Manuel du développeur

http://www.copix.org

Table des matieres

Manuel du développeur

Outils 119

Tutoriels

Guide d'installation

Installation d'une plateforme de développement

Installer Copix est une opération relativement simple qui ne durera que **5 minutes** , téléchargement compris.

Plateforme PHP

- Une version 5.1(+) de PHP est obligatoire.
- L'extension PDO doit être activée si vous souhaitez utiliser des bases de données
- Depuis Copix 3.0.1+, il est possible d'utiliser les drivers mysql pour se connecter à une base MySql, rendant PDO facultatif pour cette base de données.

Si vous êtes sous Windows, vous pouvez installer **Wamp** ou **EasyPHP version 2+**.

Sous Linux, il existe de nombreux packages pour faciliter l'installation.

Téléchargement de Copix

Vous pouvez télécharger Copix de plusieurs façons :

- Dans l'espace **téléchargements** du site (dernière version stable),
- Via le **CVS**, qui reflète la toute dernière version de développement, en utilisant le module COPIX_3.

Installation de Copix

Copiez les sources de Copix dans votre répertoire de publication, souvent nommé "www".

Vous devez donc maintenant avoir une arborescence vous permettant d'accéder à http://localhost/copix/www/index.php depuis votre navigateur

Note: Le répertoire copix/www/ correspond au répertoire de publication Copix. Il contient les seuls fichiers qui sont nécessaires d'exposer sur votre serveur, les autres pouvant se situer dans des répertoires "privés".

Droits d'écriture (Linux seulement)

Pour fonctionner correctement, Copix a besoin de pouvoir écrire dans les répertoires copix/temp/* et copix/var/*.

Ils contiennent respectivement des données temporaires (caches, fichiers php, ...) et des données applicatives (fichiers uploadés, images miniatures, ...).

Si vous êtes sous Linux, vous pouvez arriver à cette opération en tapant les commandes :

```
chmod-R 777 Copix_3/temp
chmod-R 777 Copix_3/var
```

Ou, si vous connaissez vos utilisateurs, vous pouvez également effectuer :

```
chown -R mon_user:www-data Copix_3/temp
chown -R mon_user:www-data Copix_3/var
chmod-R 775 Copix_3/temp
chmod-R 775 Copix_3/var
```

Sur un poste personnel Windows, il n'existe généralement aucune contrainte. Sur un poste Windows Server, vous serez probablement amené à donner les droits d'écriture à l'utilisateur *IUSR_**.

Une fois que tout ceci est fait, vous pouvez allez sur votre navigateur web et taper http://localhost/copix/www/index.php (si vous avez installé copix dans DocumentRoot/copix/)

Aller plus loin

- **Configuration de développement**
- **Configuration de production**
- **Installation en hebergement**

Configuration_de_developpement

Extension XDEBUG (facultatif)

L'extension XDebug va vous permettre d'identifier les erreurs plus facilement, grâce à l'affichage des piles d'appels par exemple. Cette extension ne devra être présente _QUE_ sur votre plateforme de développement, et en aucun cas sur votre plateforme de production.

Téléchargez l'extension pour votre version de PHP sur http://www.xdebug.org.

Copiez l'extension *php_xdebug.dll* ou *php_xdebug.so* dans le répertoire des extensions PHP (par défaut php/extensions/).

Inscrivez l'extension dans le php.ini en ajoutant la ligne

```
#sous windows
extension=php_xdebug.dll
#sous linux
extension=php xdebug.so
```

De la même façon, activer l'extension de cette façon, toujours dans le fichier php.ini

```
xdebug.profiler_enable = 1;
#Sous windows:
xdebug.profiler_output_dir = C:Chemin
#sous linux
xdebug.profiler output dir = /chemin/
```

Redémarrez votre serveur web et vérifiez que l'extension est bien activée. Avec un *phpinfo()*, vous devriez voir une section XDEBUG.

Sinon, allez voir le **guide d'installation** de XDEBUG pour des informations plus détaillées.

PHPUnit

Deux solutions sont à votre disposition pour installer PHPUnit.

* La première est de **télécharger les sources** et de placer le répertoire PHPUnit le répertoire www de votre projet.

* La deuxième, la plus recommandé, est d'utiliser l'installeur Pear.

Tout d'abord ajoutez le channel Pear.phpunit.de à votre installation :

```
pear channel-discover pear.phpunit.de
```

Ceci est à faire une fois et vous permettra ensuite d'installer PHPUnit par la commande suivante

```
pear install phpunit/PHPUnit
```

Configuration_de_production

Cette page regroupe quelques recommendations pour utiliser Copix en production

Protection des répertoires

Dans la mesure du possible, il convient de placer les répertoires de l'arborescence copix autre que www dans des espaces n,on accessibles par un navigateur.

Jeux de caractères

Les scripts de Copix utilisent le jeu de caractères UTF-8. Il vous faudra donc, si ce n'est déjà fait, modifier la configuration de votre serveur en modifiant la directive AddDefaultCharset.

```
AddDefaultCharset utf-8
```

Si vous ne pouvez modifier la configuration de votre serveur, placez la directive dans un fichier .htaccess et uploadez ce fichier sur votre hébergement.

Installation_en_hebergement

Introduction

Cette page recense quelques recommandations à prendre en compte avant et au moment de l'installation de Copix chez un hebergeur.

Elle contient aussi quelques astuces concercant l'installation chez des hebergeurs précis.

Vérification des prérequis

Si vous avez lu le **Guide d'installation** vous avez pu déjà avoir un aperçu des besoins de Copix. Il convient donc de vérifier si votre hébergeur propose les options suivantes :

PHP 5.1 et +

De plus en plus d'hébergeur propose PHP 5, mais assez rarement par défaut. Différentes solutions sont proposés, certaines vous demande de changer les extensions de vos fichier en .php5. Dans ce cas renommez simplement index.php en index.php5. D'autres vous permettent de définir à l'aide d'un .htaccess, une variable d'environnement indiquant la version de php à utiliser.

Exemple chez OVH :

```
SetEnv PHP_VER 5
```

Il vous faudra consulter les documentations de votre hébergeur pour connaître les options qu'ils proposent. Si votre hébergeur ne propose que PHP 4 ou php 5.0, changez en.

SGBD et PDO

Il est important de vérifier la présence de l'extension et des drivers correspondant au SGBD que vous désirez utiliser (généralement pdo_mysql et parfois pdo_pgsql). S'ils ne sont pas présent il est toujours possible de passer par le SGBD Sqlite intégré dans PHP 5.

Répertoire de destination

Idéalement, il convient de placer copix dans un répertoire différent du répertoire de publication et de rendre accessible le répertoire www.

Cas Concrets

OVH

Mise en place de PHP 5

Pour faire cohabiter PHP4 et PHP5 sur le même hebergement mutualisé, OVH distingue les versions suivant l'extension. Il vous faudra renommer le fichier copix/www/index.php en copix/www/index.php5

Autre possibilité, mettre en place un fichier .htaccess à la racine du répertoire Copix et placer à l'intérieur la ligne suivante :

```
SetEnv PHP_VER 5
```

Tous les scripts .php se situant dans ce répertoire et ses sous-répertoire, utiliseront PHP5.

Drivers PDO

La version de PHP5 chez OVH est compilé avec les drivers pdo_sqlite et pdo_mysql. Pas de problèmes particuliers à ce niveau là. Le format de la chaine de connexion est le suivant : dbname=XXX;host=YYY où dbname est le nom de votre base de données et YYY l'hôte du serveur MySQL (ex : sql4).

Répertoire de destination

Par défaut les hébergements OVH dispose d'un répertoire www pour la publication. Il est donc possible d'uploader tel quel votre arborescence Copix. Il est même possible selon vos options, de dédier un sous-domaine à copix.

Rendez vous sur le manager OVH section hébergement, rubriques sous-domaines. Dans le formulaire indiquez les informations suivantes :

- Nom du sous-domaines : copix
- Dossier cible : copix/www

Crééz sous la racine un répertoire copix. Uploadez votre arborescence dans ce répertoire.

Free

Mise en place de PHP 5

Drivers PDO

Safe Mode

Principes de base

Objectifs

Ce document a pour but de vous présenter dans les grandes lignes comment Copix organise votre application.

Un seul point d'entrée

Comme vous l'avez sans doute constaté, Copix utilise un seul point d'entrée pour vos applications: **index.php**.

Ce fichier est le seul et unique fichier PHP que vous devez laisser accessible à l'utilisateur (dans le répertoire de publication de votre serveur web). Les autres fichiers n'ont pas à être publiés, ils seront automatiquement inclus par Copix.

L'avantage de ce système est que vous n'avez pas à vous soucier de ce qui pourrait bien se passer si un utilisateur malveillant voulait inclure directement un fichier autre, car ils ne sont tout simplement pas accessibles !

Ce point d'entrée inclus un ensemble de fichiers de configuration (notamment copix.inc.php) et instancie le **ProjectController** qui aura la charge, via sa méthode "process", de lancer l'application.

A partir de ce moment, Copix prend le relais et route la demande vers les objets capables de la prendre en charge.

Des modules

Si on vulgarisait à l'extrême le rôle de Copix, on pourrait dire qu'il tente de faire fonctionner au mieux un ensemble de modules développés sur un modèle commun.

