pour plus d'informations.
asFeedHash()		Permet de redéfinir les éléments qui vont composer un Feed. Le *hash* retourné doit contenir les éléments suivants : id, title, content, link, updated_at, created_at. Voir le chapitre consacré aux Flux.
asFinderHash	&$h	Méthode à surcharger pour modifier le *hash* utilisé dans le finder. La clef virtuelle _color permet de modifier la couleur du bouton à gauche (Ex. green, red, blue, yellow, orange). Par défaut la cellule contient green, sauf si la table contient une colonne status dont valeur est inférieure à 0 (elle contient alors red).
asHeadHash()		
asSearchHash()		Se référer au chapitre consacré aux moteurs de recherche pour plus d'informations.
asUrl()		Retourne l'URL permettant de visualiser l'objet en Front. Par défaut : /extenstion/<MODEL>.view/-/id/val. Il peut être intéressant de la surcharger pour des problématiques de référencement.
belongsTo()	$item	
commit()	$h	
debug()		Affiche les attributs de l'objet dans les logs.
decrement()	$key	Décrémente la valeur d'un attribut.
decrementMeta()	$key	Décrémente la valeur d'une métadonnée.
destroy()		
getAttribute()	$key	
getAttributes()	$prefix=null $trim_flag=false	

`getComments()`	$opts=array()	Retourne une `Collection` de tous les commentaires liés à l'objet.
`getDescription()`		
`getId()`		
`getKey()`	$key	Retourne une clef liée à un objet. La clef est de la forme `model_id-record_id-secret_hash`. Il est conseillé de transmettre cette clef dans les insertions Front. L'`ActiveRecord` peut ensuite chargé de la manière suivante : `Item::getInstanceByKey($req->item_key)`.
`getMask()`	$column='flags' $decimal=false	Retourne un set contenant les bits à 1 de la colonne sélectionnée.
`getMeta()`		
`getName()`		
`getTitle()`		
`getNext()`		
`getPrev()`		
`getProperties()`		
`getProperty()`	$key	
`getTags()`		Retourne un tableau de tags.
`getModelId()`		Retourne le `model_id`. Possibilité également d'utiliser `I($item)`.
`getUser()`		Retourne l'utilisateur qui a créé l'objet. La table doit contenir la colonne `user_id` ou `visit_id`.
`hasAttribute()`	$key	
`hasAttributes()`		
`hasNext()`		
`hasPrev()`		
`increment()`		Incrémente la valeur d'un attribut.
`incrementMeta()`	$key	Incrémente la valeur d'une métadonnée.
`index()`		Voir « moteur de recherche ».
`isA()`	$model_name	
`isItem()`		Contrôle que l'élément est bien « chargé ».
`isNew()`		Indique si l'objet vient d'être créé. (A partir du deuxième appel à `commit()` l'item n'est plus considéré comme nouveau).
`getModel()`		Retourne le nom du modèle. Possibilité également d'utiliser `N($item)`.
`setAttribute()`	$key, $value	Ne déclenche aucun *hook* (à la différence de `commit()`).
`setAttributes()`	$h	Ne déclenche aucun *hook* (à la différence de `commit()`).

`setMeta()`	$type, $value	
`setMetas()`	$h	
`setProperties()`	$properties, $locale=null	
`setProperty()`	$key, $value, $locale=null	
`setTags()`	$set	
`setUser()`	$user	Associe le `$user` en tant que créateur de l'item.
`setVisit()`	$visit	

7.4.3. Méthodes et variables statiques

`collect()`	$opts	Voir Collection
`exists()`	$args	Indique si un objet existe. Retourne l'`id` si l'objet existe ou `false` dans le cas contraire. Ex. `Topic::exists(2)` ou `Page::exists(array('code' => 'cgv'))`
`getInstance()`	$args, $opts	Voir ci-dessus
`id()`	$args	Retourne, s'il existe, l'id de l'élément correspondant aux arguments. Ex. `$id = Page::id(array('name' => 'cgv'))`

7.5. Création de nouvelles méthodes

A la différence des actions, seul l'objet `$this->dbo` est directement accessible au sein des méthodes de l'ActiveRecord. L'accès à `Request`, `Response`, `Logger`, `Error` est réalisé avec `S('req')`, `S('res')`, `S('log')`, `S('err')`.

Il est vivement déconseillé d'accéder aux objets `Request` et `Response` au sein d'un `ActiveRecord`. Y avoir accès révèle souvent une erreur de conception.

Le rôle d'une action est de récupérer les informations de la requête (`$req`), de les passer à une méthode de l'ActiveRecord qui retourne alors une valeur qui pourra être utilisée au sein de la réponse (`$res`).

7.6. Navigation

Les méthodes `hasPrev()`, `hasNext()`, `getPrev()`, `getNext()` permettent de naviguer d'un ActiveRecord à l'autre. Elles prennent en compte, s'il y a lieu, les colonnes `position`, `status` ainsi que la première des relations de type `belongs_to`.

7.7. Gestion du statut

La colonne `status` particulière pour deux raisons :

1. sa valeur est codifiée ;

2. les hooks `onBeforeStatusChange(&$attrs)` et `onAfterStatusChange($old)` sont déclenchés en cas de modification de sa valeur.

La valeur de l'attribut `status` doit respecter la règle suivante :

- valeur supérieure à 0 : l'élément est visible en ligne ;
- valeur inférieure ou égale à 0 : l'élément ne doit pas apparaître en ligne.

La méthode `onAfterStatusChange()` reçoit en paramètre l'ancienne valeur de la colonne (ou `null` dans le cas d'une création d'objet). Elle se charge par défaut d'indexer l'objet (ou de le supprimer de l'index). Le paramètre `search.models` est pris en compte.

7.8. Relations entre modèles

L'architecture logicielle consiste à concevoir des modèles et à définir les relations entre eux. Ces relations sont de plusieurs types : appartenance, possession. Le paramétrage des `ActiveRecord` permet de les définir.

7.8.1. Paramètres

Les liaisons entre modèles sont définies avec les paramètres : `belongs_to`, `has_many`, `has_one`.

En prenant l'exemple de l'extension Forum, le fichier `forum.conf.inc` contient

```
$parameter['has_many'] = array('topic');
```

et `topic.conf.inc`

```
$parameter['belongs_to'] = array('forum', 'user');
```

7.8.2. Méthodes magiques

En utilisant les liaisons entre modèles (`has_many`, `has_one`, `belongs_to`), certaines méthodes sont automatiquement créées.

7.8.2.1. add[Model]()

Dans le cas d'un `has_many`, ajoute un élément. Prend en argument un *hash*. La *foreign_key* est automatiquement ajoutée au *hash*.

```
$forum = Forum::getInstance(2);
$topic = $forum->addTopic(array('title' => $req->title,
                                'content' => $req->content,
                                'user_id' => $uid));
```

Dans le cas d'une relation via un through, l'argument peut également être un ActveRecord.

```
$notice = Notice::getInstance($notice_id);
$user->addNotice($notice);
```

7.8.2.2. count[Models]($opts)

7.8.2.3. get[Model]()

Retourne un objet lié par `belongs_to` ou `has_one`. Aucune requête supplémentaire au SGBD n'est réalisée si un `include` a été précisé.

```
$topic = Topic::getInstance(11);
$forum = $topic->getForum();
```

7.8.2.4. get[Models]()

Retourne une collection dans le cadre d'un `has_many`. L'argument passé correspond un `$opts` présenté dans le chapitre Collection.

```
$forum = Forum::getInstance(2);
foreach ($forum->getTopics() as $topic) {
  $res->writeln($topic->getId().' - '.$topic->title);
}
```

7.8.2.5. hasAny[Model]()

7.8.2.6. remove[Model]()

7.9. Exceptions

7.9.1. ActiveRecordException

Exception générale dans le cadre d'un ActiveRecord.

7.9.2. ClassRecordException

Exception générale quand l'objet n'est pas encore chargé.

7.9.3. UnknownRecordException

Exception émise au moment du chargement d'un ActiveRecord.

7.9.4. DuplicateRecordException

Exception émise lors de la création d'un objet qui entre en conflit avec une contrainte d'unicité d'une table.

```
try {
  $contact = $base->addContact(array('email' => $req->email));
  $req->track('newsletter signup ('.$base->code.')', 'push');
  Conversion::add($contact);
}
catch (DuplicateRecordException $e) {
  $req->track('newsletter duplicate signup ('.$base->code.')',
              'warnings');
}
```

7.10. Métadonnées

7.10.1. Principe général

Une `MetaData` correspond à une donnée qu'il est possible d'attacher à tout ActiveRecord. L'utilisation des métadonnées vise à éviter la multiplication de colonnes identiques dans toutes les tables de KWO.

Ces métadonnées sont stockées dans la table `core_item_meta` (`Meta::TABLE`).

Une métadonnée doit ou pouvoir être reconstruite à partir d'autres tables ou être non essentielle au bon fonctionnement de l'applicatif.

7.10.2. Types

`abuses`	Nombre de reports d'abus
`admin`	ID de l'admin qui a créé l'item
`bits`	Masque de bits partagés par tous les modèles
`comment`	Nombre de commentaires
`conversions`	
`created_at`	
`dislikes`	Nombre de « je n'aime pas »
`displays`	Nombre d'affichages uniques.
`downloads`	Nombre de téléchargements.
`favorites`	Nombre de mises en favoris
`flags`	
`intents`	
`interests`	
`latitude`	
`likes`	Nombre de « j'aime »
`locales`	
`longitude`	
`parent`	
`print`	

`rate`	Nombre de notes
`revisions`	Nombre de « commits »
`shares`	Nombre de partages
`state`	Etat (voir la gestion du cache)
`status`	Identique à l'attribut `status` de l'item (s'il existe).
`subscriptions`	Nombre d'abonnements.
`updated_at`	
`user`	User_id de l'utilisateur qui a créé l'item (cf. `setUser()`)
`valuations`	Note moyenne (sur 5)
`visit`	Visit_id correspondant à la visite à l'origine de la création de l'item (cf. `setVisit()`)

7.10.3. Utilisation

7.10.3.1. ActiveRecord

```
$opts = array('metas' => array('display', 'created_at'));
$topic = Topic::getInstance($id, $opts);
```

L'objet dispose alors de colonnes supplémentaires de la forme `display`, `created_at`, etc.

La méthode `getMeta()` peut également être utilisée. Elle nécessitera, à l'inverse de l'option `metas`, une nouvelle requête à la base.

7.10.3.2. Collection

```
$opts = array('last' => 20,
              'include' => 'user',
              'metas' => array('display', 'created_at'));
$topics = $forum->getTopics($opts);
```

7.10.4. Modification

Les méthodes `increment()`, `decrement()`, `setMeta()` peuvent être utilisées.

7.11. Limitations

Un nom de modèle ne peut

- finir par la lettre *s*. Le nom *Business* n'est donc pas valide ;
- commencer par *any*.

Le nom d'un modèle est en minuscule sauf la première lettre qui est en majuscule. Le nom *NewModel* n'est donc pas valide.

Le pluriel doit respecter certaines règles :

- `s` : Product → Products ;

- `x` : Morceau → Morceaux ;
- `ies` : Party → Parties ;
- `shes` : Wish → Wishes.

Les noms de modèles *Teeth* (*Tooth*), *Child* (*Children*) ne sont donc pas envisageables.

7.12. Remarques diverses

Un ActiveRecord ne doit jamais être créé avec l'opérateur `new`.

7.13. Destruction d'objet

KWO est conçu pour assurer de la façon la plus automatisée possible la cohérence des données. La classe ActiveRecord se charge par exemple de supprimer tous les éléments liés à un objet que l'on souhaite détruire.

Lorsqu'un ActiveRecord est détruit, le système nettoie automatiquement :

- le moteur de recherche ;
- les métadonnées ;
- les tags ;
- le cache ;
- les éventuelles tables de jointure.

Il est possible de préciser que la suppression d'un type d'objet entraîne la suppression d'objets liés en précisant la relation de dépendance au niveau du paramétrage de l'ActiveRecord.

```
$parameter['id'] = 650;
$parameter['name'] = 'forum';
$parameter['label'] = 'forum';
$parameter['table'] = 'forum';
$parameter['belongs_to'] = array('column');
$parameter['has_many'] = array('topic' => array('dependent' => true));
```

7.14. Classes Model et Item

7.14.1. Classe Model

Cette classe contient des méthodes statiques permettant d'obtenir des informations sur un modèle.

`all()`	$model	Retourne un hash de tous les modèles de l'application. La clef correspond à l'id du modèle et la valeur, à son nom.
`exists()`	$model	Indique si le modèle existe.
`hasProperties()`	$model	
`hasField()`	$model, $key	Le modèle dispose t'il de la colonne `$key` dans sa table associée ?
`id()`	$model	ID du modèle (`$item->getModelId()`). Voir aussi `I()`.
`label()`	$model	

`name()`	$model	Voir aussi `N()`.
`parameter()`	$model, $name	
`parameters()`	$model	
`properties()`	$model	
`property()`	$model, $name	
`table()`	$model	Nom de la table associée. Voir aussi `T()`.

Le paramètre `$model` peut aussi correspondre au nom du modèle ou à son id.

```
echo Model::table('user'); // affiche social_user
```

7.14.2. Classe Item

La classe Item permet de créer un ActiveRecord ou une Collection en précisant le nom (ou l'id) du modèle en paramètre.

```
$forum = Item::getInstance('forum', 1);
```

```
$forums = Item::collect('forum');
```

`getInstanceByKey()`	$key, $opts	Cf. `ActiveRecord getKey()`
`getInstance()`	$model, $args, $opts	Indique si le modèle existe.
`collect()`	$model, $opts	
`exists()`	$model, $args	Permet de vérifier si un item existe. `$args` peut correspondre à un int (id), une string (code), un hash d'attribut/valeur.
`id()`	$model, $args	Retourne, s'il existe, l'item.

```
if (Item::exists($model, 2)) {
  $item = Item::getInstance($model, 2);
}
```

Les 2 exemples ci-dessous sont équivalents :

```
$id = Item::id('user', array('email' => 'moi@domaine.com'));
```

```
$id = User::id(array('email' => 'moi@domaine.com'));
```

8. Collection

8.1. Rôle

Une collection correspond à une liste d'items.

8.1.1. Création

La première méthode permettant de créer un objet `Collection` consiste à passer par la méthode statique `collect()` de `ActiveRecord`.

```
$opts = array('filters' => array('forum_id' => 2),
              'last' => 10);
$topics = Topic::collect($opts);
```

La seconde repose sur les méthodes magiques `getXxxs()` de `ActiveRecord`.

```
$forum = Forum::getInstance(2);
$topics = $forum->getTopics(array('last' => 10));
```

8.1.2. Parcours

Une `Collection` peut être parcourue avec un `foreach`. Chaque itération retourne alors un ActiveRecord.

```
foreach ($topics as $topic) {
  $res->write($topic->title.' ('.$topic->getId().') <br/>');
}
```

Le *hash* (optionnel) passé à `collect()` et `getXxxs()` est une structure normée permettant de restreindre les éléments à inclure dans la `Collection`. Par convention il est appelé `$opts`.