Ce que l'on appelle un module dans Copix, c'est un ensemble de fichiers (classes, templates, actions, ressources) qui ont pour objectif de constituer un ensemble fonctionnel (un forum, un blog, un gestionnaire de brèves, un moteur de recherche, ...).

Forme des URL

Lorsque l'on effectue une demande à Copix, on le fait via un navigateur web, et donc sous la forme d'une URL.

Copix est capable de fonctionner avec plusieurs modes d'url. En standard, copix fonctionne en mode "prepend" pour générer des formes de type index.php/module/groupe/action plutot que index.php?module=module&group=groupe&action=action

Vous découvrirez plus tard qu'il vous est possible de demander à Copix de générer les URL telles que bon vous semble, mais restons sur le cas standard.

Les 3 composantes essentielles

Lorsque vous demandez une URL de la forme "index.php/admin/log/show", voyons ce qui se passe.

Les URL dans Copix sont des demandes d'Action. Elles décrivent via un trigramme les éléments suivants :

- Le module à utiliser
- L'ActionGroup à utiliser
- L'Action à exécuter

Ces paramètres sont simplement indiqués dans l'ordre d'apparition, donc dans notre cas :

- module => admin
- actiongroup => log
- action => show

Si nous avions utilisé le mode "none" dans la gestion des url, elle aurait eu cette apparence : index.php?module=admin&group=log&action=show

Note : Copix dispose de plusieurs librairies pour créer / analyser les URL pour vous, ainsi vous développez des modules compatibles avec tous les modes d'url possibles.

Et concrètement, où va t on ?

Nous avons donc cette url "index.php/admin/log/show" et nous savons à quoi elle correspond.

Maintenant, voyons concrètement ce que Copix va exécuter.

module = admin

Cela veut dire que Copix ira chercher l'ensemble des ressources dans le module "admin". Nous avons dit plus haut que Copix était capable de faire fonctionner des modules. Ici nous choisissons donc de faire fonctionner le module "admin".

Ce module est livré dans *project/modules/stable/standard/admin*

Note : Il est possible de configurer plusieurs répertoires où Copix ira chercher les modules, c'est une option de configuration (arModules) paramétrable dans **copix.conf.php**.

group = log

Cela demande à Copix d'utiliser un objet "groupe d'actions" nommé ActionGroup**Log**. On fait référence à ces objets via la désignation "les ActionGroups"

Cet ActionGroup**Log** devra être situé dans le module "admin" (car la demande concerne le module admin).

Le fichier de cet ActionGroup sera situé au chemin *admin/actiongroups/log.actiongroup.php*.

Dans ce fichier, il existera une classe nommée ActionGroupLog qui devra déclarer un ensemble d'Actions prises en charge par ce objet.

Exemple de déclaration de ce fichier :

```
/**
* Cet objet prend en charge les demandes de type module=admin&group=log ou
           ✂ index.php/admin/log
*/
class ActionGroupLog extends CopixActionGroup {
   //...Contenu de l'objet
}
```

action = show

Le dernier paramètre "action" indique à Copix l'action à exécuter dans le groupe d'action log.

Les méthodes qui correspondent à des Actions dans les ActionGroup commencent par "process".

Ici, comme l'action demandée est "show" alors la méthode exécutée sera "process**Show**"

En résumé :

```
/**
* Cet objet prend en charge les demandes de type module=admin&group=log ou
            ✂ index.php/admin/log
*/
class ActionGroupLog extends CopixActionGroup {
   //Cette méthode intercepte index.php/admin/log/show
   public function processShow (){
      //contenu
   }
}
```

Traitements de l'action

Pour réaliser les traitements à faire dans votre action, vous avez complètement la main et faites en gros tout ce que vous voulez en PHP.

La seule contrainte sera, à la fin de vos traitements, d'indiquer à Copix la suite des opérations, à savoir pour 99% des cas : Affichage ou Redirection.

Affichage

Une demande d'affichage est effectuée via un code retour nommé PPO. PPO signifie "Plain PHP Object", donc un objet PHP tout ce qu'il y a de plus simple qui contiendra des données "prêtes" à être affichées.

Voyons un exemple où nous préparons des données prêtes à être affichées.

```
class ActionGroupLog extends CopixActionGroup {
   public function processShow (){
      //ici on peut effectuer tout un tas de traitements

      $ppo = new CopixPPO ();//création d'un objet PPO vide
      $ppo->monNom = 'Copix';//ajoute une donnée monNom destinée à être affichée
      return _arPpo ($ppo, 'template_interieur.tpl');//indique le code retour PPO
   }
}
```

et le template (situé dans templates/template_interieur.tpl)

```
<p>{$ppo->monNom} c'est trop bien !!</p>
```

Redirection

```
class ActionGroupLog extends CopixActionGroup {
   public function processShow (){
      //ici on peut effectuer tout un tas de traitements
      return _arRedirect ('http://www.copix.org');//ici on demande à rediriger sur
            ✂ copix.org
   }
}
```

comprendre_la_forme_des_URL

En standard

En standard dans PHP, lorsque vous appelez un script, vous pouvez donner des paramètres de la façon suivante :

```
index.php?parametre=valeur&parametre_2=valeur_2
```

Avec Copix

Copix (qui utilise une architecture **MVC**) dispose d'un "Front Controller" appelé dans www/index.php (voir l'**arborescence**) qui sera en charge de lancer l'application. C'est ainsi ce "index.php" qui recevra l'ensemble des paramètres de la requête.

Ce fichier d'index accepte 3 paramètres standards réservés par Copix :

- module (le module à éxécuter)
- group (**l'ActionGroup** à charger qui sera nommé ActionGroup**ValeurDuParametreGroup**)
- action (L'**action** à lancer, qui aura pour nom process**ValeurDuParametreAction** dans l'ActionGroup**ValeurDuParametreGroup**)

Chacun de ces paramètres, s'il n'est pas renseigné, prend pour valeur "default".

En mode "prepend" (URL significatives)

Copix dispose d'un mode de gestion d'URL nommé "prepend". Ce mode génère des URLs de la forme

```
index.php/nom_module/nom_group/nom_action
```

L'intérêt de ce mode est de générer des URLs plus "lisibles" par vos internautes.

En mode "default" (mode normal)

```
index.php?module=nom_module&group=nom_group&action=nom_action
```

Copix permet également de gérer les URL de façon habituelle, simplement avec la forme

Voir aussi

- CopixUrl qui est la classe de manipulation des URL.
- Les gestionnaires d'URL qui vous permettent de faire de l'URL Rewriting.
- CopixRequest qui vous permet de récupérer les paramètres envoyés par le navigateur à votre script.
- Trigramme module group action pour décrire les URL

Arborescence

Arborescence principale de Copix

Lorsque vous décompressez Copix, vous trouvez les répertoires suivants :

- *var*/ par convention, tous les fichiers non temporaires manipulés par Copix sont stockés ici. Ce chemin est représenté par la constante COPIX_VAR_PATH.
- *temp*/ tous les fichiers de cache, les fichiers générés par Copix (dao, ressources, paramètres, ...). Ce chemin est représenté par la constante COPIX_TEMP_PATH.
- *project*/ Les éléments spécifiques au projet. Ce chemin correspond à la constante COPIX_PROJECT_PATH.
  - *config*/ Emplacement par défaut du fichier de configuration (copix.conf.php)
  - *modules*/ Là où vous allez travailler ! En général (ce n'est pas obligatoire), c'est ici que l'on positionne les modules.
  - project.inc.php
- *utils*/ Les fichiers nécessaires au fonctionnement de Copix (le framework lui même)
- *www*/ Les fichiers qui seront publiés sur le serveur web dans le "DocumentRoot"
  - **index.php** Le seul fichier php qui doit être accessible aux utilisateurs pour lancer Copix

Arborescence d'un module

Dans Copix, nous développons souvent des modules pour répondre aux demandes de l'utilisateur. Ces modules disposent de cette arborescence :

- nom_du_module (doit être en minuscules)
  - actiongroups les groupes d'actions, ce sont les contrôleurs de page
  - classes c'est ici que vous placerez vos objets métiers
  - resources c'est ici que vous mettrez vos fichiers i18n et la déclaration de vos DAO
  - templates tous les modèles d'affichage de vos modules sont ici
  - zones les zones sont des objets capables de prendre en charge un élément graphique indépendant.
  - module.xml le fichier de déclaration de votre module **fichier requis**

Modele MVC

MVC, c'est quoi ?

MVC est un Design Pattern qui signifie Modèle Vue Contrôleur. C'est une façon de découper votre application en couches logiques avec d'un coté les éléments qui touchent au Modèle (la persistance, en gros la base de données), à la Vue (la façon dont les données sont affichées, les templates) et enfin les Contrôleurs (gestion de la cinématique, lancement des traitements).

Comment est-ce concrétisé dans Copix ?

Vue

La vue est concrétisée par les templates.

Les templates sont des fichiers (X)HTML (le plus souvent) qui n'attendent que les données finales pour être présentés à l'utilisateur, ce sont des modèles d'affichage.

Dans Copix, ils sont stockés dans les répertoires "templates" des différents modules.

On peut écrire les template directement en PHP (dans ce cas ils sont nommés avec l'extension ".php") ou en utilisant la syntaxe Smarty (http://smarty.php.net)

Modèle

Le modèle est concrétisé le plus souvent par les **DAO**. Les DAO sont des classes que Copix génère automatiquement pour dialoguer avec la base de données avec une syntaxe objet simple et facile à utiliser.

Dans certains cas ou les requêtes sont très complexes et multi-tables, il vous est possible de créer des objets métiers qui dialogueront directement avec la base de données via **CopixDB**.

Ces classes seront stockées dans les répertoires classes/ des modules.

Contrôleur

Les contrôleurs dans Copix sont implémentés par des objets **ActionGroup**.

Ces ActionGroup regroupent l'implémentation **d'Actions** de même nature. On appelle Action une réponse à une requête (URL).

Les ActionGroup sont stockés dans les répertoires actiongroups/ des modules.

Actions

Avant de commencer

Nous vous conseillons de lire le tutoriel **Hello You !** ainsi que **Principes de base** avant de commencer la lecture de ce document.

Qu'est-ce qu'une Action ?

Les actions dans Copix sont des réponses à une URL. A une demande correspond une Action.

Les actions sont implémentées dans des objets **ActionGroup**.

Exemple :

```
class ActionGroupExemple extends CopixActionGroup {
   //L'action par défaut
   public function processDefault (){
   }
   //Action "UneAction"
   public function processUneAction (){
   }
   //Action "UneAutreAction"
   public function processUneAutreAction (){
   }
   //Action "UneTroisiemeAction"
   public function processUneTroisiemeAction (){
   }
}
```

Vous êtes libre de faire ce que bon vous semble dans les actions et devez à la fin de chacune retourner une réponse de type "CopixActionReturn" pour indiquer à Copix la façon dont terminer le processus (affichage, téléchargement, redirection, ...)

Les différents codes retour

Affichage avec CopixActionReturn::PPO

Le type PPO est utilisé pour afficher un contenu (souvent (X)HTML).

```
public function processAffiche (){
   $ppo = new CopixPpo ();
   $ppo->TITLE_PAGE = 'Titre de la page' ;
   $ppo->nom = 'Le nom à afficher';
   return new CopixActionReturn (CopixActionReturn::PPO, $ppo,
            ✂ 'template_a_utiliser.tpl');
   //ou alors, plus rapide à écrire
   return _arPpo ($ppo, 'template_a_utiliser.tpl');
}
```

Tout sur **CopixActionReturn::PPO**

Redirection avec CopixActionReturn::REDIRECT

Le type REDIRECT est utilisé pour rediriger l'utilisateur vers une autre URL.

```
public function processAffiche (){
   return new CopixActionReturn (CopixActionReturn::REDIRECT, 'http://www.yahoo.fr');
   //ou alors, plus rapide à écrire
   return _arRedirect ('http://www.yahoo.fr');
}
```

Tout sur CopixActionReturn::REDIRECT.

Téléchargement d'un fichier avec CopixActionReturn::FILE

Le type FILE est utilisé pour présenter à l'utilisateur un fichier existant sur le serveur.

```
public function processDownload (){
   return new CopixActionReturn (CopixActionReturn::FILE,
            ✂ '/tmp/fichier_a_telecharger.pdf');
   //ou alors, plus rapide à écrire
   return _arFile ('/tmp/fichier_a_telecharger.pdf');
}
```

Tout sur CopixActionReturn::FILE.

Téléchargement d'un contenu avec CopixActionReturn::CONTENT

Le type CONTENT est utilisé pour présenter à l'utilisateur un contenu binaire. On s'en sert lorsque le contenu est généré à la volée et n'est pas conservé sur le serveur (une image, un pdf, ...).

```
public function processDownloadContent (){
   $image = new UneClasseQuiGenereUneImage ();
   return new CopixActionReturn (CopixActionReturn::CONTENT, $image->generateJpg ());
   //ou alors, plus rapide à écrire
   return _arContent ($image->generateJpg ());
}
```

Tout sur CopixActionReturn::CONTENT.

Ne rien faire de plus avec CopixActionReturn::NONE

Le type NONE est rarement utilisé et est réservé pour des occasions très spécifiques ou on demande à Copix de ne rien faire après l'action elle-même (on considère ainsi que l'action a effectué elle-même toutes les opérations nécessaires à la réponse de l'utilisateur).

```
public function processDownloadContent (){
   //des traitements divers
   return new CopixActionReturn (CopixActionReturn::NONE);//rien d'autre à faire, ni
            ✂ affichage ni rien
   //ou alors, plus rapide à écrire
   return _arNone ();
}
```

Retourner un code HTTP avec CopixActionReturn::HTTPCODE

Le type HTTPCODE est également rarement utilisé, il permet de retourner un simple entête HTTP comme retour au navigateur client.

```
public function processNotFound (){
   return new CopixActionReturn (CopixActionReturn::HTTPCODE, CopixHTTPHeader::get404
            ✂ (), "Page introuvable");
}
```

ActionGroup

Introduction

Les objets "ActionGroup" ont pour objectif de regrouper les actions par typologie (actions d'administration, actions front office, ...)

Ces objets sont stockés dans les répertoires actiongroups/ des modules.

Les fichiers qui déclarent les ActionGroup sont en minuscules et se nomment "nom.actiongroup.php".

Dans un fichier "nom.actiongroup.php" on trouvera la déclaration de la classe "ActionGroupNom"

exemple

```
class ActionGroupNom extends CopixActionGroup {
}
```

CopixActionGroup est la classe de base des ActionGroup.

Les actions

Le rôle principal des ActionGroup est de répondre à la demande de l'internaute, demande qui arrive sous la forme d'une URL.

Les **Actions** sont implémentées dans les ActionGroups par des méthodes préfixées de "process".

```
class ActionGroupExemple extends CopixActionGroup {
   function processUneAction (){
      //..code d'une action
   }
}
```

Routage par défaut

Les *ActionGroup* sont sollicités par un processus de routage qu'il est important de bien comprendre pour la suite.

Pour rappel, les **URL Copix** (générées et traitées avec **CopixUrl**) sont composées de 3 éléments :

- Le module (qui va déterminer le "chemin d'exécution")
- Le groupe (qui va déterminer l'*ActionGroup* à utiliser)
- L'action (qui va déterminer la méthode à appeler dans l'ActionGroup)

Exemple 1

index.php?module=mon_module&group=mon_groupe&action=mon_action

Ici, on va instancier l'*ActionGroup ActionGroupMon_Groupe* situé dans le module *mon_module* (emplacement exact : *chemin_du_module/mon_module/actiongroup/mon_groupe.actiongroup.php*) et exécuter la méthode *processMon_Action ()*

Exemple 2

index.php?module=mon_module&group=mon_groupe&action=mon_autre_action

Ici, on va instancier l'*ActionGroup ActionGroupMon_Groupe* situé dans le module *mon_module* (emplacement exact : *chemin_du_module/mon_module/actiongroup/mon_groupe.actiongroup.php*) et exécuter la méthode *processMon_Autre_Action ()*

Exemple 3

index.php?module=mon_module&group=mon_autre_groupe&action=mon_action

Ici, on va instancier l'*ActionGroup ActionGroupMon_Autre_Groupe* situé dans le module *mon_module* (emplacement exact : *chemin_du_module/mon_module/actiongroup/mon_autre_groupe.actiongroup.php*) et exécuter la méthode *processMon_Autre_Action ()*

Note : La casse des modules, groupes et actions n'a pas d'importance, Copix utilisera systématiquement des minuscules.

Les méthodes spéciales

beforeAction ($actionName)

La méthode beforeAction est systématiquement appelée avant le traitement à proprement parlé de votre méthode "process". Elle reçoit en paramètre le nom de l'action demandée par l'utilisateur.

Cette méthode est donc une bonne opportunité pour réaliser des opérations groupées pour votre *ActionGroup*.

Par exemple, si vous voulez protéger l'ensemble des pages de votre ActionGroup pour qu'elles soient accessibles aux seuls administrateurs, il suffit de faire :

```
class ActionGroupProtege extends CopixActionGroup {
   public function beforeAction ($pActionName) {
      CopixAuth::getCurrentUser ()->assertCredential ('basic:registered');
   }
}
```

Note: Si vous levez une exception dans votre méthode beforeAction, la méthode "process" (de l'action demandée) ne sera pas exécutée. (Si vous implémentez catchActionErrors, cette méthode aura la main)

Si vous décidez de retourner un CopixActionReturn depuis cette action, alors l'action initialement demandée ne sera pas exécutée et Copix prendra directement en compte votre retour.

Exemple :

```
class ActionGroupDefault extends CopixActionGroup {
   public function beforeAction ($pActionName) {
      if (in_array ($pActionName, array ('action1', 'action2', 'action3'))){
         //Si l'action demandée est 1 2 ou 3, alors on redirige l'utilisateur vers une
            ✂ autre URL
         return _arRedirect (_url ('module|group|action'));
      }
   }
}
```

NOTE : Si votre méthode beforeAction a retourné un code, alors l'action d'origine n'est pas exécutée. Le processus continue ensuite normalement avec afterAction qui reçoit en paramètre "beforeAction" comme nom d'action et le code retour de beforeAction comme données.

afterAction ($actionName, $toReturn)

La méthode afterAction est appelée systématiquement après l'appel à votre méthode process.

Elle reçoit en paramètre le nom de l'action dont on a demandé l'exécution ($actionName) et le retour apporté par la méthode de l'action (processXXX).

Note: Si une exception est levée dans beforeAction ou dans votre Action, la méthode afterAction ne sera pas exécutée.