8.1.3. Options : $opts

Les clefs du *hash* `$opts` sont limitées aux éléments suivants :

`cache`	booléen	Indique si la requête doit être mise en cache. Valeur par défaut : `true`.
`filters`	hash	*Hash* dont les clefs correspondent à des noms de colonnes et les valeurs, aux valeurs que doivent prendre ces colonnes. Voir le point ci-dessous pour plus de détails.
`having`		
`include`	array	Permet d'inclure les colonnes des objets liés (`belongs_to`, `has_one`). Dans le cadre d'un `has_many`, une colonne supplémentaire sera disponible contenant les ids des objets liés : `model_set`. Cette colonne n'est disponible qu'en écriture.
`last`	entier	Récupère les `n` derniers éléments. Si `order` n'est pas précisé le tri se fait sur la colonne `id`.
`limit`	entier	Nombre d'éléments (maximal) dans la collection.

`map`	array	Précise un *mapping* de colonnes. Le format des résultats passe alors automatiquement en `FORMAT_HASH`. Seules les colonnes précisées dans le *mapping* sont incluses. Ex. `array('name', 'table.title' => 'label')`
`metas`	array	Tableau de MetaDatas.
`offset`	entier	Le décalage ne se fait pas sur la valeur de l'*offset* mais sur cette valeur multipliée par la valeur de *limit*.
`order`	hash	Précise le nom de la colonne de tri. Ex. `array('name', 'price' => 'DESC')`.
`random`	booléen	Classement aléatoire.
`reverse`	booléen	Inverse le tri. Si la clef `order` n'est pas précisée le classement se fait sur la clef primaire `id`.
`title`		

8.1.4. Les filtres : filters

8.1.4.1. Types

Les clefs peuvent également être préfixées avec des critères `|critère|nom_colonne` :

`=`	= 'value' La valeur peut correspondre à un tableau (l'opération correspond alors à un IN).
`IN`	IN (value1, value2) La valeur doit correspondre à un tableau
`%`	LIKE 'value%'
`%%`	LIKE '%value%'
`>` `>=`	> value
`<` `<=`	< value
`TAG`	élément a le tag
`MANAGER`	élément est managé par user_id
`Y`	année = 'value' (la colonne est un timestamp)
`M`	mois = 'value' (le colonne est un timestamp)
`><`	BETWEEN 'value1' AND 'value2' (la valeur passée est un tableau de 2 éléments)
`REGEXP`	REGEXP 'value'
`EMPTY`	vérifie si la colonne est vide

8.1.4.2. Négation

Chaque critère peut être préfixé par le caractère `!` pour indiquer une négation de la condition.

```
$filters['status'] = 1;
if ($req->y > 2000) {
  $m = $req->hasAttribute('m', true) ? $req->m : date('n');
  $filters['|Y|created_at'] = $req->y;
  $filters['|M|created_at'] = $m;
  $filters['|!%|name'] = 'test';
}
```

Il est également possible de passer un *hash* contenant deux clefs spéciales : `sql` (un bout de requête SQL correspondant au `WHERE`) et `bindings` (cf. chapitre `DataBaseObject`).

8.1.4.3. Combinatoire

Les critères sont par défaut liés par un ET logique. Il est cependant possible de combiner des groupes de critères avec une syntaxe spécifique.

```
$cousins = array('BTM-SC-04','BTM-SC-02');
$opts = array('filters' => array(array('code' => $cousins,
                                       'status' => 1),
                                 array('OR' => array('range' => $cousins,
                                                     'range_is_default' => 1,
                                                     'status' => 1))));
$products = Product::collect($opts);
```

La requête SQL associée est la suivante :

```
SELECT shop_product.* FROM shop_product WHERE (shop_product.code IN ('BTM-SC-04','BTM-SC-02') AND
shop_product.status='1') OR (shop_product.range IN ('BTM-SC-04','BTM-SC-02') AND
shop_product.range_is_default='1' AND shop_product.status='1')
```

Toute combinaison est envisageable.

8.2. API

8.2.1. Méthodes

`asArray()`	$field=null	Retourne un tableau de *hash* correspondant à chaque ActiveRecord de la Collection. Si `$field` est précisé, le tableau retourné est un tableau de valeurs.
`asCloud()`	$num=20	
`asFeed()`		Retourne un objet `Feed` (cf. chapitre Flux).
`asHash()`	$key_field='id' $value_field=null	
`asSet()`	$field='id'	
`asSql()`		
`average()`	$field	
`count()`		Alias de `numRows()`.

`debug()`		
`first()`		Retourne le premier élément.
`foundRows()`		Nombre d'ActiveRecord en passant outre `LIMIT`.
`getFields()`		Retourne un tableau de noms de colonnes.
`getItemAt()`	$index=0	
`getItemId()`	$id	
`getPageNum()`		Retourne le numéro de la page actuelle.
`getPageCount()`		Retourne le nombre de pages.
`getPages()`		Retourne un tableau d'entier correspondant aux pages nécessaires pour parcourir toute la collection.
`getPagination()`		Retourne un objet `Pagination`.
`hasNext()`		Indique s'il y a lieu de gérer suivant.
`hasPagination()`		Indique s'il y a lieu de gérer de la pagination.
`hasPrev()`		Indique s'il y a lieu de gérer précédent.
`isMapped()`		
`last()`		Retourne le dernier élément.
`map()`	$map	Permet de préciser un *mapping* de colonnes.
`maximum()`	$field	
`minimum()`	$field	
`numRows()`		Nombre d'ActiveRecord.
`offsetFrom()`		Index du premier élément en prenant en compte l'offset.
`offsetTo()`		Index du dernier élément en prenant en compte l'offset.
`setFormat()`		Voir les constantes ci-dessous.
`setTitle()`		Utile pour `asFeed()` et pour la conversion en string (`__toString()`).
`sum()`	$field	
`unmap()`		

8.3. Cas concrets

8.3.1. Chargement de MetaDatas

```
$opts = array('last' => 20,
              'offset' => $req->offset,
              'metas' => array('comment', 'display'));
$topics = $forum->getTopics($opts);
```

8.3.2. Utilisation de colonnes spéciales dans le filtre

Le code suivant permet de sélectionner les sujets d'un forum qui :

- comportent un unique commentaire (cf. `metas`),
- ont été rédigés par un auteur vivant en France (cf. `include`).

```
$opts = array('filters' => array('comment' => 1,
                                 'user_country_id' => 250)),
              'offset' => $req->offset,
              'include' => 'user',
              'metas' => array('comment'));
$topics = $forum->getTopics($opts);
```

8.3.3. Insertion en base de données

Un objet `Collection` peut être passé en argument à la méthode `insert()` de `$dbo`.

```
$dbo->insert('backup', Blog::collect(array('fields' => array('name'))));
```

9. Composants

9.1. Pagination

9.1.1. Principe

L'idée est ici de déporter la gestion souvent fastidieuse de la pagination au sein d'une classe spécifique.

Un objet pagination est généralement obtenu à partir de la méthode `getPagination()` d'une `Collection`.

Le comportement peut être modifié en faisant appel à la méthode `asHtml()` qui prend en paramètre un hash d'options :

`class`	Classe des liens.
`first`	Caractère utilisé pour se rendre au premier élément (par défaut : «).
`href`	Redéfinit le `href` (peut contenir `((offset))`).
`last`	Caractère utilisé pour se rendre au dernier élément (par défaut : »).
`next`	Caractère utilisé pour se rendre à l'élément suivant (par défaut : ›).
`onclick`	Redéfinit le `onclick` (peut contenir `((offset))`).
`prev`	Caractère utilisé pour se rendre à l'élément précédent (par défaut : ‹).
`window_size`	Nombre de pages à afficher.

9.1.2. Exemples

```
<div style="text-align:center; clear:both;">
<?=$products->getPagination()?>
</div>
```

```
<?php if ($finder->hasPagination()): ?>
<div class="pagination">
<?=$finder->getPagination()->asHtml(array('onclick' => '$(\''.$finder->id.'_offset\').value=((offset));
Kwo.Finder.search(this, {container: \''.$finder->id.'\'});'))?>
</div>
<?php endif; ?>
```

9.2. Envoi de mail

L'envoi de mail repose sur la classe `Mail`. Cette classe permet notamment d'envoyer des mails HTML, des pièces jointes.

9.2.1. Paramétrage de l'application

Plusieurs paramètres liés à l'envoi de mail peuvent être définis au niveau du fichier de configuration de l'application.

```
$parameter['mail.from.name'] = 'Cool Site';
$parameter['mail.from.email'] = 'webmaster@site.com';
$parameter['mail.bounce'] = 'error@site.com ';
```

9.2.2. API

9.2.2.1. Méthodes

`addAttachment()`	$resource, $data=null	La contenu du message peut être passé dans la variable `$data`. La ressource est alors identifiée ainsi : `mem://fichier.extension`. Emet une `IOException` en cas d'erreur.
`addBcc()`	$email	
`addCc()`	$email, $name=null	
`addHeader()`	$key, $value	
`addRecipient()`	$email, $name=null	
`reset()`		
`send()`	$server	Possibilité de passer par un serveur extérieur en précisant l'adresse d'un serveur SMTP. Retourne le nombre de destinataires.
`setBody()`	$content, $template=true	Si le paramètre `$template` contient `true` le paramètre `mail.template` de l'application est utilisé. Il est également possible de passer le chemin d'un autre template.
`setBodyHtml()`	$content	
`setBodyText()`	$content	
`setBounce()`	$email	
`setFrom()`	$name, $email=null	
`setLang()`	$lang	
`setReplyTo()`	$email	
`setSubject()`	$subject	
`setUrl()`	$url	L'URL est utilisée lorsque le client mail n'est pas en mesure de lire l'HTML.

9.2.3. Exemples d'utilisation

```
$mail = new Mail();
$mail->addRecipient('bob@kernix.com', 'Robert Dupont');
$mail->setSubject('test mail');
$mail->setBodyHtml("<html><body>trop <strong>cool</strong></body></html>");
$mail->send('smtp.server.com');
```

9.3. CsvReader

L'objet `Csv` permet de représenter un fichier CSV.

Un fichier CSV doit toujours avoir une première ligne d'en tête contenant le nom des colonnes.

Le format de référence est :

- champs entourés par `"`
- champs séparés par `;`
- lignes séparées par `\r\n`
- retours à la ligne au sein d'un champ représentés par `\n`
- caractère `"` au sein d'un champ remplacé par `""`

```
"colonne1";"colonne2"
"a";"b""b"
```

9.3.1. Utilisation

9.3.1.1. Parcours

Un objet `Csv` peut être parcouru à l'aide d'un `foreach`.

```
foreach (Csv::getInstance($req->file_path) as $h) {
  $res->write($h['email']);
}
```

9.3.1.2. Téléchargement

La méthode `sendData()` de `$res` est en mesure de recevoir en argument une `Collection`. Elle retourne alors un flux de données au format Csv.

```
$contacts = Base::getInstance('Base', $req->id)->getContacts();
$res->sendData($contacts);
```

9.3.1.3. Insertion en base de données

Un objet `Csv` peut être passé en argument de la méthode `insert()` de `$dbo`.

```
$csv = Csv::getInstance('contacts.csv');
$dbo->insert('test', $csv->map(array('email' => 'name')));
```

9.3.2. API

9.3.2.1. Méthodes

`map`	$h	Définit un *mapping* de colonnes. Retourne `$this`.
`unmap`		Supprime le *mapping*. Retourne `$this`.

9.4. CsvWriter

9.4.1. Utilisation

9.4.1.1. Téléchargement

La méthode `sendData()` de `$res` est en mesure de recevoir en argument une `Collection` (ou un hash). Elle retourne alors un flux de données au format Csv.

```
$contacts = Base::getInstance((int) $req->id)->getContacts();
$res->sendData($contacts);
```

La méthode `sendData()` prend en second argument un hash d'options. Il est ainsi possible de préciser le nom du fichier csv avec la clef `filename`.

```
$contacts = Base::getInstance((int) $req->id)->getContacts();
$res->sendData($contacts, array('filename' => 'mescontacts.csv'));
```

9.4.1. API

9.4.1.1. Méthodes

`append`	$h	Définit un *mapping* de colonnes. Retourne `$this`.

9.5. Flux

La méthode recommandée pour créer un flux consiste à utiliser la méthode `asFeed()` d'une `Collection`.

```
try {

  $res->resetTemplates();
  $res->setContentType('rss');

  if ($res->isCached()) return ;

  $forum = Forum::getInstance((int) $req->id);

  $filters = array('forum_id' => $forum->getId(),
                   'status' => Topic::STATUS_OK);

  $topics = $forum->getTopics(array('filters' => $filters,
                                    'last' => 10,
                                    'metas' => array('comment', 'display')));
  $res->write($topics->asFeed());

}
catch (Exception $e) { $err->add($e); }
```

Chaque élément du flux est construit à partir de la méthode `asFeeditem()` de ActiveRecord. Cette méthode peut être surchargée pour adapter le contenu. Le *hash* retourné par `asFeedItem()` doit contenir les éléments suivants : `id`, `title`, `content`, `link`, `updated_at`, `created_at`.

La méthode `setTitle()` de Collection peut être utilisée pour forcer le titre du flux.

10. Echanges HTTP

10.1. HttpClient

10.1.1. API

10.1.1.1. Méthodes statiques

`json()`	$url, $params, $opts
`get()`	$url, $params, $opts
`post()`	$url, $params, $opts
`call()`	$url, $params, $opts

10.1.1.1. $opts

`$method`	
`$ttl`	
`$timeout`	
`$headers`	
`$ignore_errors`	bool

10.1.2. Exemples

```
HttpClient::get('/1/calc.add', array('x' => 1, 'y' => 1));
```

10.2. HttpRequest

10.2.1. API

10.2.1.1. Méthodes

`setParams()`
`setUrl()`
`setHeaders()`
`setHeader()`
`setBody()`

`setMethod()`	
`send()`	Retourne une instance de `HttpResponse`

10.2.1.2. Méthodes statiques

`getInstance()`	$url, $params, $opts

10.2.1.3. $opts

`$method`	
`$params`	
`$body`	
`$timeout`	
`$headers`	
`$ignore_errors`	bool

10.2.2. Exemples

10.3. HttpResponse

10.3.1. API

10.3.1.1. Méthodes

`getContent()`
`debug()`
`getJson()`
`getXml()`
`getStream()`
`getMetas()`
`getHeaders()`
`getStatus()`
`isError()`

10.3.1.2. $opts

`$method`	
`$params`	
`$body`	
`$timeout`	
`$headers`	
`$ignore_errors`	bool

10.3.2. Exemples

```
HttpRequest::getInstance('/calc', array('x' => 1, 'y' => 1))->send()->getBody();
```

11. AUTHENTIFICATION

11.1. Principe général

La gestion de l'authentification repose sur le fonctionnement suivant : lorsqu'une action nécessite que l'utilisateur soit authentifié, elle fait appel à la méthode `getUser()` de `$req`.