Si votre méthode afterAction retourne un CopixActionReturn, alors ce dernier sera préféré à celui émis à l'origine par l'Action.

Exemple

```
class ActionGroupDefault extends CopixActionGroup {
   public function afterAction ($pActionName, $pActionReturn) {
      //On va transformer la sortie en PDF
      if ($pActionReturn->code == CopixActionReturn::PPO){
         $tpl = new CopixTpl ();

         $tpl->assign ('ppo', $pActionReturn->data);
         $contenu = $tpl->fetch ($pActionReturn->more);
         $contenu = _class ('moduleconversion|pdf')->renderPdf ($contenu);

         return _arContent ($contenu, array ('filename'=>'archive.zip'));
      }
   }
}
```

catchActionExceptions ($e)

Lorsqu'une exception est levée dans beforeAction, afterAction ou votre Action, cette dernière est transmise à la méthode catchActionExceptions.

Cette méthode vous permet de traiter les erreurs spécifiques à votre module et d'y réagir convenablement.

Si votre méthode ne gère pas l'exception en question, il faudra la relancer avec "throw $e".

exemple

```
function catchActionExceptions ($e){
   if ($e instanceof ExceptionQueJeGere){
      //traitement
   }else{
      throw $e;//exception non gérée ici
   }
}
```

otherAction ($actionName)

Lorsque l'internaute demande une action à votre ActionGroup alors que celui-ci ne sait pas la gérer, c'est cette méthode qui intercepte la demande. La méthode otherAction reçoit le nom de l'action qui avait été demandée par l'utilisateur.

Par défaut, si vous n'implémentez pas cette méthode dans votre ActionGroup, Copix retournera un 404.

Demander l'exécution d'une Action via la méthode process

Il vous est possible de demander à Copix d'exécuter une action via la méthode statique CopixActionGroup::process ('nomAction', $parametresAction);

Vous ne devriez réserver cette possibilité qu'à des fins exceptionnelles, lorsque vous avez besoin de faire appel directement à une action tierce. Un bon exemple d'utilisation est le module **genericTools** avec par exemple son **message de confirmation**.

La description de l'action à utiliser sera de la forme : nomModule|nomGroupe::nomAction

Exemple d'utilisation

```
//... des conditions nous amènent à demander confirmation d'un message
   return CopixActionGroup::process ('genericTools|Messages::getConfirm',
                                     array ('message'=>'Êtes vous sûr de cela ?',
                                            'confirm'=>_url ('module|actionOui'),
                                            'cancel'=>_url ('module|actionNon')));
```

Voir aussi

- Le tutorial **Hello You !**
- les **Principes de base**

CopixActionReturn

Présentation

La classe CopixActionReturn représente le retour d'une **action** implémentée dans un **ActionGroup**.

Cette classe va vous permettre d'indiquer si vous souhaitez faire un **affichage classique**, un téléchargement de **fichier** ou de **contenu** ou une **redirection**.

Veuillez vous référer à la documentation des **action** pour des exemples concrets.

CopixRequest

Présentation

CopixRequest est la classe de base qui vous permet de récupérer les données en provenance des éléments $_GET, $_POST ou $_FILES du navigateur.

Cette classe est essentiellement appelée depuis vos **actions** et contient quelques méthodes pratiques pour récupérer la demande utilisateur.

Méthodes disponibles

get ($pVarName, $pDefaultValue = null, $pDefaultIfEmpty = true)

La méthode get vous permet de récupérer une variable envoyée par le navigateur. Vous devez simplement spécifier le nom de la variable à récupérer. Si le navigateur n'a pas envoyé la variable en question, une valeur par défaut ($pDefaultValue) sera retournée (null si non spécifiée). Le troisième paramètre optionnel indique si vous voulez qu'en cas de valeur vide ('') Copix retourne également la valeur par défaut que vous avez spécifié.

Exemple

```
//avec l'url suivante
//index.php?parametre=valeur&parametre2=

$value = CopixRequest::get ('parametre');//on récupère valeur
$value = CopixRequest::get ('parametre2');//on récupère null
$value = CopixRequest::get ('parametre2', 1);//on récupère 1 car parametre2 est vide
et
            ✂  la valeur par défaut que l'on demande dans ce cas est 1
$value = CopixRequest::get ('parametre2', 1, false);//on récupère "" (chaine vide) car
            ✂  on indique que l'on souhaite récupérer vide en cas de valeur vide
```

On peut également utiliser la fonction raccourcis _request.

Exemple

```
$value = _request ('parametre');//on récupère valeur
$value = _request ('parametre2');//on récupère null
$value = _request ('parametre2', 1);//on récupère 1 car parametre2 est vide et la
            ✂ valeur par défaut que l'on demande dans ce cas est 1
$value = _request ('parametre2', 1, false);//on récupère "" (chaine vide) car on
            ✂ indique que l'on souhaite récupérer vide en cas de valeur vide
```

assert

Cette méthode vous permet de vous assurer qu'un ensemble de paramètres a bien été envoyé à votre application. Dans le cas contraire, la méthode assert lance une exception de type CopixRequestException.

Exemple

```
//avec l'url suivante
//index.php?parametre=valeur&parametre2=
CopixRequest::assert ('parametre', 'parametre2'); //ne lance aucune exception car les
            ✂ paramètres sont renseignés

CopixRequest::assert ('parametrePasDonne'); //lance une exception
```

exists ($varName)

Cette méthode s'assure que l'on a bien donné le paramètre de nom donné.

Exemple

```
//avec l'url suivante
//index.php?parametre=valeur&parametre2=
CopixRequest::exists ('parametre'); //true
CopixRequest::exists ('parametre2'); //true
CopixRequest::exists ('parametrePasDonne'); //false
```

getNumeric ($pVarName, $pDefaultValue = null)

Cette méthode est l'équivalent de "get" à la seule différence qu'elle s'assure de retourner une chaîne qui ne contient que des caractères numériques.

Exemple

```
//index.php?parametre=123&parametre2=abcd1234&parametreVirgule=10.23
//&parametreInvalide=abcde&parametreVide=

$valeur = CopixRequest::getNumeric ('parametre');//retourne 123
$valeur = CopixRequest::getNumeric ('parametre2');//retourne 1234
$valeur = CopixRequest::getNumeric ('parametre3', 0);//retourne 0
$valeur = CopixRequest::getNumeric ('parametre4', 'valeur nulle');//retourne la chaîne
          ✂ "valeur nulle"
$valeur = CopixRequest::getNumeric ('parametreVirgule'); //retourne 1023 (on ne prend
          ✂ pas les virgules, pour les virgules utilisez getFloat)
$valeur = CopixRequest::getNumeric ('parametreInvalide');//retourne '' (chaîne vide)
$valeur = CopixRequest::getNumeric ('parametreVide', 17); //retourne 17
```

Note : Il est possible de spécifier une valeur par défaut qui n'est pas numérique.

getFloat ($pVarName, $pDefaultValue = null)

Cette méthode est l'équivalent de "get" à la seule différence qu'elle s'assure de retourner une valeur décimale.

Exemple

```
//index.php?parametre=123&parametre2=abcd1234&parametreVirgule=10.23
//&parametreVirguleAuMilieu=1a.2z3&parametreInvalide=abcde&parametreVide=

$valeur = CopixRequest::getFloat ('parametre');//retourne 123.0
$valeur = CopixRequest::getFloat ('parametre2');//retourne 1234.0
$valeur = CopixRequest::getFloat ('parametre3', 0);//retourne 0.0
$valeur = CopixRequest::getFloat ('parametre4', 'valeur nulle');//retourne la chaîne
          ✂ "valeur nulle"
$valeur = CopixRequest::getFloat ('parametreVirgule'); //retourne 10.23
$valeur = CopixRequest::getFloat ('parametreVirguleAuMilieu');//retourne 1.23
$valeur = CopixRequest::getFloat ('parametreInvalide');//retourne 0.0
$valeur = CopixRequest::getFloat ('parametreVide', 1); //retourne 1
```

Note : Il est possible de spécifier une valeur par défaut qui n'est pas un flottant.

getInt ($pVarName, $pDefaultValue = null)

Cette méthode est l'équivalent de "get" à la seule différence qu'elle s'assure de retourner une valeur entière (int). Exemple

```
//index.php?parametre=123&parametre2=abcd1234&parametreVirgule=10.23
//&parametreInvalide=abcde&parametreVide=

$valeur = CopixRequest::getInt ('parametre');//retourne 123
$valeur = CopixRequest::getInt ('parametre2');//retourne 1234
$valeur = CopixRequest::getInt ('parametre3', 0);//retourne 0
$valeur = CopixRequest::getInt ('parametre4', 'valeur nulle');//retourne la chaîne
          ✂ "valeur nulle"
$valeur = CopixRequest::getInt ('parametreVirgule'); //retourne 1023 (on ne prend pas
          ✂ les virgules, pour les virgules utilisez getFloat)
$valeur = CopixRequest::getInt ('parametreInvalide');//retourne 0
$valeur = CopixRequest::getInt ('parametreVide', 17); //retourne 17
```

Note : Il est possible de spécifier une valeur par défaut qui n'est pas un entier.

getAlpha ($pVarName, $pDefaultValue = '')

Cette méthode est l'équivalent de "get" à la seule différence qu'elle s'assure de retourner une chaîne qui ne contient que des caractères alphabétiques (espaces compris).

Exemple

```
//Avec l'url index.php?parametre=alpha&parametre2=alpha1&parametre3=alpha beta
//&parametreVide=&parametreInvalide=12345

$valeur = CopixRequest::getAlpha ('parametre'); //retourne la chaîne alpha
$valeur = CopixRequest::getAlpha ('parametre2'); //retourne la chaîne alpha
$valeur = CopixRequest::getAlpha ('parametre3'); //retourne la chaîne 'alpha beta'
$valeur = CopixRequest::getAlpha ('parametreVide', 'default');//retourne la chaîne
          ✂ "default"
$valeur = CopixRequest::getAlpha ('parametreInvalide');//retourne la chaîne vide ''
```

getAlphaNum ($pVarName, $pDefaultValue = '')

Cette méthode est l'équivalent de "get" à la seule différence qu'elle s'assure de retourner une chaîne qui ne contient que des caractères alphabétiques et numériques (espaces compris).

```
//Avec l'url index.php?parametre=alpha&parametre2=alpha1&parametre3=alpha1 beta2
//&parametreVide=&parametreInvalide=@)]#

$valeur = CopixRequest::getAlphaNum ('parametre'); //retourne la chaîne alpha
$valeur = CopixRequest::getAlphaNum ('parametre2'); //retourne la chaîne alpha1
$valeur = CopixRequest::getAlphaNum ('parametre3'); //retourne la chaîne 'alpha1
beta2'
$valeur = CopixRequest::getAlphaNum ('parametreVide', 'default');//retourne la chaîne
          ✂ "default"
$valeur = CopixRequest::getAlphaNum ('parametreInvalide');//retourne la chaîne vide ''
```

getInArray ($pVarName, $pMustBeInArray, $pDefaultValue = null)

Cette méthode est l'équivalent de "get" à la seule différence qu'elle s'assure avant de retourner la valeur du paramètre que ce dernier fait partie d'une liste de valeurs pré-établie (dans un tableau).

Exemple

```
//Avec l'url index.php?parametre=alpha&parametre2=trois
$valeur = CopixRequest::getInArray ('parametre', array ('alpha', 'beta', 'gamma'));
//retourne la chaîne alpha
$valeur = CopixRequest::getInArray ('parametre2', array ('alpha', 'beta', 'gamma'));
//retourne null
$valeur = CopixRequest::getInArray ('parametre2', array ('alpha', 'beta', 'gamma'),
          ✂ 'alpha');//retourne alpha
```

asArray ()

La méthode asArray retourne l'ensemble des paramètres reçus (en $_GET et $_POST) sous la forme d'un tableau associatif.

Exemple

```
//Avec l'url index.php?parametre=alpha&parametre2=trois
$valeur = CopixRequest::asArray (); //retourne array ('parametre'=>'alpha',
           ✂ 'parametre2'=>'trois')
```

setRequest ($pArray)

Cette méthode est l'inverse de la méthode asArray: Elle initialise la requête à partir d'un tableau associatif.

Attention Toutes les anciennes valeurs de la requête sont supprimées.

Exemple

```
CopixRequest::setRequest (array ('parametre'=>'alpha', 'parametre2'=>'trois'));
CopixRequest::get ('parametre');//on récupère 'alpha'
CopixRequest::get ('parametre2');//on récupère 'trois'
```

set ($pVarName, $pValue)

Cette méthode permet d'ajouter un paramètre dans la requête.

Exemple

```
CopixRequest::set ('parametre', 'valeur');
CopixRequest::get ('parametre');//on récupère 'valeur'
```

getFile ($pVarName, $pPath=null, $pFileName=null)

Cette méthode permet de récupérer facilement un fichier uploadé via un formulaire.

- $pVarName correspond à l'attribut name du champ file du formulaire.
- Si $pPath est spécifié, Copix tente de copier le fichier reçu vers le chemin indiqué.
- Si $pFileName est spécifié, Copix tente de copier le fichier reçu vers $pPath avec le nom de fichier $pFileName (au lieu d'utiliser le nom indiqué lors de l'upload)

Dans tous les cas, getFile retourne un objet de type **CopixUploadedFile** si le fichier à été correctement uploadé et false dans le cas contraire.

Exemple

```
if (CopixRequest::getFile ('fichier', $cheminDestination) !== false){
      //la copie s'est bien passée
}
```

Bases de données

CopixDB

Introduction

CopixDB est la couche d'accès aux données de Copix.

Les profils de connexions sont déclarés via l'interface dédiée à cet effet dans le module d'administration des connexions.

Requêtes simples

```
$ctProfilParDefaut = CopixDB::getConnection ();
$arResultats = $ctProfilParDefaut->doQuery ('select * from test');
```

$arResultats contient la liste des enregistrements trouvés

```
$ctProfilParDefaut = CopixDB::getConnection ();
$cptAffectedRows = $ctProfilParDefaut->doQuery ('delete from test');
```

$cptAffectedRows contient le nombre d'enregistrements affectés par la requête

Requêtes paramétrées

```
$ctProfilParDefaut = CopixDB::getConnection ();
$arResultats = $ctProfilParDefaut->doQuery ('select * from test where something =
            ✂ :someValue', array (':someValue'=>$someValue));
```

$arResultats contient la liste des enregistrements trouvés. $someValue est la valeur non traitée de ce que l'on souhaite mettre dans le champ en question (sans quote pour les chaines par exemple)

```
$ctProfilParDefaut = CopixDB::getConnection ();
$cptAffectedRows = $ctProfilParDefaut->doQuery ('delete from test where ID = :id',
            ✂ array (':id'=>$id));
```

$cptAffectedRows contient le nombre d'enregistrements affectés par la requête $id est la valeure non traitée de ce que l'on souhaite contrôler comme valeur

Requêtes dans un contexte transactionnel

```
CopixDB::begin ();
try {
   $ct  = CopixDB::getConnection ('mysql');
   $ct2 = CopixDB::getConnection ('oracle');
   $ct->doQuery ('divers')
   $ct2->doQuery ('divers');
   CopixDB::commit ();
}catch (Exception $e){
   //Les DEUX sources de données sont rollbackées.
   CopixDB::rollBack ();
}
```

Note sur les transactions : Pour des raisons techniques, les connexions persistantes ne pourront accueillir plus d'une transaction simultanée. De ce fait, si une nouvelle transaction est demandée alors qu'une transaction est déjà en cours dans une connexion persistante, une nouvelle connexion (non persistante) sera demandée au serveur.

Copix propose un mode de gestion transparent pour vos transactions via **CopixServices**.

CopixDBException

Toutes les manipulations de base de données retournent des **CopixDBException**[1] en cas d'échec.

Ecriture plus rapide

Il existe une **fonction raccourcis** pour la syntaxe CopixDB::getConnection ()->doQuery () qui est _doQuery ().

Exemple

```
$results = _doQuery ('select * from myTable where id=:id', array (':id'=>1));
```

CopixDAO

Introduction

Les DAO dans Copix sont un moyen pour dialoguer avec la base de données rapidement et sans effort. En effet, Copix va générer pour vous toutes les requêtes SQL classiques sur une table donnée dans un objet prêt à l'emploi.

Vous pouvez demander à Copix un DAO soit à partir d'un **fichier XML**, soit simplement à partir du nom d'une table.

Au besoin, vous pourrez **écrire des méthodes spécifiques** en PHP qui feront partie de l'objet CopixDAO.

---

[1] **CopixDBException** est l'exception qui est lancée par la couche d'accès aux données lorsqu'il survient une erreur. Ce type d'exception peut être levé par exemple pour :

- Une erreur de connexion
- Une erreur de requête
- Une erreur de transaction

Un exemple concret et complet de l'utilisation des DAO :

```
$enregistrement = _dao ('maTable')->get ('clef_primaire');
```

Structure

Dans Copix, nous avons implémenté cet élément sous une forme simple que nous avons voulue efficace :

- Une fabrique (CopixDAOFactory) pour instancier les DAO. Cette fabrique dispose d'un alias (_dao () ou _ioDAO ())
- Les "DAO" qui implémentent les dialogues avec la base (insertion, modification, récupération, recherche, ...)
- Les "DAORecords" qui sont des représentations objets des enregistrements.