Cette méthode retourne un objet `User` si l'utilisateur est authentifié ; et émet une exception de type `AuthException` dans le cas contraire.

Il est ainsi possible d'attraper cette exception et de déclencher l'affichage d'une page d'identification avec la méthode `$res->sendAuthenticationForm()`.

La méthode `$res->sendAuthenticationForm()` fonctionne également dans le cas d'une requête AJAX. Le code d'erreur transmis dans la réponse JSON est : 401. Il permet de déclencher l'affichage d'une *dialog* d'authentification.

La grande force de ce système est de ne pas nécessiter un changement d'URL (et par là même une sauvegarde de l' « état » de la requête). Ainsi lorsque l'utilisateur s'identifie ou crée son compte, la page est rechargée avec la même URL mais cette fois l'action se déroule, si les informations sont correctes, sans émettre de `AuthException`.

Ce mode de fonctionnement met en exergue l'importance du respect des méthodes *GET* ou *POST* pour l'appel des actions. Une action qui réalise un affichage est appelée en *GET*; il est ainsi possible de recharger la page et l'authentification est compatible. Une action qui modifie l'état du système (ajout, modification, suppression d'un modèle) est appelée en POST (un rechargement, logiquement, échouera).

```
try {
  $user = $req->getUser();
  $forum = Forum::getInstance((int) $req->id);
  $topic = $forum->addTopic(array('title' => $req->title,
                                  'content' => $req->content));
  $topic->setUser($user);
  $res->sendMessage('article ajouté');
}
catch (AuthException $e) { $res->sendAuthenticationForm(); }
catch (Exception $e) { $err->add($e); }
```

Quelques remarques :

- la méthode `getUserId()` (qui retourne le `user_id`) émet également une `AuthException` ;
- il est essentiel de faire appel à `getUser()` ou `getUserId()` en haut de l'action ;
- l'objet retourné par `getUser()` est un ActiveRecord ;
- il est préférable d'utiliser `getUserId()` lorsque l'ActiveRecord `$user` n'est pas nécessaire, cela économise une requête.

Deux méthodes permettent de savoir si le visiteur est authentifié sans déclencher d'exception :

1. La consultation de l'attribut `_user_id` dans le contexte `$ctx`. Cette méthode est utile au sein d'un template pour afficher un contenu différent suivant que le visiteur est authentifié ou non.
2. L'utilisation de la méthode `$req->isAuthenticated()`.

```
<?php if ($_user_id >= 1): ?>
 Bonjour <?=$_user['name']?>!
<?php else: ?>
 Veuillez vous authentifier.
<?php endif; ?>
```

11.2. Informations du contexte

Une action authentifiée dispose de 2 variables supplémentaires dans son contexte `$ctx` : `_user_id` et `_user` qui est un *hash* contenant les clefs `id` et `name`.

En cas d'erreur lors de l'authentification, la variable `_user_err` contient l'origine de l'échec.

11.3. Paramétrage

La table contenant les identifiants et mots de passe est `social_user`.

La colonne identifiant est `email` et celle mot de passe, `password`.

11.4. Utilisateurs et Membres

Certaines applications ne peuvent se contenter des informations de la table `social_user`. Afin de ne pas modifier la table `social_user` (et par là même l'API), KWO permet de définir une table membre dans le fichier de configuration de l'application.

11.5. Verrouillage de l'ensemble de l'application

Cela peut être réalisé en ajoutant une directive de configuration au niveau de l'application.

```
$parameter['auth_required'] = true;
```

Certaines actions passent tout de même à travers cette protection : `core/captcha`, `core/snippet`, `social/password.send`, `core/password.set`, `core/user.confirm`.

12. CACHE

Les objectifs ci-dessous ont orienté la conception du gestionnaire de cache de KWO :

- compatibilité entre le cache « interne » et un cache extérieur de type Varnish ;
- capacité du développeur d'indiquer les requêtes à ne pas mettre en cache.

12.1. Gestionnaire de cache

12.1.1. Cache interne / Cache externe

KWO dispose d'un gestionnaire interne de *cache*. Il peut être activé en associant une valeur supérieure à 0 au paramètre `cache.ttl`. Toutes les requêtes (en dehors du back office) sont alors mises en cache.

Pour empêcher la mise en cache, le développeur fait appel, au sein de l'action, à `$res->disableCache()`.

Dans le cas où Varnish est préféré au cache interne de KWO, l'en-tête HTTP `X-Kwo-Disable-Cache` (avec la valeur 1) doit être pris en compte au niveau du fichier de configuration[7].

12.1.2. Conseils

Il est important d'empêcher la mise en cache notamment dans les cas suivants :

- la vue utilise une session ou un cookie ;
- la vue est spécifique à l'utilisateur (ex. panier de l'utilisateur).

12.1.3. Vider le cache

Cette opération peut être réalisée avec la commande suivante :

```
> php init.php cache.clean
```

Il est possible de désactiver le Cache en associant la valeur 1 à la clef `cache.disabled`.

Les fichiers de cache sont situés dans le répertoire : `var/cache`.

12.2. Cache client

La mise en cache d'une action au niveau du navigateur peut être réalisée en faisant appel à la méthode `$res->cache()`. Un tableau associatif `$opts` peut optionnellement être passé. La clef `ttl` représente le temps de la mise en cache.

[7] Le paramètre cache.ttl ne doit pas être utilisé dans le cas d'un serveur de cache externe.

13. API ET WEBSERVICES

13.1. Webservices

Ce composant permet au site de mettre à disposition une API via des webservices.

Une API peut être assimilée à un contrat entre l'éditeur du site et le développeur qui met en œuvre les fonctionnalités disponibles en mode SAAS. Il est par conséquent indispensable que le développeur soit sûr que les appels qu'il va faire aux webservices ne vont pas disparaitre ou être modifiés du jour au lendemain. Un *versioning* est par conséquent mis en œuvre pour permettre une évolutivité des webservices sans risquer de *casser* les appels réalisés par les développeurs. Une version d'API est toujours passée dans l'URL d'appel aux webservices.

Les appels aux webservices peuvent être réalisés de deux manières différentes :

- Appel centralisé : tout appel passe toujours par la même URL (une URL par format d'échange). Les paramètres sont transmis dans le corps (*body*) de la requête (méthode POST).
- Appel REST : chaque appel met en œuvre une URL spécifique qui contient le nom de la méthode.

Dans les deux cas le nom de la méthode correspond à un nom d'action

`lib/<EXTENSION_PROJET>/api/<METHODE>.inc`.

Les paramètres sont accessibles, comme pour une requête standard, via les attributs de l'objet `$req`.

13.1.1. Appel centralisé

Les actions doivent être placées dans le répertoire api : `lib/<EXTENSION_PROJET>/api/<ACTION>.inc`.

13.1.1.1. XML-RPC

La mise en place d'un web service en mode XML-RPC ne nécessite aucun développement particulier. Une simple convention d'appel est nécessaire.

Action	`http://domain/1/services.xmlrpc`
Méthode	Nom de l'action de l'extension projet (`P('app.extension')`).
Paramètres	une représentation de type de *hash* où chaque clef correspond à un paramètre

KWO dispose d'une classe client *web service* client qui permet d'illustrer ce fonctionnement.

```
$client = new RpcClient('domain', '/1/services.xmlrpc');
$client->setMethod('calc.add');
$client->setParams(array('x'=>12, 'y'=>'2'));
print_r($client->call());
```

L'exécution de cette action provoque l'affichage du tableau suivant :

```
Array
(
    [0] => Array
        (
            [resultat] => 14
        )

)
```

L'action `calc.add.inc` de l'extension test contient quant à elle le code suivant :

```
$ctx->resultat = $req->x + $req->y;
```

En ajoutant `$client->debug()` le fichier de log vous permet de d'observer les différentes enveloppes XML échangées.

Paramètres

```
[18-oct-2007 16:34:00] 0.044 [apache:26824] DEBUG
array (
  'x' => 12,
  'y' => '2',
)
```

Paramètres au format XML-RPC

```
[18-oct-2007 16:34:00] 0.044 [apache:26824] DEBUG
<?xml version="1.0" encoding="iso-8859-1"?>
<methodCall>
<methodName>calc.add</methodName>
<params>
 <param>
  <value>
   <struct>
    <member>
     <name>x</name>
     <value>
      <int>12</int>
     </value>
    </member>
    <member>
     <name>y</name>
     <value>
      <string>2</string>
     </value>
    </member>
   </struct>
  </value>
 </param>
</params>
</methodCall>
```

Réponse au format XML-RPC

```
[18-oct-2007 16:34:00] 0.044 [apache:26824] DEBUG
<?xml version="1.0" encoding="iso-8859-1"?>
<methodCall>
<params>
<param>
 <value>
  <struct>
   <member>
    <name>resultat</name>
    <value>
     <int>14</int>
    </value>
   </member>
  </struct>
 </value>
</param>
</params>
</methodCall>
```

Réponse au format PHP

```
[18-oct-2007 16:34:00] 0.044 [apache:26824] DEBUG
array (
  0 =>
  array (
    'resultat' => 14,
  ),
)
```

13.1.1.2. SOAP

Ce service fonctionnera de la même manière que le XML-RPC. Bien que les bases soient déjà présentes, la gestion de ce protocole n'est pas encore finalisée au sein de KWO.

Le développement d'une action de génération du fichier WSDL est un projet qui devrait influencer le développement de KWO4.

13.1.1.3. XML

Il s’agit d’un format hybride.

Chemin : `http://domain/1/services.xml`

Exemple du format de requête

```
<request version="calc.add">
 <x>1</x>
 <y>2</y>
</request>
```

Exemple de format de réponse

```
<response error="0">
 <resultat>3</resultat>
 <y>2</y>
</response>
```

13.1.1.4. JSON-RPC

13.1.2. Appel REST

Il est également possible de faire appel aux webservices en mode REST. Les paramètres sont alors transmis directement dans l'url et le format de retour est JSONRPC.

Exemple d'appel : `http://domain/1/calc.add?x=1&y=2`

13.1.3. Limitations

Les actions API ne peuvent accéder aux cookies et aux sessions.

13.2. Formats des réponses

Pour les scopes account, front, middle, KWO permet de forcer le format d'une réponse.

Les exemples ci-dessous se basent sur le contexte (`$ctx`) suivant :

```
$ctx->x = 1;
$ctx->colors = array('blue', 'red');
$ctx->user = array('firstname' => 'linus',
                   'lastname' => 'torvalds');
```

Le paramètre `kof` (***k**ernix **o**utput **f**ormat*) permet de spécifier le format souhaité. La méthode `setFormat()` de `$res` permet également de forcer un type de format (se reporter à la liste des constantes de la classe `Response` pour l'argument passé à `setFormat()`).

13.2.1. Paramètre kof

13.2.1.1. Format PHP

Le contexte est retourné sous la forme d'un *hash* PHP sérialisé.

`http://www.domaine.com/test/data?kof=php`

```
a:3:{s:1:"x";i:1;s:6:"colors";a:2:{i:0;s:4:"blue";i:1;s:3:"red";}s:4:"user";a:2:{s:9:"firstname";s:5:"l
inus";s:8:"lastname";s:8:"torvalds";}}
```

13.2.1.2. Format JSON

Le contexte est retourné sous la forme d'un *hash* JSON.

`http://www.domaine.com /test/data?kof=json`

```
{"x":1,
 "colors":["blue","red"],
 "user":{"firstname":"linus",
         "lastname":"torvalds"}}
```

Ce format est également utilisé lorsqu'une requête est réalisée avec `Kwo.exec()` en mode *non update*.

13.2.1.3. Format XML

Le contexte est retourné sous la forme d'un fichier XML.

`http://www.domaine.com/data?kof=xml`

```
<response error="0">
  <x>1</x>
  <colors>blue,red</colors>
  <user>
    <firstname>linus</firstname>
    <lastname>torvalds</lastname>
  </user>
</response>
```

14. TACHES AUTOMATIQUES

14.1. Principe

Une tâche correspond à un fichier placé dans un des sous répertoires `cron.hourly`, `cron.middaily`, `cron.daily`, `cron.weekly`, `cron.monthly`, `cron.yearly` du répertoire `cli` d'une extension.

Le nom du fichier commence par deux chiffres suivis du nom de l'action CLI (sans le `.inc`). Le fichier peut être créé avec la commande `touch`.

```
-rw-r--r-- 1 root root   0 Nov 23 10:10 lib/tracker/cli/cron.daily/20report.build
-rw-r--r-- 1 root root 153 Nov 20 16:14 lib/tracker/cli/report.build.inc
```

Les deux premiers chiffres permettent de définir une priorité. Cette dernière n'est généralement pas importante. Les préfixes commençant par 0 ou 9 sont réservés et ne doivent pas être utilisés.

La mise en œuvre des tâches automatiques de KWO nécessite l'exécution de l'action `scheduler` toutes les heures. L'ajout de la ligne suivante dans `/etc/cron.hourly/kernix_hourly.sh` est la méthode standard d'y parvenir.

```
cd /var/web/monsite && /usr/local/bin/php init.php scheduler
```

14.2. Blocage

Il est courant de se retrouver dans la situation de devoir bloquer les tâches automatiques associées à une application. KWO prévoit cette situation en permettant de déposer un fichier `scheduler.stop` à la racine du compte.

```
> cd /var/web/monsite && touch scheduler.stop
```

Il est conseillé de bloquer les tâches automatiques lors :

- d'une intervention sur la base,
- de la mise à jour du code de l'application.

14.3. Application

Il est possible de forcer l'utilisation d'une application en initialisant la variable d'environnement `KWO_APPLICATION`.

```
> setenv KWO_APPLICATION monsite
> php init ping
```

14.1. Avertissements

Dès qu'une tâche a une erreur associée (`$err->add()`), un mail automatique est transmis à l'adresse email `P('contact.devel')`.

15. Hooks

Les *hooks* permettent d'exécuter du code à différents moments cruciaux du flux d'exécution d'une action sans avoir à modifier le framework.

15.1. Hooks de méthodes

Ces méthodes peuvent être surchargées et permettent de réaliser des contrôles et des actions spécifiques avant et après certains évènements critiques de la vie d'un ActiveRecord.

Attention, la visibilité des méthodes ci-dessous est `protected`.

`onAfterCommit()`	&$h	
`onAfterCreate()`	&$h	
`onAfterDestroy()`		
`onAfterUpdate()`	&$h	
`onAlter()`		Appelée après `commit()` et `destroy()`. Notamment utile pour nettoyer les caches ou mettre à jour un objet lié.
`onBeforeCommit()`	&$h	
`onBeforeCreate()`	&$h	
`onBeforeDestroy()`		
`onBeforeUpdate()`	&$h	

```
print($topic);
class Topic extends ActiveRecord
{
  public function onAfterCreate() {
    $this->getForum()->increment(Meta::CHILD_COUNT);
  }
}
```

Certaines méthodes reçoivent en argument un *hash* d'attributs à mettre à jour. Il s'agit d'une référence (`&`) sur ce *hash* ce qui permet de le mettre à jour avant l'exécution de la requête SQL.

15.2. Hooks d'évènements

Les méthodes décrites ci-dessous doivent doit être définies de manière à pouvoir recevoir en argument un hash de paramètres.

Il s'agit de méthodes statiques optionnelles présentes dans le manager.

Les hooks assurent une évolutivité extrême du framework tout en évitant d'alourdir les méthodes de certaines classes critiques (ex. `User`, `Order`). Il serait par exemple tout à fait aberrent de vouloir prévoir tous les processus possibles qui peuvent faire suite à la création d'un nouvel utilisateur. Il est largement préférable de réaliser un petit développement spécifique pour chaque projet.

Les Hooks d'évènements ne sont pas appelés en mode *CLI*.

`onActionStart()`	Appelée avant l'exécution d'une action. Cette méthode est particulièrement conseillée pour compléter le contexte `$ctx`. Paramètre : aucun Propagation des exceptions : non
`onActionStop()`	Appelée après l'exécution d'une action. Paramètre : aucun Propagation des exceptions : non
`onActionFail()`	Appelée en cas d'erreur lors de l'exécution de l'action. Cette méthode peut être pratique lors d'une migration de site lorsque l'on souhaite maintenir des URLs en réalisant des redirections permanentes. Paramètre : aucun Propagation des exceptions : non
`onAfterCommit()`	Appelée après chaque *commit* sur un ActiveRecord. Paramètre : `item`, `attrs`
`onAfterStore()`	Appelée après chaque enregistrement (BO) sur un ActiveRecord. Paramètre : `item`
`onAnonymousUpload()`	 Paramètre : `file`
`onAppStart()`	 Paramètre : aucun Propagation des exceptions : non
`onAppStop()`	 Paramètre : aucun Propagation des exceptions : non
`onBeforeStore()`	Appelée avant chaque enregistrement (BO) sur un ActiveRecord. Paramètre : `item`
`onConversionAdd()`	 Paramètre : item
`onConversionRemove()`	 Paramètre : item
`onEmailChange()`	 Paramètre : `user`, `from` (`email_id` précédent), `to` (`email_id` actuel)

`onLetterDeliver()`	Paramètre : `letter`, `mail`, `recipient` (email), `bindings`, `opts`
`onMiddleStart()`	
`onVisitNew()`	Nouvelle visite. Paramètres : `visit_id` et `visitor_id`
`onOrderCheckout()`	Appelé dans le controller associé à chacune des étapes du processus d'achat. cart, order.amounts, order.details, order.shipping.methods, order.discount, order.shipping, order.method, order.coupon.compose, order.billing, order.shipping.carriers
`onOrderLoad()`	Appelé dans toutes les actions front offices où un panier ou une commande est nécessaire. Paramètre : order
`onOrderUpdate()`	Appelé dès qu'un achat est ajouté/supprimé du panier. Paramètre : order
`onPrivilegeAdd()`	Ajout d'un privilège. Paramètre : user, privilege_id
`onPrivilegeRemove()`	Suppression d'un privilège. Paramètre : user, privilege_id
`onRequestDispatch()`	Permet d'influencer l'initialisation du controller. (en utilisant $req->setParameter(), scope, extension, action) Paramètre : req
`onSearch()`	Paramètre : searcher
`onSubmissionAdd()`	Paramètre : submission
`onUpload`	Paramètre : `file`
`onUserAuth()`	Appelée au moment de l'authentification d'un utilisateur. Paramètre : `user_id`
`onUserCreate()`	Appelé au moment de la création d'un compte utilisateur. Paramètre : `user`

`onUserUpload()`	Paramètre : `file`
`onUserValidate()`	Appelé avant la création d'un compte utilisateur. Utilisé pour vérifier que tous les éléments nécessaires à la création d'un compte sont remplis. Paramètre : aucun

```
<?php

class MonApp
{
  public static function onUserCreate($h) {
    S('log')->trace('new user ', $h['user']->email);
  }
}
```

16. Statistiques et tracking

16.1. Tracking

Le tracking repose essentiellement sur l'utilisation de la méthode `track()` de l'objet requête `$req`.

Deux modes sont disponibles selon les paramètres passés.

16.1.1. Tracking spécifique

La méthode `track()` prend deux arguments. Le premier correspond au nom de l'évènement à tracer. Le second correspond à un nom de groupe d'évènements.

Il est essentiel de définir ces groupes avant la mise en production du projet pour ne pas polluer les statistiques.

```
$req->track('ajout panier', 'boutique');
```

Plusieurs évènements peuvent être tracés dans le cadre d'une même action. Plusieurs appels à `track()` sont alors réalisés.

Deux groupes spécifiques doivent être utilisés pour reporter des erreurs ou des avertissements : `errors`, `warnings`.

16.1.2. Tracking d'ActiveRecord

La méthode `track()` prend alors en argument un ActiveRecord. Le nom de l'évènement correspond à la valeur retournée par la méthode `getName()` de l'ActiveRecord, et le groupe, au nom du modèle. Pour rappel, la méthode `getName()` (si elle n'est pas surchargée) retourne la valeur de la colonne `name`.

Ce mode a également l'avantage de tracer les accès uniques aux différents `ActiveRecords` et de garder une trace de ces accès dans la table `core_item_intent`. La métadonnée `display` est alors mise à jour. Il convient de bien comprendre que cette donnée correspond à la consultation unique d'un objet donné par un visiteur donné durant une visite. La valeur `display` est par là même très précise en ne prenant pas en compte des personnes affichant plusieurs fois une même page pendant leur visite.

A titre d'exemple `display` est utilisé dans le module Forum pour afficher le nombre de consultations d'un sujet.

La table `core_item_intent` est également très intéressante pour générer des statistiques sur les objets les plus populaires d'un modèle donné.

16.1.3. Interdiction du tracking

Il est possible d'empêcher le tracking associé à une action en passant le paramètre ntf (no tracking flag) avec la valeur 1.

16.2. Conversions

Une conversion correspond à un évènement utilisateur représentant une valeur marchande. A titre d'exemple il est possible de lister : la création d'un compte, une réponse à un formulaire, le paiement d'une commande.

La création d'une conversion repose sur la méthode statique `convert()` de la classe `Conversion`. Cette méthode prend en paramètre un `ActiveRecord`.

```
Conversion::add($command);
```

17. Entrees / Sorties

Les objets liés aux entrées/sorties (I/O) émettent des exceptions de type `IOException`.

17.1. File

17.1.1. Utilisation

17.1.1.1. Instanciation

17.1.2. API

17.1.2.1. Méthodes

`asArray()`		Retourne un tableau ou chaque cellule correspond à une ligne du fichier.
`close()`		
`copyIn()`	$folder_path	
`copyTo()`	$file_path $same_path=false	Réalise une copie du fichier. `$file_path` correspond au chemin complet du nouveau fichier. Si `$same_path_flag` contient la valeur `true`, le fichier est copié dans le même répertoire que le fichier d'origine et seul le nom de `$file_path` est pris en compte. Retourne un objet `File` correspondant au nouveau fichier.
`getChunk()`	$size=128	
`getContent()`		Retourne le contenu du fichier.
`getCtime()`		
`getFolder()`		Retourne un objet `Folder` correspondant au répertoire dans lequel se trouve le fichier.
`getExtension()`		Retourne l'extension d'un fichier.
`getFileName()`		Retourne le nom du fichier.
`getFilePath()`		Retourne le chemin du fichier.
`getNext()`		Retourne la ligne suivante dans le fichier. Retourne `false` une fois que le fichier est parcouru en entier.
`getPath()`		Retourne le chemin du répertoire dans lequel se trouve le fichier.
`getSize()`		
`isFolder()`		
`isImg()`		
`isText()`		

`MoveIn()`	$folder_path	
`moveTo()`	$folder_path	Retourne `$this`.
`rename()`	$new_file_name	Renomme le fichier.
`rewind()`		Permet de replacer le pointeur interne du fichier au début.
`unlink()`		Supprime le fichier.
`writeln()`	$str	Ajoute un `\n` à la fin de la ligne.
`write()`	$str	

17.2. Répertoire

Un répertoire est représenté par un objet `Folder`.

17.2.1. Utilisation

17.2.1.1. Instanciation

Un objet `Folder` doit être initialisé à l'aide de la méthode statique `getInstance()`.

```
$folder = Folder::getInstance(DOC_PATH);
```

Un second argument correspondant au mode d'ouverture peut également être passé. Les modes disponibles sont lecture (`'r'`), écriture (`'w'`).

```
$folder = Folder::getInstance(DOC_PATH, 'w');
```

17.2.1.2. Parcours

Un objet `Folder` peut être parcouru à l'aide d'un `foreach`. Les répertoires sont également présents.

```
foreach (Folder::getInstance(DOC_PATH) as $file_name) {
  print($file_name."\n");
}
```

17.2.2. API

17.2.2.1. Attributs

Les données de la session.

17.2.2.2. Méthodes

```
addFolder()
```

`asHash()`		Retourne un *hash*. Le *hash* est classé dans l'ordre alphabétique. Les répertoires sont placés avant les fichiers.
`compress()`		
`getFolders()`		
`getFiles()`		
`getHtmlPath()`		
`getNbFiles()`		
`getParentPath()`		
`getPath()`		
`getUsedSpace()`		
`hasFile()`		
`hide()`		
`isRoot()`		
`isWritable()`		
`receive()`		
`showPath()`		
`size()`		

17.3. Images

Les formats gérés sont : `.jpg`, `.gif`, `.png`.

17.3.1. Utilisation

La classe `Img` hérite de la classe `File`.

17.3.2. API

17.3.2.1. Méthodes

`getWidth`		
`getHeight`		
`setWidth`	$width, $preserve_ratio=false	Retourne `$this`.
`setHeight`	$height, $preserve_ratio=false	Retourne `$this`.
`setBox`	$width, $height=0	Retourne `$this`.
`setSize`	$width, $height=0	Retourne `$this`.

`rename`		Retourne `$this`.
`rotate`	$angle	Retourne `$this`.
`crop`	$x1, $y1, $x2, $y2	Retourne `$this`.

17.3.3. Exemples d'utilisation

17.3.3.1. Chaînage d'opération

```
$img = Img::getInstance('test.jpg').setBox(100).rename('avatar.jpg');
```

Le changement de nom contrôle l'extension du fichier et change automatiquement le format de l'image en cas de besoin.

```
$img = Img::getInstance('test.jpg').rename('test.gif').setBox(100);
```

18. MOTEUR DE RECHERCHE

18.1. Indexation

Tout objet ActiveRecord dispose de la méthode `asDocumentHash($locale)`.

Cette méthode peut être surchargée pour modifier le contenu à indexer.

Les éléments suivants peuvent être remplacés :

```
title
```

```
description
```

```
content
```

```
keywords
```

```
locale
```

```
user_id
```

```
class Article extends ActiveRecord {
  public function asDocumentHash() {
    $h = parent::asDocumentHash();
    $h['title'] = $this->getHeading()->getTitle().' - '.$this->title;
    $opts = array('order' => array('position', 'title'));
    foreach ($this->getChapters($opts) as $chapter) {
      $h['content'] .= "<h3>".$chapter->title."</h3>";
      $h['content'] .= $chapter->content;
                }
    return $h;
  }
}
```

L'indexation est automatique. Les modèles à indexer sont précisés dans le fichier de configuration de l'application.

```
$parameter['search.models'] = array(MODEL_PAGE, MODEL_PRODUCT);
```

Tous les objets modifiés ou créés la veille sont indexés. Un objet devant être indexé doit donc disposer d'une colonne `updated_at` (indexée si possible).

Certaines extensions permettent de forcer l'indexation temps réels de leurs objets. La méthode `onAlter()` est conseillée.

Un ActiveRecord est indexé à l'aide de sa méthode `index()`.

```
$catalog = Catalog::getInstance(1);

$opts = array('filters' => array('status' => 1),
              'limit' => 100);

foreach ($catalog->getProducts($opts) as $product) {
  $product->index();
}
```

La méthode `isIndexable()` peut être surchargée pour définir exactement si l'`ActiveRecord` doit être indexé ou non.

18.2. Recherche

La recherche repose sur l'objet `Searcher`.

```
$searcher = new Searcher(array('query' => $req->query,
                               'limit' => 20,
                               'offset' => $req->offset));
$res->write($searcher->asText());
```

La méthode `asText()` peut prendre en argument un *template*.

19. INTERNATIONALISATION

19.1. Paramétrage

Une application peut être internationalisée à partir du moment où son fichier de configuration contient le paramètre `locales` :

```
$parameter['app.locales'] = array(I18N::LANG_FR, I18N::LANG_EN);
```

19.2. Dictionnaire

Les termes du dictionnaire sont créés dans le BO dans *Cms>Phrases*.

Chaque modification d'une valeur entraine la génération de fichiers `var/locales/messages_XX.inc`.

Le fichier correspond à une simple table de *hash*.

```
<?php
$strings = array('add' => 'ajouter',
                 'close' => 'fermer',
                  …
```

Lorsque plusieurs termes sont reliés ensemble autour d'un concept commun, la notation doit respecter la règle suivante : `concept1.concept2.conceptN.clef`.

```
  'error.cart.invalid' => 'panier invalide',
  'error.coupon.amount_too_low' => 'montant de la commande insuffisant',
  'error.coupon.expired' => 'ce coupon a expiré',
  'error.coupon.unknown' => 'code coupon inconnu',
  'error.coupon.used' => 'ce coupon a déjà été utilisé',
  'error.product' => 'produit invalide',
  'error.product.notinsale' => 'le produit n\'est plus disponible à la vente',
```

19.3. Fonction l()

L'utilisation des termes définis dans ces dictionnaire repose sur la fonction globale `l()`.

```
<div><a href="void(window.close())"><?=l('close')?></a></div>
```

La fonction `l()` récupère la traduction de *close* dans le dictionnaire correspondant à la langue du visiteur.