Méthodes standard

Les DAO disposent, en standard des méthodes suivantes :

- get pour récupérer un enregistrement à partir de sa clef primaire
- insert pour insérer un enregistrement
- update pour mettre un enregistrement à jour
- delete pour supprimer un enregistrement à partir de sa clef primaire
- findAll pour récupérer le contenu complet de la table
- findBy pour récupérer des données, selon certains critères de sélection ou de tri
- countBy pour compter des données, selon certains critères de sélection
- deleteBy pour supprimer des données, selon certains critères de sélection
- check pour vérifier la validité de l'enregistrement

Méthodes spécifiques

Il est également possible de déclarer, voir de développer facilement d'autres méthodes, soit dans :

- le fichier de définition XML
- un fichier PHP de définition de classe

Un exemple complet

Le fichier de définition XML

Supposons que nous ayons une table d'actualités appelée *news* où chaque actualité puisse appartenir soit à aucun soit à un auteur :

```
<?xml version="1.0" encoding="UTF-8" ?>
<daodefinition>
   <datasource>
      <tables>
         <table name="news" primary="yes" />
         <table name="authors" join="left" />
      </tables>
   </datasource>

 <properties>
  <property name="id_news"
         captioni18n="dao.news.fields.id_news"
         pk="true"
         required="yes"
         type="autoincrement"
         sequence="SEQ_news"
         />
  <property name="title_news"
         captioni18n="dao.news.fields.title_news"
         type="string"
         maxlength="50"
         required="yes"
         />

  <property name="summary_news"
         captioni18n="dao.news.fields.summary_news"
         type="string"
         maxlength="255"
         required="yes"
         />

  <property name="content_news"
         captioni18n="dao.news.fields.content_news"
         type="string"
         required="yes"
         />

  <property name="datewished_news"
         captioni18n="dao.news.fields.datewished_news"
         type="int"
         required="yes"
         />

  <property name="status_news"
         captioni18n="dao.news.fields.status_news"
         type="int"
         required="no"
         />
```

```
   <property name="id_autor"
         captioni18n="dao.news.fields.id_autor"
         type="int"
         fkTable="authors"
         fkFieldName="id_autor"
         />
   <property name="author_name"
         captioni18n="authors|dao.autors.autor_name"
         type="string"
         />
   <property name="author_email"
         captioni18n="authors|dao.autors.autor_email"
         type="string"
         maxlength="255"
         />
 </properties>
</daodefinition>
```

La définition de méthodes supplémentaires en XML

Nous souhaiterions récupérer toutes les actualités d'un auteur ; donc une méthode findAllByAutor :

```
<?xml version="1.0" encoding="UTF-8" ?>
<daodefinition>
   <datasource>
        ....
   </datasource>

 <properties>
    ....
 </properties>

 <methods>

   <method name="findAllByAutor" type="select">

         <parameters>
            <parameter name="id_author" />
         </parameters>

         <conditions>
            <condition property="id_author" operator="=" value="$id_author" />
         </conditions>

         <order>
            <orderitem property="title_news" way="asc" />
         </order>

    </method>
 </methods>

</daodefinition>
```

La définition de méthodes supplémentaires en php

Nous souhaiterions envoyer un mail à l'auteur lorsqu'on supprime une actualité lui appartenant et disposer d'un compteur d'actualité par auteur.

On crée un fichier *news.dao.php* dans le répertoire *classes* du module news :

```php
<?php

class DAONews {

  function delete (id_news){

    $news = $this->get($id_news);

     // Si l'auteur dispose d'une adresse de messagerie
     if ($news->autor_email != "") {
         $from = "contact@copix.org";
         $fromName = "contact@copix.org";

         $subject = "Suppression d'actualité"
         $message = "Une actualité a étée supprimée";

         $eMail = new CopixHtmlEMail ($to, "", "", $subject, $message);
         $eMail->send ($from, $fromName);

     }

     // je supprime l'objet normalement
     $this->_compiled_delete(id_news);
  }

   function nbNews(id_autor) {
      // La requête sur la base
      $query = 'select count(ID_NEWS) as count_news from news where
            ✂ id_autor='.intval ($id_autor);

      // On récupère Les résultats
      $arResults = CopixDB::getConnection()->doQuery($query);


      // On récupère la valeur à retourner
      $toReturn = $arResults[0]->count_news;

      return $toReturn;
   }
```

Utilisation de la DAO

```
// On récupère toutes les actualités
   $arr_news = _dao ('News')->findAll();

   // On boucle sur toutes les actualités
   foreach ($arr_news as $news) {
     // On supprime toutes actualités qui n'ont pas d'auteur
     if (is_null($news->id_autor)) {
       _dao ('News')->delete(news->id_news);
     } else {
       $nbNews = _dao ('News')->nbNews (news->id_autors);
       echo($news->autor_name." dispose de $nbNews actualités.");
     }
   }
```

CopixServices

Introduction

CopixServices est la classe de base pour l'implémentation des services sous Copix.

Lorsque vous utilisez un service, vous êtes assurés que tout s'exécute dans un contexte transactionnel (et donc que votre base n'est modifiée que si les opérations sont réussies).

Exemple d'utilisation

```
try {
   //demande de publication de la nouvelle d'identifiant 4
   $results = CopixServices::process ('module|News::publish', array ('id_news'=>4));
}catch (exception $e){
   //CopixServices aura fait un rollback pour nous car l'opération a échouée.
   //Il ne nous reste plus qu'a indiquer à l'utilisateur que ce n'est pas possible.
}
```

Exemple d'implémentation d'un service

Dans le fichier *cheminModuleNews/news/classes/news.services.php*

```
class ServicesNews extends CopixServices {
   private function publish (){
      //On trace dans la table d'historique les modifications
      $recordHistorique = CopixDAOFactory::createRecord ('NewsHistorique');
      $daoHistorique    = CopixDAOFactory::create ('DaoHistorique');
      $recordHistorique->date_hist = date ('Ymd');
      $daoHistorique->insert ($recordHistorique);

      $daoNews = CopixDAOFactory::create ('News');
      if (($newsRecord = $daoNews->get ()) !== null){
         $newsRecord->status_news = NEWS::Published ();
         $daoNews->update ($newsRecord);
      }else{
          throw new ApplicationException ("La nouvelle à publier n'existe pas");
      }
   }
}
```

Dans cet exemple, si nous arrivons dans le cas ou la nouvelle n'existe pas (on lance l'exception), Copix annule pour vous les insertions qui ont déjà été réalisées via le $daoHistorique.

En bref, si jamais une exception est levée par votre service, toutes les modifications en cours sur les bases sont annulées automatiquement !

Paramétrage de la gestion transactionnelle

Au moment de l'appel, vous pouvez spécifier la façon dont vous voulez que CopixServices gère la transaction

- CopixServices::NEW_TRANSACTION - Réalise un begin avant de démarrer le service, rollback en cas de problème sinon commit. (C'est le mode par défaut si vous ne spécifiez rien)
- CopixServices::CURRENT_TRANSACTION - Intègre les requêtes dans la transaction courante sans faire appel a begin / commit. Si aucune transaction n'est active, alors le mode est "auto commit" à chaque requête effectuée dans la base.

Exemple

```
CopixServices::process ('SomeService::someMethod', array ('param'=>$param),
        ✂ CopixServices::NEW_TRANSACTION);
```

Les modules

Qu'est-ce qu'un module ?

Un module dans Copix est un ensemble de classes, d'actions, de templates et de ressources qui servent une fonction particulière (un forum, un blog, un gestionnaire de nouvelles, une gestion d'activités, ...)

Il est possible de configurer plusieurs emplacements ou Copix ira chercher les modules qu'il peut installer / utiliser.

Cette variable se configure dans le **copix.conf.php** et se nomme *arModulesPath*.

exemple

```
$config->arModulesPath = array (
COPIX_PROJECT_PATH.'modules/public/stable/standard/',
COPIX_PROJECT_PATH.'modules/public/stable/webtools/',
COPIX_PROJECT_PATH.'modules/public/stable/tools/',
COPIX_PROJECT_PATH.'modules/public/devel/webtools/');
```

Voir aussi

- Le fichier **module.xml**

CopixClassesFactory

Présentation

CopixClassesFactory est une classe utilitaire vous permettant d'appeler les classes (qui respectent certaines convention) dans vos applications Copix.

Supposons que vous avez déclaré les classes suivantes dans votre arborescence Copix :

```
/modules
-/module_exemple
--/classes
---/classe1.class.php
---/classe2.class.php
-/module_exemple2
--/classes
---/exemple1.class.php
---/exemple2.class.php
```

Vous pouvez utiliser CopixClassesFactory depuis n'importe quelle partie de votre application pour instancier ces classes avec la syntaxe suivante :

```
$classe1 = CopixClassesFactory::create ('module_exemple|classe1');
$classe2 = CopixClassesFactory::create ('module_exemple|classe2');
$classe1 = CopixClassesFactory::create ('module_exemple2|exemple1');
$classe2 = CopixClassesFactory::create ('module_exemple2|exemple2');
```

Conventions

Les noms de fichier doivent être en minuscules, se nommer nomclass.class.php et contenir la définition de la classe nomclass.

Exemple :

```
//dans exemple.class.php
class Exemple {
   //définition de la classe
}

//pourra être appelée via _class ('exemple');
```

API Complète

Create

La méthode create inclut la définition de la classe demandée puis l'instancie.

```
$uneInstance      = CopixClassesFactory::create ('module_exemple|classe1');
$uneAutreInstance = CopixClassesFactory::create ('module_exemple|classe1');
```

Il existe une fonction raccourcie pour CopixClassesFactory::create qui est _class.

Depuis Copix 3.0.2+, il est possible d'envoyer des paramètres au constructeur de la classe instanciée, en donnant en second argument un tableau des paramètres à envoyer.

```
$uneInstance      = CopixClassesFactory::create ('module_exemple|classe1', array
('parametre1', 'parametre2'));

//équivaut à
CopixClassesFactory::fileInclude ('module_exemple|classe1');
new Classe1 ('parametre1', 'parametre2');
```

getInstanceOf

La méthode getInstanceOf est identique à la méthode create à l'exception qu'elle va entre plusieurs appels successifs pour un même objet retourner la même instance de l'objet. Cela vous assure de ne manipuler qu'une seule instance de votre classe, ce qui peut parfois être pratique pour ne calculer certaines opération lourdes qu'une seule fois.

```
$uneNouvelleInstanceAuPremierAppel     = CopixClassesFactory::getInstanceOf
           ✂ ('module_exemple|classe1');
$laMemeInstance                        = CopixClassesFactory::getInstanceOf
           ✂ ('module_exemple|classe1');
```

Il existe une fonction raccourcie pour CopixClassesFactory::getInstanceOf qui est _ioClass.