Il convient de différencier la langue de la requête et la langue de la réponse.

La langue de la réponse peut être forcée en préfixant l'url de la page avec le code de la langue.

Ex. `http://domain/fr/page.view`

Avec une telle URL la réponse sera en français quelle que soit la langue choisie par le visiteur. Cette fonctionnalité est essentielle pour le référencement de sites internationaux (les robots des moteurs de recherche sont en effet dans l'incapacité de changer de langue).[8]

Le contexte contient en permanence la variable `$_locale` (`$ctx->_locale`) qui permet de préfixer de manière automatique les URLs.

```
<a href="/<?=$_locale?>/content/page.view/-/code/contact"><?=l('contact_us')?></a>
```

19.4. Appel à des contenus

19.4.1. l('snippet:code')

Il s'agit des messages administrés dans « Système > Fragments ». Le code et la langue sont essentiels.

19.5. Bindings

Il est possible de transmettre un deuxième paramètre à `l()` correspondant à des *bindings* à remplacer dans le contenu.

Si l'élément `'hello'` contient `'Hi ((m)) !'`, la syntaxe remplace `((m))` par `Mr`.

```
<?=l('hello', array('m' => 'Mr'))?>
```

19.6. Forcer une langue

La méthode `l()` peut prendre un troisième argument correspondant à une locale.

```
<?=l('hello', array('m' => 'Mr'), I18N::LANG_EN)?>
```

Utiliser la valeur `null` comme deuxième paramètre si ce dernier n'est pas nécessaire.

19.7. ActiveRecord et Properties

Les attributs d'un `ActiveRecord` donnent accès aux attributs ainsi qu'aux `properties`. Les méthodes `get/set/hasAttribute()` ne donnent accès qu'aux attributs.

Les méthodes `setProperties()` et `setProperty()` permettent de modifier la valeur d'une *property*.

Les méthodes `getProperties()` et `getProperty()` donnent accès aux *properties*.

Les propriétés d'un ActiveRecord sont définies dans la rubrique : ***Système > Propriétés***.

L'option `$locale` peut être passé dans le `$opts` de `ActiveRecord` ou `Collection`.

[8] Un nom d'extension doit par conséquent contenir au moins trois lettes.

```
$product = Product::getInstance(1, array('locale' => I18N::LOCALE_EN));
echo $product->getProperty('title');
```

Lorsqu'un modèle dispose d'une colonne `locales`, il devient possible de préciser les locales dont dispose l'item.

Si cette colonne n'est pas présente et que le modèle dispose de *properties*, la méthode `getLocales()` retourne l'ensemble des locales dont dispose l'application.

20. Routage

Les routes permettent de redéfinir les URLs de certaines actions sans recourir à l'*Url Rewriting*.

Les routes doivent être placées dans le fichier de configuration de l'application de la manière suivante :

```
$parameter['front.routes'] = array(array('post/(?P<id>\d+)', 'post'),
                                   array('(.*)/art/(?P<id>\d+)', 'topic', array('x' => 2)));
```

L'exemple ci-dessus définit 3 routes :

1. Une url de la forme `/post/12` devient `/post` avec un `$req` disposant d'un attribut `id` contenant la valeur `12`.
2. Une url de la forme `/blablabla/art/12` devient `/topic` avec un `$req` disposant d'un attribut `id` contenant la valeur `12`, et d'un attribut `x` à `2`.

Deux remarques par rapport à l'expression régulière :

- le caractère / n'a pas besoin d'être « échappé » ;
- l'url transmise est nettoyée de l'éventuelle présence d'un / en début ou à la fin de chaîne.

21. JavaScript

Deux méthodes centrales doivent être utilisées systématiquement pour toute requête aussi bien AJAX qu'HTTP.

21.1. Kwo.exec()

Cette méthode permet de réaliser une requête AJAX.

```
Kwo.exec("/test/exec", {"x": 1});
```

Cette commande appelle en AJAX l'action `lib/test/front/exec.inc` en lui transmettant le paramètre `x` (ayant la valeur `1`).

21.1.1. Arguments

21.1.1.1. Premier argument : la ressource

Il s'agit de l'url de l'action.

21.1.1.2. Deuxième argument : les arguments

Il peut s'agir :

- d'un objet JavaScript (ex. `{"x" : 2, "y" : 2}`) ;
- d'un contrôle d'un formulaire (`input`, `textarea`, `select` etc.) : le formulaire associé est alors automatiquement récupéré ;
- d'un formulaire ;
- d'un élément HTML disposant de l'attribut `data-values` (la valeur associée à cet attribut doit être générée avec `H::asDataValues($h)`) ;
- d'un tableau composé d'éléments listés ci-dessus.

21.1.1.3. Troisième argument : les options

Cet objet JavaScript, souvent appelé `opts`, peut contenir les éléments suivants :

`async`	bool	Précise si la requête est en mode synchrone ou non.
`callback`	function	Fonction appelée suite à une exécution AJAX. Le hash JSON généré par l'action est transmis à la fonction. L'utilisation de la méthode Prototype `Function.bind(obj)` est essentielle pour pouvoir propager la variable `this` au sein de la fonction. `{"callback": truc.machin.bind(truc)}` Si l'option `container` est précisée, ce callback permet de déclencher une action une fois que le contenu de la zone a été mis à jour.
`confirm`	string	Message affiché dans le cadre d'une fenêtre `confirm()`. Il peut également s'agir d'un élément qui contient un attribut `data-confirm`.
`container`	dom_id	Bloc qui va être mis à jour suite à l'action AJAX. Il peut s'agit d'un élément HTML ou d'un id d'élément HTML.

`disable`	elt	Si l'argument correspond à un élément formulaire, le formulaire passe en *disabled*.
`reset`	elt	Si l'argument est un formulaire, le formulaire sera remis à zéro en cas de succès.

21.1.2. L'utilisation d'un callback

Le `callback` reçoit un objet json disposant de 2 attributs « parent » : `error` et `result`. Cet objet doit, par convention, être nommé `resp`. Lorsque l'opération se déroule correctement la valeur de `error` est égale à `0` et le `result` correspond à un objet JavaScript contenant les valeurs du `$ctx` de l'action appelée.

Une vérification de l'attribut `error` du json doit absolument être réalisée. Une valeur supérieure ou égale à 1 signifie en effet qu'une erreur a eu lieu au niveau de l'action. Les éventuels messages d'erreurs provenant de `$err->add()` sont disponibles dans un tableau `messages` dans `result`.

Il est essentiel d'initialiser le contexte à la fin de l'action, à la suite de toutes les vérifications et de toutes les commandes susceptibles d'émettre des Exceptions. En fonctionnant ainsi le contexte est maintenu cohérent.

21.1.2.1. Classe JavaScript kwo.Response

La réponse reçue par la fonction callback peut être convertie en un objet `kwo.Response` de la manière suivante :

```
resp = new kwo.Response(resp);
```

Les méthodes disponibles sont les suivantes :

`hasAttribute`	attr	Vérifie si l'attribut attr est présent dans
`getAttribute`	attr	

`isAuthError`		Détecte s'il s'agit d'une erreur liée à l'authentification (cf. `AuthException` et `$res->sendAuthenticationForm()`). L'étape suivante consiste généralement à ouvrir une modale d'authentification : `Kwo.Auth()`.
		
		`onSubmitCallback: function(resp) {`
		`  resp = new kwo.Response(resp);`
		`  if (resp.isAuthError()) {`
		`  var am = new Kwo.Class.Auth();`
		`  am.onCallback = this.onSubmit.bind(this);`
		`  return ;`
		`  }`

`isRedirection`		
`redirect`		Redirection dans le cas d'un `$res->sendRedirect()`.
`alert`		Réalise un alert() en utilisant les messages d'erreur.
`getMessage`		Message transmis par `$res->sendMessage()`.
`hasMessage`		
`getResult`		Retourne l'objet `resp["result"]`.

```
getErrors
getError          separator="<br/>"
hasError
```

21.1.2.2. Exemple

L'exemple ci-dessous réalise l'addition de deux chiffres. Quatre scripts sont mis en œuvre :

- `lib/test/front/calc.inc` et `web/test/templates/calc.psp` pour l'affichage du formulaire ;
- `lib/test/front/calc.sum.inc` réalise l'exécution de l'addition ;
- `web/test/scripts/calc.js` est responsable de la gestion de l'interaction entre le formulaire et le script de calcul.

`lib/test/front/calc.inc`

```
<?php

try {
  $res->useTemplate();
}
catch (Exception $e) { $err->add($e); }
```

`web/test/templates/calc.psp`

```
<form>
 <input type="text" name="x" /> +
 <input type="text" name="y" /> =
 <span style="font-size:3em;"></span>
 <input type="submit" value="exécuter" />
</form>
<?=H::trigger('new kwo.calc(this)')?>
```

`lib/test/front/calc.sum.inc`

```
<?php

try {
  if (!is_numeric($req->x) || !is_numeric($req->y)) {
    throw new Exception('arguments invalides');
  }
  $ctx->sum = $req->x + $req->y;
}
catch (Exception $e) { $err->add($e); }
```

web/test/scripts/calc.js

```
kwo.calc = Class.create({

  "form": null,

  initialize: function(elt) {
    this.form = $(elt).previous();
    this.onBind();
  },

  onBind: function() {
    this.form.observe("submit", this.onSubmit.bindAsEventListener(this));
  },

  onSubmit: function(evt) {
    evt.stop();
    Kwo.exec("/test/calc.sum", this.form,
             {"disable": this.form,
              "callback": this.onCallback.bind(this)});
  },

  onCallback: function(resp) {
    resp = new kwo.Response(resp);
    if (resp.hasError()) return resp.alert();
    if (resp.hasAttribute("sum")) {
      this.form.down("SPAN").update(resp.getAttribute("sum"));
    }
  }

});
```

L'utilisation de `<?=H::trigger('new kwo.calc(this)')?>` est essentielle. Cette commande permet de déclencher l'exécution de code JavaScript à l'emplacement exact où elle est située sur la page. Cette méthode a l'avantage d'être compatible avec une mise à jour de contenu de type AJAX.

21.1.3. Callback + container

L'utilisation combinée des champs `callback` et `container` est nécessaire lorsque l'on souhaite pouvoir exécuter du code JavaScript une fois le container « repeint ».

Cela est très souvent utilisé pour réaliser des bindings sur des éléments HTML (ex. FORM, INPUT).

Attention, il n'est pas envisageable d'avoir au sein du code HTML retourné pour le container une balise <script>. Cette balise ne sera pas interprétée (voir `H::trigger()`).

```
kwo.test.marchand.onDisplay = function() {
  var container = $("marchand-box");
  var args = {"id": 1};
  Kwo.exec("/test/marchand", args,
           {"confirm": "êtes vous sûr ?",
            "container": container,
            "callback": function() {
              container.down("FORM").observe("submit", function(evt) {
                evt.stopPropagation();
                evt.preventDefault();
                alert("submit !");
              });
            }});
}
```

21.2. Kwo.go()

Cette méthode permet d'ouvrir une page.

Les deux premiers arguments sont les mêmes que `Kwo.exec()`.

Les options sont en revanche spécifiques.

`confirm`	string	Message affiché dans le cadre d'une fenêtre `confirm()`.
`popup`	true ou un objet	Cet objet peut avoir les clefs suivantes : `blank`, `width`, `height`.
`target`	dom_id ou "blank"	Ouvre la page dans une frame ou dans une popup blank.

21.3. Développement d'une fenêtre modale

Il convient d'hériter de la classe `Kwo.Dialog`.

```
kwo.test.marchand.composer.Geolocator = Class.create(Kwo.Dialog, {
  attr1: null,
  attr2: null,

  initialize: function($super, elt) {
    this.name = "geolocator";
    this.elt = $(elt);
    this.width = 640;
    this.height = 530;
    $super(this.onDisplay, null);
  },

  onDisplay: function() {
    Kwo.exec("/marchand.edit", this.args,
             {container: this.support,
              callback: this.onDisplayCompleted.bind(this) })
  },

  onDisplayCompleted: function() {
    …
  },
```

21.4. Requête AJAX et formulaire

Lorsqu'un formulaire doit être transmis en mode AJAX, plusieurs règles doivent être respectées :

- Le bouton de validation doit être de type `submit` ;
- Le `<form>` doit disposer d'un `onsubmit=""` qui fera appel à une méthode spécifique d'un objet JS suivie de `return false;` ;
- L'utilisation des noms de méthodes `onSubmit()` et `onCallback()` est conseillée ;
- Le bouton `submit` doit être passé en `disabled` pendant l'exécution de l'action et repassée en `enabled` au niveau du callback.

```
<script>
Kwo.Calc = {
  "button": null,
  "onSubmit": function (args) {
    Kwo.Calc.button = $(args).select("input[type=submit]")[0];
    Kwo.Calc.button.disable();
    Kwo.exec("/divide", args, {"callback": Kwo.Calc.onCallback});
  },
  "onCallback": function (res) {
    Kwo.Calc.button.enable();
    if (Kwo.hasError(res)) {
      return Kwo.error(res);
    }
    Kwo.warn("Résultat:" + res["result"]["valeur"]);
  }
};
</script>

<form onsubmit="Kwo.Calc.onSubmit(this); return false;">
 <label>x<label> : <input type="text" name="x" /><br/>
 <label>y<label> : <input type="text" name="y" /><br/>
 <input type="submit" value="diviser" />
</form>
```

```
<?php
try {
  if ($req->y == 0) {
    throw new Exception('division par zéro interdite');
  }
  $ctx->valeur = $req->x / $req->y;
}
catch (Exception $e) { $err->add($e); }
```

22. Helpers

Les *Helpers* sont des classes contenant des méthodes statiques.

22.1. H

Il s'agit des méthodes utilisées au sein d'un template.

`amount()`	$str, $currency_id=null	Utilisée pour l'affichage d'un prix.
`birthdate()`	$value, $opts=array()	Retourne trois SELECT permettant de préciser une date d'anniversaire. $opts permet de redéfinir le nom de l'input (`name`), l'id (`id`).
`breadcrumb()`	$prefix=null	Retourne une chaîne de caractères correspondant au fil d'Ariane.
`button()`	$label	Crée un bouton de type *submit*.
`collection()`	$collection	
`date()`	$date	Présentation d'une date adaptée à la locale.
`datenice()`	$date	Présentation évoluée.
`datetime()`	$date	Présentation d'une date et heure adaptée à la locale.
`escapeHtml()`	$str	Normalise une donnée HTML.
`escapeJs()`	$str	Normalise une donnée JS.
`get()`	$url, $args=null, $ttl=null, $key=null	Cette méthode permet de récupérer le contenu d'une URL. Cette méthode l'avantage de propager le cookie de tracking. Aucune mise en cache n'est réalisée par défaut.
`inc()`	$templates, $bindings=true	
`options()`	$h, $default=''	

`url()`	$action
	$params=null
	$opts=null
		Cette méthode retourne une url à partir des éléments passés en paramètre.
		`H::url(/toto, array('a' => 'b'))`
		devient
		`/toto?a=b`
		Elle prend en compte les paramètres d'application suivants : `host.name`, `host.port`, `host.secure`.

22.1.1. H ::elt()

Il est vivement conseillé d'utiliser cette méthode dans les actions ou les méthodes de classes.

```
$attrs = array('class' => array('selected',  'yellow'),
               'on' => array('click' => 'Kwo.go(this)',
                             'mouseover' => 'addclass(this)'),
               'data' => array('url' => '/mapage',
                               'values' => array('x' => 1,
                                                 'y' => 2)));
$html = H::elt('div', 'click here', $attrs);
```

est équivalent à

```
<div class="selected yellow"
     onclick="Kwo.go(this)"
     onmouseover="addclass(this)"
     data-url="/mapage"
     data-values="{&amp;quot;x&amp;quot;:1,&amp;quot;y&amp;quot;:2}">click here</div>
```

22.2. XH : Xml Helpers

Xml Helpers

22.2.1. Méthodes

`asHash()`	$doc (string ou DOMDocument)
`asTree()`	$doc (string ou DOMDocument)

22.2.2. Exemples

```
XH::asHash('<struct><x>1</x><y>2</y><x>toto</x></struct>') ;

Array
(
    [x] => Array
        (
            [0] => 1
            [1] => toto
        )

    [y] => 2
)
```

```
XH::asTree('<struct><x>1</x><y>2</y><x>toto</x> <y><z>3</z></y></struct>') ;

Array
(
    [tag] => struct
    [children] => Array
        (
            [0] => Array
                (
                    [tag] => x
                    [value] => 1
                )

            [1] => Array
                (
                    [tag] => y
                    [value] => 2
                )

            [2] => Array
                (
                    [tag] => x
                    [value] => toto
                )

            [3] => Array
                (
                    [tag] => y
                    [children] => Array
                        (
                            [0] => Array
                                (
                                    [tag] => z
                                    [value] => 3
                                )

                        )

                )

        )

)
```

22.3. DH : Date Helpers

`now()`		Retourne la date actuelle dans un format SQL.
`today()`		
`asHash()`	$date	La date peut être au format FR ou GB. Le *hash* de retour contient les clefs suivantes : `day`, `month`, `year`, `hour`, `min`, `sec`, `date`, `time`. Retourne `false` en cas d'erreur.

`isBetween()`	$from, $o, $date=null	Vérifie si la date est comprise entre `$from` et `$to`. Si le troisième argument n'est pas transmis, la date actuelle est utilisée.
`asTextual`		Affiche une date de la manière suivante : jeudi 25 janvier 2012
`isHolyday`	$date	Indique si une date est un jour férié. 1er janvier, 1er mai, 8 mai, 14 juillet, 15 août, 1er novembre, 11 novembre, 25 décembre + pacques, ascension et pentecôte.
`yesterday()`		
`isNull()`		

22.4. AH : Array Helpers

Array Helpers

`asXml()`	$h	

22.5. SH : String Helpers

String Helpers

`asArray()`	$str	Convertit une chaîne de caractères en *array*. Les caractères `(\r)?\n|,|;|\t` sont considérés comme caractères de séparation.
`asSet()`	$str	Crée un tableau en découpant suivant le caractère « virgule ». Les valeurs du tableau sont « trimées ».
`clean()`	$str $opts=null	Permet de filtrer une chaîne de caractères. Avec `$opts['html'] = true`, le contenu est filtré en tant que contenu HTML.
`crypt()`	$str	Méthode générique de cryptage d'une chaîne reposant sur une clef secrète définie au niveau de la plateforme.
`password()`	$length=8	Retourne un mot de passe.
`spacer()`		Retourne une chaîne de caractères correspondant à une image (GIF) transparente.
`stripAccents()`	$str	Retourne une chaîne de caractères où les accents sont remplacés par la lettre associée non accentuée.
`toSize()`	$size	Transforme une taille de fichier dans un format lisible. Ex. `1.2M`, `1.2K`, `2`
`urlify`	$str	Adapte une chaîne à une utilisation dans une url. Ex. `"L'île  aux PIRATES"` devient `"l-ile-aux-pirates"`

22.1. IH : Ip Helpers

`dnsblquery()`	$ip	Vérifie si l'IP est dans un DNSBL (black list).
`ip2country()`	$ip, $return_code=false	Retourne le `country_id`.
`ip2host()`		
`ip2num()`		
`isOpenedPort()`	$ip, $port, $timeout=3	
`num2ip()`		

23. Classes statiques

23.1. Valid

`isAlphanum()`	$str, $strict=false	Si `$strict_flag` contient `false`, les caractères - et _ sont également acceptés.
`isEmail()`	$email, $check_dns=false	Vérifie si l'adresse email est valide. Le deuxième argument permet de contrôler si le domaine associé dispose bien d'un champ MX.
`isFloat()`	$str	
`isNum()`	$str	
`isRib()`	$rib_hash	
`isSiren()`	$str	
`isSiret()`	$str	
`isUrl()`	$str	
`isWord()`	$str	

23.2. W : Widgets

Les widgets correspondent à des modules graphiques qu'il est possible de placer sur les pages et qui doivent être réutilisables.

Les widgets peuvent généralement être utilisées sans que l'extension associée soit installée. Elles retournent dans ce cas une chaîne vide. Elles se mettront cependant à fonctionner dès l'installation de l'extension. Il est donc conseillé de placer les widgets dès le départ afin de ne pas avoir à repasser sur tous les templates d'un projet une fois que ce dernier est lancé.

23.2.1. Abus

Ouvre une boite permettant de reporter un abus lié à un ActiveRecord.

Ce widget est lié à l'extension `abuse`.

```
<?=W::abuse($topic)?>
```

argument 1		Un ActiveRecord.
argument 2	optionnel	

23.2.2. Captcha

Affiche une boite permettant de renseigner un code secret. Aucun développement au niveau serveur n'est nécessaire.

```
<form>
<?=W::captcha()?>
</form>
```

23.2.3. Choix de langue

La méthode `W::locale()` permet d'afficher un petit menu permettant de sélectionner sa langue d'affichage. La clef de registre `locales` est prise en compte.

23.2.4. Commentaire

Affiche une boite contenant les commentaires liés à un ActiveRecord.

Ce widget est lié à l'extension `comment`.

```
<?=W::comment($product)?>
```

argument 1		Un ActiveRecord.
argument 2	optionnel	

23.2.5. Erreur

Affiche une boite d'erreur.

```
<?=W::error('Produit inconnu')?>
```

24. Conventions et Normes

24.1. Recommandations générales

- Les lignes des fichiers sources ne doivent pas dépassées **80 colonnes** ;
- **Deux espaces** pour l'indentation quel que soit le type de fichier (les tabulations sont interdites) ;
- Encoding : **utf-8**.

24.2. PHP

24.2.1. Principes généraux

- Coder en PHP5 ;
- Eviter les extensions ou méthodes « deprecated » (ex. utiliser `preg` plutôt que `ereg`) ;
- Un fichier ne contenant que du PHP dispose d'une balise `<?php` ouvrante mais pas d'une balise `?>` (les actions respectent toujours ce principe) ;

24.2.2. Variables

Booléen	Il est obligatoire de placer un verbe en préfixe : `can_buy`, `is_premium`, `has_right`
Tableau	Utiliser le pluriel. Ex. `$children`, `$products`, etc.
Quantité	Ajouter le suffixe : `_count`. Nom : singulier Ex. `$product_count`
Période / Intervalle	Utiliser un terme précis suivi de `_from` et `_to` Ex. `$on_sale_from`, `$on_sale_to`, `$visible_from`, `$visible_to`

24.2.3. Optimisations

Les ressources de type `Collection`, `ResultSet`, `Csv` sont préférables à des représentations de type *hash* ou tableau. Elles ont en effet l'avantage de nécessiter beaucoup moins de mémoire.

24.2.4. Nom des fichiers

L'extension est `.inc`.

Les classes doivent être suffixées de la façon suivante : `.class.inc`.

24.2.5. Standards de notation

24.2.5.1. Variables

Notation PHP : minuscules + termes séparés par des `_`.

Utiliser `product_name` plutôt que `productname`, `product-name` ou `productName`.

24.2.5.2. Méthodes

Notation Camel

```
class Request {
  public function addAttribute() {
  }
}
```

Le nom de la méthode commence par un verbe : `getTopic()` et non `topicGet()`.

Si la méthode renvoie un booléen le verbe est à mettre à la troisième personne du singulier : `hasChildren()` et non `haveChildren()`. Certains préfixes peuvent être privilégiés : `is`, `can`, `has`.

Les parenthèses sont collées au nom de la fonction/méthode.

24.2.5.3. Structures de contrôle

- Un espace est présent avant la parenthèse ouvrante.
- Un espace est présent après la parenthèse fermante.
- L'accolade ouvrante se situe sur la même ligne que la parenthèse fermante.
- L'accolade fermante est « à la ligne ».

```
if ($item instanceof ActiveRecord) {
  $datas['item'] = $item->getKey();
}
else {
  $datas['keyword'] = str_replace('"', ' ', (string) $item);
}
```

24.2.5.4. Constantes

En majuscule avec le type inclus dans le nom (en préfixe).

```
class Response {
  const FORMAT_JSON = 2;
}
```

Les constantes sont regroupées par préfixes.

```
class Request
{
  const SERVICE_XMLRPC = 1;
  const SERVICE_SOAP = 2;
  const XHR_EXEC = 1;
  const XHR_UPDATE = 2;
```

24.2.5.5. Appels

Méthode statique : aucun espace avant et après `::`

Ex. `$req::validate($url)`

Constante ou variable de classe : aucun espace avant et après `::`

Ex. `Request::FORMAT_JSON`, `Tracker::$referers`, etc.

24.2.5.6. Commentaires

Des commentaires « inline » doivent être évités, dans la mesure du possible, au profit :

- d'un code clair (nom de méthodes, de variables, de paramètres, de fichiers, etc.),
- d'une documentation annexe.

Le commentaire suivant doit être utilisé pour spécifier des zones de codes à mettre à jour :

```
/* TODO : ajouter une vérification de la valeur $n */
```

L'élément `TODO :` doit être respecté.

L'élément `WORKAROUND :` est aussi normalisé pour indiquer des lignes de codes liées à un bug du langage.

24.2.5.7. Eviter les syntaxes trop verbeuses

Il convient de ne pas surcharger les lignes de code avec des vérifications à outrance. PHP n'est pas Java et des lignes telles que ci-dessous ne font que rendre la lecture plus difficile.

```
$value = isset($h['name']) ? $h['name'] : null;
/* préférer : */
$value = $h['name'];

$value = $h['name'] ? $h['name'] : 120;
/* préférer : */
$value = $h['name'] ?: 120;
```

```
if ($req->hasAttribute('name') === true)
/* préférer : */
if ($req->hasAttribute('name'))
```

24.2.5.8. Noms de tables

L'utilisation « directe » des noms de tables au sein de requêtes SQL n'est pas autorisée.

Dans le cadre d'une table associée à un ActiveRecord on utilise la fonction `T('<MODEL>')`.

Dans les autres cas, utiliser des constantes de classe : ex. `Base::TABLE_EMAIL`.

24.2.5.9. Code

Pour une structure de contrôle la parenthèse ouvrante est placée sur la même ligne que la condition. Des espaces entourent les parenthèses.

```
if (true) {
}
```

Un espace est prévu autour des opérateurs de comparaison/assignation.

Les lignes ne doivent pas dépasser plus de 80 caractères. Le retour à la ligne au niveau du passage d'arguments est souvent un bon moyen pour y parvenir.

```
$ret = $this->dbo->insert(T('comment'),
                          array('model_id' => $this->getURI(),
                                'record_id' => $this->getId(),
                                'user_id' => (int) $user_id,
                                'content' => $content,
                                'is_staff_flag' => 1));
```

Les comparateurs stricts (`===`) doivent être privilégiés.

```
if (strpos($str, '.') === false) {
}
```

Les arguments de méthodes sont séparés par un espace. La virgule colle l'argument qui la précède.

L'opérateur `=>` des *hash* est entouré d'espaces.

Les chaînes de caractères sont entourées de simples guillemets. L'inclusion de variable provoque l'utilisation de l'opérateur de concaténation.

```
$str = 'valeur de x = ' . $req->x;
```

Une chaîne de caractères construite sur plusieurs lignes avec l'opérateur de concaténation doit être de la forme :

```
$str = 'opérateur de concaténation est placé'
       . ' à la ligne'
       . ' (espace en début de ligne)';
```

Toute référence aux variables globales de PHP est interdit : `$GLOBALS`, `$_GET`, `$_POST`, `$_SESSION`, `$_COOKIE`, `$_SERVER`, etc.

Pour des gains de mémoire il est préférable d'utiliser le chaînage de méthodes lorsqu'une variable intermédiaire n'est pas nécessaire.

```
foreach (Forum::getInstance(1)->getTopics() as $topic) {
  $res->writeln($topic->title);
}
```

La création d'objets avec `getInstance()` est préférable à l'instanciation via l'opérateur `new`.

24.2.5.10. Autres

Ne jamais utiliser `$GLOBALS`.

24.2.6. Log

Lorsque l'objet `Logger` est utilisé au sein d'une classe, le message doit contenir une référence à la classe et à la méthode d'où elle est émise. La norme est la suivante :

```
S('log')->error('registry insertion failure', __METHOD__);

try {

}
catch (Exception $e) { $log->error($e, __METHOD__); }
```

D'autres normes existent :

- une référence à un élément textuel est placée entre `[ ]`,
- une référence à un numérique est placée entre `( )`.

```
$this->log->error('display error ['.$obj->title.'] ('.$obj->getId().')', __METHOD__);
```

24.2.7. Programmation OO

24.2.7.1. Méthodes get/setAttribute()

La majorité des objets du framework dispose des méthodes `get/setAttribute()`. Il serait bon de conserver cette pratique. La méthode `getAttributes()` est également conseillée.