Vous pouvez gérer des instances uniques "identifiées", en fournissant un second paramètre.

```
$instance1     = CopixClassesFactory::getInstanceOf ('module_exemple|classe1',
            ✂ 'premier');
$idemInstance1 = CopixClassesFactory::getInstanceOf ('module_exemple|classe1',
            ✂ 'premier');

$instance2     = CopixClassesFactory::getInstanceOf ('module_exemple|classe1',
            ✂ 'second');//instance différente de $instance1
$idemInstance2     = CopixClassesFactory::getInstanceOf ('module_exemple|classe1',
            ✂ 'second');
```

fileInclude

La méthode fileInclude va se contenter d'inclure la définition de votre classe. C'est globalement un alias au morceau de code suivant :

```
require_once (CopixModule::getPath ('modulename').'classes/maclasse1.class.php');
//est équivalent à
CopixClassesFactory::fileInclude ('modulename|maclasse1');
```

Il existe une fonction raccourcie pour CopixClassesFactory::fileInclude qui est _classInclude.

modulexml

Présentation

Le fichier module.xml contient les informations principales de votre module, comme ses paramètres, ses dépendances, son nom, sa version, les événements auxquels il répond, ...

Ce fichier est nécessaire si vous voulez que Copix considère votre module (sans ce fichier, votre module n'apparaitra par exemple pas dans la liste des modules installables).

Exemples

Minimaliste

```
<?xml version="1.0"?>
<moduledefinition>
 <general>
  <default name="nom_repertoire" description="Description du module" />
 </general>
</moduledefinition>
```

Section general

La section générale (general) contient un seul attribut obligatoire qui est l'attribut nommé "default".

Cette section accepte ces attributs

- description qui est le nom du module "en clair", compréhensible par l'utilisateur.
- descriptioni18n qui est le nom du module "en clair", compréhensible par l'utilisateur, sous la forme d'une clef i18n (voir CopixI18N).
- longdescription qui est la description longue du module, généralement on explique ici ce que le module fait, en quelques lignes
- longdescriptioni18n qui est l'équivalent en clef i18n de longdescription
- icon icone du module, dans /www/themes/monTheme/img/icons/ (version 3.0.1)

Exemple sans utiliser I18N

```
<general>
 <default name="nom_repertoire" description="Description du module"
          ✂ longdescription="Ce module permet de faire tout un tas de chose plutot
          ✂ pratique, c'est pour ça qu'on l'a fait." />
</general>
```

Exemple Avec I18N

```
<general>
 <default name="nom" descriptioni18n="nom.clefDescription"
longdescriptioni18n="nom.clefLongDescription" />
</general>
```

Section paramètres

La section parameters vous permet de déclarer des paramètres qui seront administrables via l'écran d'administration et manipulables via **CopixConfig**.

```
<parameters>
  <parameter name="nom_parametre" caption="Un paramètre" default="1" />
  <parameter name="param_text" caption="Texte" type="text" default="Mon texte"
      ✂ maxLength="255" />
  <parameter name="param_int" caption="Entier" type="int" default="1" minValue="0"
      ✂ maxValue="100" />
  <parameter name="param_bool" caption="Oui / Non" type="bool" default="1" />
  <parameter name="param_email" caption="E-mail" type="email"
      ✂ default="monmail@mailer.ext" maxLength="255" />
  <parameter name="param_select" caption="Liste déroulante" type="select" default="1"
      ✂ listValues="0=>Ma valeur 0;1=>Ma valeur 1" />
  <parameter name="param_multiSelect" caption="Liste déroulante à choix multiple"
      ✂ type="multiSelect" default="1" listValues="0=>Ma valeur 0;1=>Ma valeur 1" />
</parameters>
```

La déclaration d'un paramètre accepte plusieurs attributs :

- name le nom du paramètre
- caption le libellé du paramètre (obligatoire)
- captioni18n la clef i18n pour le libellé du paramètre (obligatoire si caption n'est pas donné)
- default la valeur par défaut du paramètre à l'initialisation du module.
- type le type du paramètre (text, int, bool, email, select, multiSelect)
- minValue borne minimum de saisie pour le type int
- maxValue borne maximum de saisie pour le type int
- maxLength nombre de caractères maximum pour les types text et email
- listValues liste des valeurs pour les types select et multiSelect (syntaxe : 0=>Texte;1=>Texte)

En savoir plus sur les **paramètres**.

ATTENTION default correspond à la valeur par défaut qu'aura le paramètre lors de l'installation du module. Une fois le module installé, modifier la valeur de "default" dans le fichier n'aura aucun impact sur l'application. Pour modifier la valeur d'un paramètre, vous devez utiliser le module d'administration, section paramètres.

Groupes de paramètres (Copix 3.0.2 minimum)

Vous pouvez afficher les paramètres sous forme de groupes, pour séparer plusieurs types de paramètres. Les groupes ne servent que lors de l'affichage.

```
<parameters>
  <group captioni18n="auth.group.createUser">
    <parameter name="createUser" captioni18n="auth.createUser" default="0" type="bool"
            ✂ />
  </group>
  <group captioni18n="auth.group.others">
    <parameter name="timetolive" captioni18n="auth.remember_time" default="2678400"
            ✂ type="int" />
    <parameter name="cryptPassword" captioni18n="auth.cryptPassword" type="select"
            ✂ default="md5" listValues="md5=>MD5;sha1=>SHA1;SHA256=>SHA256" />
  </group>
</parameters>
```

La déclaration d'une node group doit avoir un attribut caption ou captioni18n.

Section dependencies

Cette section vous permet d'indiquer de quels modules dépend votre module courant.

Ceci permet à Copix de vérifier lors de l'installation de votre module que toutes les dépendances sont satisfaites. Si ce n'est pas le cas, Copix tentera d'installer les dépendances avant votre module.

```
<dependencies>
 <dependency name="copixheadings" kind="module"/>
 <dependency name="parameters" kind="module"/>
 <dependency name="profile" kind="module"/>
 <dependency name="gd" kind="extension"/>
 <dependency name="mysql_connect" kind="function"/>
 <dependency name="PHPUnitClass" kind="class"/>
</dependencies>
```

- le kind module teste l'existence d'un module et l'install si il n'est pas installé
- le kind extension teste l'installation d'une extension PHP, si elle n'est pas installé le module ne pourra pas etre installé
- le kind function teste l'existence d'une fonction, si elle n'est pas installé le module ne pourra pas etre installé
- le kind class teste l'existence d'une class, si elle n'est pas installé le module ne pourra pas etre installé

Section admin

Cette section vous permet de demander à Copix d'ajouter des liens pour votre module dans la page d'administration (module admin).

Exemple de fichier

```
<admin>
 <link url="admin|install|manageModules" captioni18n="install.button.manageModules"
          ✂ credentials="basic:admin" />
 <link url="admin|session|" captioni18n="session.admin" credentials="basic:admin" />
</admin>
```

Chaque élément link accepte 4 paramètres

- url une url au format Copix (CopixUrl)
- captioni18n une clef i18n pour le libellé du lien
- caption le libellé du lien si captioni18n n'est pas fourni
- credentials la chaîne de droit nécessaire pour pouvoir afficher le lien.

(A partir de copix 3.0.1) Si vous avez plusieurs modules qui ont un thème commun (Outils de développement, Outils divers, Administration, etc), vous pouvez créer un "groupe". Tous les modules qui ont le même attribut "groupid" auront leurs liens dans la partie admin dans une section portant le nom de l'attribut "groupcaption" ou "groupcaptioni18n", et non pas dans une section portant le nom du module.

```
<admin groupid="myGroup" groupcaptioni18n="myGroup.groupName" groupicon="tools.gif">
```

Attributs possibles pour admin

- groupid identifiant du groupe, qui doit être commun entre les modules d'un même groupe
- groupcaption caption du groupe (doit être présent dans au moins un des modules du groupe)
- groupcaptioni18n caption i18n du groupe (doit être présent dans au moins un des modules du groupe)
- groupicon icon du groupe, retour d'un _resource ('img/icons/' . $groupicon) (doit être présent dans au moins un des modules du groupe)

Section events

La section events permet de déclarer l'ensemble des listeners que votre module déclare ainsi que les évènements auxquels ils réagissent.

```
<events>
 <listeners>
   <listener name="NomListener">
    <event name="NomEvenement" />
    <event name="AutreEvenement" />
   </listener>
 </listeners>
</events>
```

En savoir plus sur les **évènements**.

Section credentials

La section credentials permet de déclarer des droits disponible pour notre module

```
<credentials>
    <credential name="commentaires">
        <value name="lecture" level="1">
        <value name="ecriture" level="2">
        <value name="moderation" level="3">
    </credential>
    <credential name="connexion" />
</credentials>
```

Nous déclarons ici plusieurs droits, tout d'abord

1. le droit commentaires qui possède 3 valeurs différentes :
    1. lecture ayant un level de 1
    2. ecriture ayant un level de 2
    3. moderation ayant un level de 3
2. le droit connexion qui ne possède pas de sous-valeur

Pour plus d'infos sur la **gestion des droits par module**.