24.2.7.2. Méthode getInstance()

Cette méthode statique permet de retourner une instance de la classe.

```
public static function getInstance() {
  $class = __CLASS__;
  return new $class;
}
```

En proposant cette méthode, certaines syntaxes deviennent possibles. Chaîner instanciation de l'objet et appel d'une de ses méthodes en est un bon exemple.

```
Document::getIntance($arr)->index();
```

Cette méthode peut également être utilisée pour mettre en place une politique de singleton.

```
public static function getInstance() {
  if (self::$instance) {
    return self::$instance;
  }
  $class = __CLASS__;
  self::$instance = new $class;
  return self::$instance;
}
```

24.2.7.3. Fonctions statiques

KWO dispose de quelques fonctions « globales » : `l()`, `h()`, `P()`, `S()`, `M()`, `T()`.

Aucune autre fonction globale ne peut être créée. Elles doivent être remplacées par des méthodes statiques attachées à des classes.

24.2.7.4. Visibilité

La visibilité des attributs et méthodes doit toujours être précisée : `public`, `protected`, `private`.

24.2.7.5. Arguments

Ils doivent être « auto explicatifs ». Une méthode chargée d'envoyer d'un mail doit préférer `sendTo($email)` à `sendTo($s)`. Nul besoin cependant d'aller trop loin dans le détail (`$recipient_email` semble superflu).

Une méthode ne doit avoir plus de 4 arguments. Si un tel besoin se faisait sentir, l'utilisation d'un hash devrait être envisagée.

24.2.7.6. Noms de méthodes

Certaines méthodes sont présentes dans de nombreuses classes du framework. Il serait de suivre le même modèle afin de favoriser les réflexes.

`get/setAttribute()`, `getAttributes()`, `asArray()`, `asHash()`, `asSet()`, `map()`, `destroy()`.

24.2.7.7. Type hinting

Cette technique permet de demander à PHP de vérifier si un paramètre est bien un objet d'une classe donnée ou un tableau.

```
public static function comment(ActiveRecord $obj,
                               array $opts=array()) {
  $opts['model_id'] = $obj->getModelId();
  $opts['record_id'] = $obj->getId();
  return Template::render('comment:widget', $opts);
}
```

24.2.7.8. Exceptions

Le message associé à une `Exception` doit être explicite.

24.2.7.1. Attributs

Notation PHP

```
class Request {
  public $is_closed = false;
}
```

Les attributs doivent être initialisés.

24.2.7.2. Constantes / Attributs statiques

Il est essentiel de passer par des constantes pour définir des types ou des états.

```
class Campaign extends ActiveRecord
{
  const STATE_ARCHIVED = -1;
  const STATE_EDIT = 0;
  const TYPE_TEST = 0;
  const TYPE_PROD = 1;
}
```

Il est également intéressant de créer des tableaux associés décrivant ces constantes pour pouvoir y faire référence dans l'interface administration par exemple.

```
public static $statuses = array(self::STATUS_ARCHIVED => 'archivée',
                                self::STATUS_EDIT => 'rédaction',
                                self::STATUS_PENDING => 'prête',
                                self::STATUS_PROCESSING => 'envoi',
                                self::STATUS_DELIVERED => 'finalisée');
```

24.2.7.3. Méthodes statiques

Il est vivement déconseillé de passer directement par un attribut statique d'une classe. La classe en question doit fournir une méthode statique permettant de d'y accéder. La variable statique peut à tout moment disparaître et voir son contenu stocké dans une base de données.

24.3. Actions

Il est essentiel de coder une action en respectant la norme afin faciliter la vie de vos collègues qui auront à reprendre ou faire évoluer votre projet.

24.3.1. Structure Standard

```
try {
  $topic = Topic::getInstance((int) $req->id);
  $ctx->topic = $topic;
  $res->setTitle('Sujet : '.$topic->title);
  $res->useTemplate();
}
catch (Exception $e) { $err->add($e); }
```

24.3.2. Nommage

Le principe suivant doit toujours être respecté : `model.action.inc`

Tout est en minuscule pour ne pas avoir de surprise avec les URLs. Une méthode composée de plusieurs termes utilise le caractère _ comme séparateur.

Ex. `forum.widget.inc`, `topic.compose.inc`, `forum.topic.add.inc`

Deux actions s'écrivent d'une manière particulière afin d'améliorer l'url :

- visualisation : `topic.inc` plutôt que `topic.view.inc` ;
- listing : `topics.inc` plutôt que `topic.list.inc`.

Comment savoir qui est le modèle ?

Il s'agit de l'objet qui sera instancié avec le paramètre `id`.

Plus généralement une action correspond à l'exécution d'une méthode d'un modèle. La partie gauche du nom de l'action correspond donc au modèle et la partie droite à la méthode.

24.3.3. Règles essentielles à respecter

1. Une action doit éviter de convenir du SQL (il est préférable de faire appel à des méthodes à la classe `Collection`).
2. Une action place dans son contexte les données nécessaires à l'affichage du template associé. Il convient cependant de toujours avoir à l'esprit que le client pourrait demander une réponse XML (`&kof=xml`) plutôt que HTML. Cette règle permet également de faire le tri entre ce qui relève de la présentation ou du contenu.
3. L'ordre d'exécution suivant est essentiel : traitement des paramètres, chargement des modèles, opérations sur le modèle, initialisation du contexte, paramétrage de la réponse. Le tout doit être englobé dans un `try {} catch {}`.
4. Une action doit être tracée (ex. `$req->track()`) et identifiée (`$res->setHead()`).

24.4. Bases de données

24.4.1. Nom de table

Toutes les tables liées à une même extension doivent disposer du même préfixe : `shop_order`, `shop_product`, etc.

24.4.2. Colonnes obligatoires

Toute table associée à un modèle doit disposer au minimum des colonnes ci-dessous.

Nom	Fonction
`id`	Identifie chaque instance du modèle de manière unique.
`name`	Nom de l'instance. Cette colonne est avant tout utile pour une identification de l'objet en backoffice. Il est préférable de créer d'autres colonnes dans l'optique d'une utilisation au sein de templates (ex. `title`, `label`, `subject`).

24.4.3. Ordonnancement des colonnes

Plusieurs règles doivent être respectées pour construire la table associée à un ActiveRecord :

1. `id` est toujours placée en première position ;
2. `model_id` (smallint) et `record_id` (int), si présentes, sont placées derrière `id` ;
3. dans le cadre d'une table de jointure simple (2 ou 3 colonnes), les deux clefs étrangères sont placées à gauche ;
4. les autres colonnes sont classées par ordre alphabétique.

24.4.4. Ecriture des requêtes

Les éléments du langage SQL doivent être en majuscule au sein de la requête.

```
SELECT col1, col2 FROM table AS t1 JOIN table2 ON t1.col1=table2.col3 WHERE col4=3 LIMIT 3
```

Les requêtes peuvent être coupées afin de limiter la taille des lignes.

```
$ret = $this->dbo->asHash('SELECT DATE_FORMAT(created_at, "%Y-%m") AS `monthkey`,'
                          .     ' COUNT(*) AS `count`,'
                          .     ' DATE_FORMAT(created_at, "y/%Y/m/%c") AS `link`'
                          .' FROM '.T('post')
                          .' WHERE blog_id=:id: AND status=:status:'
                          .' ORDER BY `monthkey` DESC',
                          array(':id:' => $this->getId(),
                                ':status:' => Post::STATUS_OK));
```

24.4.5. 2 Colonnes spéciales

- `model_id` : l'id du modèle ;
- `record_id` : l'id d'un enregistrement pour un modèle donné.

Il est essentiel de respecter cette norme car la suppression d'un ActiveRecord entraîne des suppressions dans les tables contenant ces colonnes (cf. `$parameter['item_tables']` dans `etc/inc/platform.conf.inc`).

Une colonne `published_on` est particulièrement intéressante lorsqu'un index de date est nécessaire. Dans le cadre d'un ActiveRecord, cette colonne est automatiquement initialisée.

24.4.6. Colonnes courantes

Colonne	Type	Fonction
address		
app_id	tinyint	Nécessaire dans le cadre d'une donnée UGC enregistrée sur un site multi-app.
address_extra		
amount		discount_amount, total_amount, …
business		Secteur d'activité
caption		Sous-titre
city		
civility		
code		
completed_at	timestamp	
content		
country_id		
description		
email		
fax		
first_name		
id	int unsigned NOT NULL auto_increment	
image		Image (taille normale)
label		
latitude	double NOT NULL default '0'	
lede		chapô
level		
locale	tinyint unsigned NOT NULL default 0	
locales	int unsigned NOT NULL default 0	
longitude	double NOT NULL default '0'	
mobile		
name	varchar	
occupation		Situation professionnelle.
organization		Société, organisme
phone		

position		La position (ne pas utiliser order)
postal_code		
price		discount_price, full_price
status	tinyint	Il est courant de définir cette colonne en tant tinyint(1) lorsqu'elle s'apparente plus à un booléen. Cette colonne est prise en compte en back office dans les *finders* : couleur rouge pour la valeur 0, couleur verte pour 1.
thumbnail		Image (taille réduite)
title		
type	tinyint	
user_id	int	Fait référence à un User
website		

Le choix des noms de colonnes est essentiel tout comme le choix d'un nom de classe, d'action ou d'extension. Il conviendra de toujours essayer de s'approcher le plus possible de la norme en vigueur sur le web. Les pages ci-dessous peuvent être d'excellentes sources d'inspiration.

Un bon choix de colonne facilite :

- la relecture ;
- l'exportation (ex. Google Base) ;
- le référencement.

24.4.7. Indexation

Toute colonne prise en compte dans une recherche doit être indexée. Une colonne n'intervenant dans une recherche qu'avec une autre peut faire l'objet d'un index double :

```
UNIQUE KEY `object` (`visit_id`,`model_id`,`record_id`)
```

L'index ci-dessus ne sera utilisé que sur des requêtes faisant intervenir visit_id ou visit_id+model_id ou visit_id+model_id+record_id. Une requête n'incluant que record_id dans la condition ne ferait pas intervenir l'index.

Il est préférable de choisir le type date pour des index temporels.

Un index composé des colonnes model_id,record_id est appelé item.

24.4.8. Autres règles

- Utiliser le type DATE pour les dates disposant d'un index associé.
- Une colonne de type datetime se finit par _at ; le préfixe devient _on avec le type date.
- Toujours avoir à l'esprit ce que le nom de la colonne donnera au niveau de l'ActiveRecord.
- Une colonne ne doit jamais commencer avec le caractère « underscore » ou contenir un point.
- Les colonnes sont en minuscule.
- Une colonne ne peut commencer par property_ ou meta_.
- Les colonnes numériques doivent disposer d'une valeur par défaut.
- Le type de la colonne doit coller au plus juste au besoin (TINYINT, SMALLINT, MEDIUMINT, INT). La précision UNSIGNED est obligatoire pour des colonnes ne contenant que des valeurs positives ou nulles.

24.5. HTML

24.5.1. En-tête HTML

La génération de l'en-tête de la page HTML (<HEAD>) est réalisée par la méthode `H::head()`.[9]

```
<?=H::head()?>
<div class="wrapper">
…
```

Cette méthode génère les éléments suivants : `<doctype>`, `<head>`, et `<body>`.

```
<!DOCTYPE html>
<html lang="fr">

<head>
 <title>Test</title>
 <meta charset="utf-8" />
 <script type="text/javascript">
  var _user_id = 0, _locale = 1, _extension = "test", _scope = "front";
  var _events = [["marchandises","pages",12345]];
 </script>
 <script src="/web/core/scripts/prototype.js" type="text/javascript"></script>
 <script src="/web/core/scripts/kwo.js" type="text/javascript"></script>
 <script src="/web/social/scripts/front.js" type="text/javascript"></script>
 <script src="/web/test/scripts/master.js" type="text/javascript"></script>
 <link rel="stylesheet" href="/web/core/styles/stylesheet.css" type="text/css" media="screen" />
 <link rel="stylesheet" href="/web/test/styles/master.css" type="text/css" media="screen" />
 <link rel="stylesheet" href="/web/test/styles/front.css" type="text/css" media="screen" />
 <link rel="alternate" href="/marchandises.feed" type="application/rss+xml" title="News" />
 <link rel="icon" href="/web/test/images/favicon.png" type="image/png" />
</head>

<body class="scope-front extension-test action-marchandises">
```

La méthode charge les fichiers CSS, JavaScripts, ainsi que le sitemap, les metas (y compris open graph).

Elle peut prendre en argument un tableau permettant de préciser des scripts/styles complémentaires à charger.

```
<?=H::head(array('styles' => array('/web/<EXT>/styles/hover.css'),
                 'scripts' => array('/web/<EXT>/scripts/slider.js'))?>
```

24.5.1.1. Paramètres pris en compte

`$parameter['front.feeds']`

`$parameter['front.metas']`

9 La méthode H::foot() permet de « terminer » la page.

`$parameter['app.head.version']`	Dès que la valeur est supérieure ou égale à 1, les ressources présentes dans le head sont suffixées de la manière suivante : `<script src="/web/core/scripts/kwo.js?X"></script>` Lors de modifications importantes de CSS ou JS, l'utilisation de ce paramètre permet d'éviter des soucis liés au cache des navigateurs.
`$parameter['<SCOPE>.scripts']`	L'utilisation de ce paramètre est utile lorsque plusieurs skeletons sont utilisés pour un même scope.
`$parameter['<SCOPE>.styles']`	
`$parameter['google.analytics.account']`	

24.5.1.2. Interactions avec $res

L'objet `$res` dispose de méthodes permettant d'enrichir l'en-tête HTML de la page.

`addClass()`	$name	La classe est ajoutée à l'attribut class de <BODY>
`addFeed()`	$name, $url	
`addMeta()`	$name, $value	
`addScript()`	$path	
`addStyle()`	$path	
`setHead()`		

24.5.1.3. Chargement des scripts

En fonction du scope et de l'action, la méthode se charge d'inclure automatiquement et s'ils existent les fichiers suivants :

`/web/<EXT>/scripts/master.js`	Contient le code utilisé dans tous les scopes (front, account, middle).
`/web/<EXT>/scripts/front.js`	Code utilisé au sein des scopes front et account.
`/web/<EXT>/scripts/account.js`	Code spécifique au scope account.
`/web/<EXT>/scripts/back.js`	Code spécifique au scope back.
`/web/<EXT>/scripts/middle.js`	Code spécifique au scope middle.
`/web/<EXT>/scripts/<ACTION>.js`	Code spécifique à une action, quelque soit le scope.

Cette séparation du code facilite le travail collaboratif.

Il est recommandé de créer les objets principaux au sein du master.js.

```
kwo.test = {"marchand": {}, "marchandise": {}};
```

L'enrichissement de méthodes dans les autres scripts doit être réalisé de la manière suivante.

```
kwo.test.onComment = function () { … };
```

Les scripts `/web/core/scripts/prototype.js` et `/web/core/scripts/kwo.js` sont toujours chargés.

24.5.1.4. Chargement des styles

Fonctionnement identique aux scripts pour le chargement.

Le master.css contient l'équivalent du reset.css ainsi que toutes les règles partagées par les scopes front, middle, account.

24.5.2. Conventions

24.5.2.1. Notations

Les attributs d'une balise sont entourés de doubles guillemets.

```
<div id="box">coucou</div>
```

Bien que cela soit déconseillé, les chaînes de caractères JavaScript pour les codes *inline* sont placées entre simples guillemets.

```
<a href="javascript:void(0)" onclick="alert('coucou')">test</a>
```

La notation XHTML est de rigueur : `<br/>` plutôt `<br>`.

Tout attribut non standard doit être préfixé par `data-` .

```
<span data-item="xxx">test</span>
```

Tout attribut HTML doit disposer d'une clef et d'une valeur.

Il est par conséquent demander d'écrire `disabled="disabled"` plutôt que `disabled`.

24.5.2.2. Indentation

Tout bloc doit être indenté. Il n'est cependant pas nécessaire de respecter une indentation pour les éléments « fils » du `<body>`.

24.5.2.3. Formulaires

Ils sont transmis en AJAX en passant systématiquement par une classe intermédiaire qui fera appel à `Kwo.exec()`.

24.6. JavaScript

24.6.1. Namespace Kwo

Les scripts doivent être regroupés dans le *namespace* `kwo`

```
kwo.Topic = {
  "view": function(id) {
    Kwo.go("/topic.view", {"id": id});
  }
};
```

24.6.2. Référencement

Les pages de contenu devant être indexables par un moteur de recherche doivent être accessibles par lien `<a href="">` et non par une commande `Kwo.go()`.

24.6.3. Notation des objets

Les objets JavaScript seront écrits

```
var obj = {"id": 3};
```

et non

```
var obj = {id: 3};
```

24.6.4. Déclaration des variables

Il est essentiel de déclarer les variables locales.

```
function test() {
  var n = 2;
  for (var i = 0; i <= n; i++) {}
}
```

24.6.5. Passer par les fonctions Prototype

`document.getElementById("mon_div")`	`$("mon_div")`
`$("div").innertHTML = "hello";`	`$("div").update("hello");`
`$("input").value = "hello";`	`$("input").setValue("hello");`

24.6.6. Exécution d'un JavaScript

Deux cas sont possibles :

1. Le clic aboutit à un changement de page : il convient de placer le JavaScript au sein du `href=""` entouré de la méthode `void()`.

```
<a href="javascript:void(Kwo.Topic.view(1))">voir l'article</a>
```

2. Le clic ne réalise qu'une action JavaScript sans quitter la page : le `href` contient `void(0)` et l'action est placée dans le `onclick=""`.

```
<a href="javascript:void(0)"
   onclick="Kwo.Topic.composeComment()">ajouter un commentaire</a>
```

Il convient de ne pas oublier que `this` fera référence à l'objet uniquement dans le cas d'un `onclick`.

24.6.7. Standards de notation

Objet : utiliser la majuscule

```
kwo.Forum = {
};
```

Méthodes : notation "Camel"

```
viewTopic()
```

Utiliser les doubles guillemets plutôt que les simples.

```
var s = "bonjour" + "monde";
```

Accès aux éléments d'un tableau.

```
h["result"]["msg"] ; // plutôt que h.result.msg
```

24.7. CSS

24.7.1. Standards de notation

Le nom des class (et autres `id`) doit contenir en préfixe le nom de l'extension.

Le trait d'union est préférable au _.

Ex. `.forum-topic_entry`, `#forum-header`

Les id des éléments doivent être explicites et construit toujours de la même manière.

Ex. `order-details-box`, `cart-update-button`.

Le caractère séparateur est le `-` et non `_`.

Il est recommandé de construire l'id de la manière suivante :

1. notion associée à l'élément ;
2. type (ex. form, box, button).

Ex. `additem-button`, `comment-box`, `type-select`.

24.7.2. Id ou Class

Il est toujours préférable d'utiliser une classe plutôt qu'un id.

24.8. Templates

24.8.1. Standards de notation

L'extension d'un template est `.psp`.

La grande majorité des templates reprennent le nom de l'action associée afin de pouvoir utiliser la méthode `$res->useTemplate()`.

Un template uniquement appelé par d'autres templates (via `H::inc()`) est préfixé par `_`.

```
<?php foreach ($products as $i => $product): ?>
 <?=H::inc('shop:_product.view', $product)?>
<?php endforeach; ?>
```

24.8.2. Constantes

Les constantes listées ci-dessous sont obligatoires.

PIX_URL	/web/core/images
PUB_URL	/pub
DOC_URL	/doc
WEB_URL	/web

```
<img src="<?=PIX_URL?>/throbber.gif" />
```

24.8.3. Conventions PHP

24.8.3.1. Structures de contrôle

Les structures de contrôles doivent s'écrire avec la notation dite alternative :

```
<?php foreach ($products as $product): ?>
  <div class="shop-price"><?=$product->price?></div>
<?php endforeach; ?>
```

Un template ne contient pas de code à proprement parler. La nécessité d'écrire un bloc code implique généralement le besoin d'un *helper* à placer en tant que méthode statique des classes `W` ou `H`. Il y a en effet de fortes chances pour que d'autres projets en aient besoin. Si le besoin est extrêmement spécifique la méthode statique peut également être placée dans un ActiveRecord.

KO	OK
`Prix : <?=number_format($price, 2, ',', ' ')?>`	`Prix : <?=H::amount($price)?>`

Dans la "littérature" ces méthodes sont souvent qualifiées de *helper*.

Un espace est obligatoire entre la structure de contrôle et la parenthèse ouvrante.

```
<?php if ($products): ?>
```

24.8.3.2. Echappements de caractères / encodage

`H::url()` est préférable à une construction manuelle d'une *query string*.

```
<a href="<?=H::url('/redir', array('target' => 'http://kernix.com'))?>">GO</a>
```

Toute donnée provenant d'un utilisateur doit être échappée.

```
<div><?=h($blog->title)?></div>
```

Les codes JavaScript inline ne doivent jamais contenir de chaînes de caractères complexes. Une variable intermédiaire doit être construite afin d'y faire référence.

```
<?php
$str = "une phrase
sur plusieurs lignes
avec des \"doubles\" & 'simples' guillemets";
?>

<script>
var str = "<?=H::escapeJs($str)?>";
</script>

<a href="void(Kwo.Model.show(window.str))">attention</a>
```

24.8.3.3. Présentation

Dans la grande majorité des cas, les données stockées en base ne doivent pas contenir pas d' « informations » liées à la présentation. Dans 99% des cas, elles sont donc stockées en minuscules.

Le template associé à l'action se charge d'adapter la présentation : capitalisation, majuscules, texte en gras, présentation des dates.

```
<h1><?=c($title)?></h1>
```

24.8.3.4. Valeur de retour

Un template ne doit pas retourner une valeur. Seule la commande `return ;` peut être envisagée dans certains cas extrêmement rares (il est généralement préférable de charger des templates différents).

24.8.3.5. H::elt

L'utilisation de cette méthode est essentielle dès qu'une balise contient de nombreux attributs (et d'autant plus lorsque que la construction de cette balise fait intervenir du code PHP).

25. Installation

25.1. Création de l'arborescence

Se placer dans le répertoire web où sera hébergée l'application et y extraire l'archive.

```
> tar -zxvf archive.tgz
```

Nous supposerons désormais que le répertoire d'installation est : `/var/web/site`

Le répertoire doit alors contenir, à la racine, les fichiers suivants :

```
> ls
doc  etc  init.php  lib  pub  usr  var  web
```

Test de fonctionnement

```
> cd /var/web/site && php init.php ping
pong
```

25.2. Environnement

Serveur Web	Apache >= 2.2
Base de données	MySQL >= 5.1

PHP	PHP >= 5.3 Extensions : - calendar, - ctype, - curl, - date, - dom, - gd, - hash, - iconv, - json, - mbstring, - mysql, - mysqli, - mysqlnd, - openssl, - posix, - pcre, - session, - simplexml, - sqlite3, - tidy, - zip, - zlib

25.3. Mise en place du Virtualhost

Le `virtualhost` doit être adapté pour refléter le chemin d'installation de l'application.

Le module `RewriteEngine` doit être présent.

25.4. Base de données

Les informations de connexion doivent être renseignées dans le fichier `/var/web/site/etc/platform.conf.inc` ou dans un fichier de configuration d'une des apps.

```
$parameter['app.dsn.host'] = 'localhost';
$parameter['app.dsn.name'] = 'xxx';
$parameter['app.dsn.login'] = 'yyy';
$parameter['app.dsn.password'] = 'zzz';
```

25.5. Paramétrage de l'application

Ne pas oublier de modifier le nom de domaine ainsi que les adresses emails dans `/var/web/site/etc/app/site.conf.inc`

Tout changement d'un de ces fichiers de configuration (`etc/ext`) doit être accompagné d'une reconstruction de la configuration de l'application

```
> cd /var/web/site && php init.php build
```

Attention, le fichier `/etc/php-cli.ini` doit contenir :

```
mbstring.func_overload = 7
```

25.6. Tâche automatique

KWO nécessite le lancement d'une tâche automatique toutes les heures. Vous devez ajouter un appel à celle-ci dans le fichier `11kwo.sh` situé dans `/etc/cron.hourly`

```
#!/bin/sh

### votre nom - nom application - JJ/MM/AAAA
cd /var/web/site && php init.php scheduler
```

25.7. Droits

Certains répertoires doivent absolument avoir comme propriétaire l'utilisateur associé au serveur HTTP : `etc`, `doc` et `var`

```
> cd /var/web/site && chown -R apache:apache etc doc pub usr var
```

Les autres fichiers et répertoires peuvent appartenir à un utilisateur différent (appartenant au groupe apache), notamment pour permettre aux développeurs l'édition des fichiers via FTP.

25.8. Emplacements des fichiers

`doc`	Fichiers envoyés par les administrateurs depuis le back office.
`etc`	Configuration
`lib`	Actions et Class
`usr`	Données utilisateur
`var`	Sessions, logs, cache, fichiers d'upload
`web`	Templates (`.psp` dans `web/<EXT>/templates`), JavaScripts (`.js` dans `/web/<EXT>/scripts`), styles (`.css` dans `/web/<EXT>/styles`)

25.9. Fichier de log

Il pourrait être intéressant de vérifier de temps en temps le fichier de log de l'application.

```
> cd /var/web/site && php init.php log
```

25.10. Diagnostique

La commande CLI `diagnosis` permet de valider un certain nombre de paramètres, qu'il s'agisse de PHP, Apache, MySQL, etc.

```
> php init.php diagnosis
```

26. SECURITE

Un des objectifs de KWO est de fournir un environnement intrinsèquement sécurisé.

La seule contrainte consiste à bien utiliser les outils mis à disposition.

26.1. Base de données

- Utiliser les *bindings* pour les requêtes (`query`, `exec`),
- Utiliser les fonctions de type `insert`, `update`, `replace`.

26.2. Templates

Les chaînes de caractères doivent être échappées (cf. fonction globales `h()` ou `H::escapeJs()`).

26.3. Log

Il est essentiel de tracer à l'aide de l'objet `Logger` (`$log`) tous les évènements « étranges » (ex. non réception d'un paramètre).

Attention en revanche de ne pas « polluer » les logs avec des flots d'avertissements. Cela n'est gênant pas pendant le développement mais doit être éliminé avant la mise en production.

La consultation des logs est **obligatoire** pendant la semaine faisant suite à une mise en production.

27. OUTILS DE DEVELOPPEMENT

27.1. Unicode

Attention de paramétrer les différents outils de manière à utiliser l'encodage UTF-8 (unicode) : éditeur, client FTP.

27.2. Editeur

2 espaces pour la tabulation.

28. DEFINITIONS

Notation Camel	chaque première lettre de mot en majuscule sauf pour le premier mot.
Notation PHP	les mots sont séparés par des _.
Throbber	Image animée permettant d'indiquer qu'une activité est en cours.

29. Index

A

abus · 105
action · 15
ActiveRecord · 42
  Tracking · 82
AJAX · 29, 93
API · *Voir* webservices
application · 12
authentification · 69

B

base de données · 34
  conventions · 116
  paramétrage · 129

C

cache · 51, 71, 121
captcha · 105
ClassRecord · 44
CLI · 12
Collection · 55
  $opts · 55
  ResultSet · 38
  RSS · 65
commentaires · 106
Configuration
  /etc · 20
Context · 10, 30
Conventions
  HTML · 122
Cookie · 28, 40
Cron · *Voir*tâches automatiques
CSS
  fichiers · 21
CSV
  génération · 32

D

DataBaseObject · 34
date · 102

E

email · 61
espace
  administration · 12
  membre · 12
  partenaire · 12
  public · 11
espace d'administration · 22
espace membre · 22
exceptions
  usage · 113
Exceptions
  ActiveRecord · 49
Extension · 15, 31
  core · 19
extension projet · 13

F

fichier · 83
flux · *Voir* RSS
fonctions
  globales · 20

G

gestion d'erreur · 25

H

hook · 13, 48, 78
HTML
  construction d'élément · 100
  formulaires · 122

I

image · 85
installation · 128
internationalisation · 89, 106
item · 44

J

JavaScript
  API KWO · 93
  callback · 17, 93, 94, 96
  container · 93, 96
  fichiers · 21
  JSON · 75
  modal · 97

L

ligne de commande · 12, 27
log · 26
logs · 132
  policy · 132

M

manager · 13, 78
MetaDatas · *Voir* métadonnées
métadonnées · 50, 56, 82
modèle · 42
MVC · 11

P

pagination · 58, 60
paramètres · 13, 14, 48, 72, 73
PHP
  conventions · 108
Properties · 90

R

recherche · 48, 87
référencement · 27
répertoire · 83
Request · 27
Response · 32
  formats · 75
ResultSet · 37
routage · 14, 92
RSS · 45, 57, 65

S

scheduling
  tâches automatiques · 77
scopes · 11
sécurité · 132
  authentification · 69
Serveur SMTP · 61
Session · 41
statistiques · *Voir* tracking

T

tâches automatiques · 77
téléchargement · 39
template · 10, 14
  menu · 33
templates · 17, 21, 32
  conventions · 126
tracking · 82
  conversion · 82

U

Url Rewriting · Voir routage

V

Variables d'environnement · 77
virtualhost · 129

W

webservices · 72
  JSON-RPC · 75
  REST · 75
  SOAP · 74
  XML · 74
  XML-RPC · 72
widgets · 105

X

XML · 76