ModuleCredential

ModuleCredential vous permet de définir des droits pour votre module, il vous suffit pour cela de les déclarer, et au moment de l'installation, ces droits seront installé et disponible dans l'interface de gestion des groupes.

Déclaration des droits

La déclaration des droits se passe dans le fichier **module.xml**.

Test du droit

Vous pouvez ensuite tester votre droit en faisant seulement

```
//Test d'un droit déclarer dans le module news, ayant pour nom commentaires et
        ✂ pour valeur lecture
CopixAuth::getCurrentUser()->assertCredential
        ✂ ('module:commentaires|lecture@news');
```

Exemple de droit

Exemple pour de credential dans un module appelé news

Déclaration dans **module.xml**

```
<credentials>
    <credential name="commentaires">
        <value name="lecture" level="1" />
        <value name="ecriture" level="2" />
        <value name="moderation" level="3" />
    </credential>
    <credential name="connexion" />
</credentials>
```

Nous déclarons ici plusieurs droits, tout d'abord

1. le droit commentaires qui possède 3 valeurs différentes :
    1. lecture ayant un level de 1
    2. ecriture ayant un level de 2
    3. moderation ayant un level de 3
2. le droit connexion qui ne possède pas de sous-valeur

Les droits répondront alors de la manière suivante

1. Si le droit commentaires - lecture est coché dans l'interface, le groupe aura les droits sur la chaine module:commentaires|lecture@news

2. Si le droit commentaires - ecriture est coché dans l'interface, le groupe aura les droits sur la chaine module:commentaires|ecriture@news et module:commentaires|lecture@news
3. Si le droit commentaires - moderation est coché dans l'interface, le groupe aura les droits sur la chaine module:commentaires|moderation@news, module:commentaires|ecriture@news et module:commentaires|lecture@news
4. Si le droit connexion est coché , le groupe aura les droits sur la chaine module:connexion@news

Les plugins

Introduction

Les plugins sont des petites fonctionnalités qui vous permettent de rajouter des capacités à votre application.

Les fonctionnalités peuvent être d'ordre systématiques, comme par exemple des statistiques, des échappements de caractères spéciaux provenant de l'url, des contrôles de sécurité, des calculs de performances, ...

Les plugins sont développés dans le répertoire plugins des modules. Le nombre des plugins n'est pas limité. Vous pouvez activer / désactiver des plugins via l'interface d'administration dans la section "Activer / Désactiver des plugins".

Les deux composantes

Les plugins sont constitués de deux parties, la partie configuration (facultative) et la partie plugin. Ces deux parties se présentent sous forme de classes :

- La partie plugin correspond à l'implémentation des objectifs du plugin, ou en tout cas à l'appel de cette implémentation.
- La partie configuration ne sert qu'a recevoir les options d'exécution de ces plugins.

Dans le cas d'un plugin nommé Exercice, la classe PluginExercice sera déclarée dans project/plugins/exercice.plugin.class.php.

Cette classe devra hériter de CopixPlugin.

La classe CopixPlugin propose la signature de trois méthodes, méthodes qui seront appelées à différentes étapes de la création de la réponse.

La classe Plugin

Méthode beforeSessionStart

Cette méthode est appelée juste avant le démarrage effectif de la session. Par exemple, si vous souhaitez déclarer des classes avant le démarrage de la session, si vous souhaitez effectuer des traitements avant l'appel à session_start, c'est ici que vous placerez votre code.

Méthode beforeProcess ($action)

Cette méthode est appelée avant l'appel de l'implémentation de l'action (méthode de l'objet ActionGroup).

En paramètre, cette fonction reçoit l'action (CopixAction) qui va être exécutée.

Ici, vous avez l'opportunité de contrôler et de modifier ce qui sera réellement exécuté, en modifiant l'action.

Méthode afterProcess

La méthode afterProcess est appelée juste après votre **action** et en reçoit le code retour (**CopixActionReturn**).

Méthode beforeDisplay ($pContent)

La méthode beforeDisplay est appelée juste avant l'affichage du contenu de votre page. Cette méthode n'est bien sûr appelée que si votre action génère un affichage.

Cette méthode reçoit en paramètre une chaine de caractère qui corresponds à la chaine qui va être affichée au navigateur du client.

La classe de configuration du plugin

La classe de configuration est une classe libre qui nous servira à configurer notre plugin.

Dans le cas d'un plugin nommé **Exercice**, la classe de configuration du plugin doit s'appeler *PluginConfExercice*, hériter de *CopixPluginConf* et être située dans le fichier *exercice.plugin.conf.php*.

Gestion d'événements

Présentation

Copix vous permet de profiter d'un système d'évènements afin de réaliser un couplage dit "souple" entre vos modules.

Ce système d'évènement permet, en une ligne, de faire en sorte que votre module prévienne tous les autres modules installés qu'il a réalisé une action (ajouté un contenu, supprimé un contenu, a mis à jour les droits, ...) afin que d'autres puissent avoir le choix d'y répondre.

C'est un système de couplage "souple" car il ne lie pas physiquement les modules concernés entre eux et permet à chacun de s'exécuter indépendamment les uns des autres.

Ce système d'évènement peut être intéressant lorsque vous développez des modules à fort caractère réutilisable (un moteur de recherche, un flux RSS pour indiquer les changements de votre site, un système de notification par mail, ...)

Différences entre couplage fort / couplage souple

Présentation du problème

Vous disposez d'un module de nouvelles.

Vous souhaitez que votre module de nouvelles ajoute ses publications au moteur de recherche. Aujourd'hui, vous avez développé un module de moteur de recherche nommé mysearch.

Code de la publication de nouvelle :

```
function processAddNews (){
    //.... tout le code de validation est effectué avant

    //insertion de la nouvelle
   _ioDAO ('nouvelle')->insert ($recordNouvelle);

   return _arRedirect ('nouvelle|admin|');
}
```

Couplage fort, sans évènement

Avec un couplage fort, inutile de se poser trop de question, le réflexe est de procéder ainsi :

Implémentation

```
function processAddNews (){
    //.... tout le code de validation est effectué avant

    //insertion de la nouvelle
   _ioDAO ('nouvelle')->insert ($recordNouvelle);

   //On ajoute la nouvelle au moteur de recherche
   _ioClass ('mysearch|index')->add ($recordNouvelle->titre, $recordNouvelle-
>contenu);

   return _arRedirect ('nouvelle|admin|');
}
```

Ici, clairement, on appelle en dur la classe d'indexation de notre moteur de recherche, et on suppose que notre module "nouvelle" soit parfaitement au fait de l'API de ce dernier.

Inconvénients

- Si la classe index du moteur de recherche change, il faut modifier le code dans nouvelle
- Si on souhaite utiliser un autre moteur de recherche que "mysearch", il faudra modifier le code dans nouvelle
- Si on ne souhaite pas livrer de module de recherche, il sera nécessaire de supprimer tous les appels dans le module de nouvelle qui en est dépendant.

Et encore, ce n'est ici que la face visible du problème, car aujourd'hui cela ne concerne qu'un moteur de recherche.

Le coeur du problème, là où les choses se complexifient de façon exponentielle, c'est lorsqu'en plus d'un moteur de recherche nous souhaitons réaliser d'autres éléments (notifier de la publication d'une nouvelle, envoyer un mail pour une newsletter, ajouter le contenu aux nouveaux contenus du site, ...)

Le tout va faire que notre module sera ancré dans ses dépendances, et au final notre module qui devait être réutilisable ne le sera presque pas, ou en tout cas pas de façon "unitaire".

Couplage faible

C'est la que le système d'évènement rentre en jeu. Plutôt que d'appeler directement les classes d'un module, vous allez vous contenter d'indiquer à qui veut bien l'entendre que vous avez ajouté un contenu.

Implémentation

```
function processAddNews (){
    //.... tout le code de validation est effectué avant

    //insertion de la nouvelle
   _ioDAO ('nouvelle')->insert ($recordNouvelle);

   //on lance l'évènement
   _notify ('PublishedContent', array ('title'=>$recordNouvelle->titre,
                                       'content'=>$recordNouvelle->content));

   return _arRedirect ('nouvelle|admin|');
}
```

Conséquence

Ici on voit clairement qu'aucune API ou aucune dépendance n'est rajoutée à notre module de nouvelle, mais que ce dernier est ouvert au dialogue avec quiconque le souhaitera.

L'inconvénient majeur sera qu'il vous sera nécessaire de développer un **listener** qui servira de pont entre l'interface de l'évènement et l'API de la recherche elle-même.

C'est donc un processus de développement un peu plus long, mais pour une souplesse plus grande.

Utilisé à bon escient, le système d'évènement vous permet de développer des modules capables de s'intégrer dans un ensemble cohérent et ouvert au dialogue.

CopixEvent

Présentation

CopixEvent est la classe qui représente un **évènement**. Lorsque vous **lancez un évènement**, vous décrivez le nom de l'évènement et un ensemble de paramètres.

Tout ceci est passé au **listener** sous la forme d'un objet de type CopixEvent.

Cet objet dispose de différentes méthodes afin de faciliter la lecture de ces paramètres.

Méthodes de CopixEvent

getName

La méthode getName indique le nom de l'évènement qui a été lancé.

getParam ($pNomParametre)

La méthode getParam indique la valeur du paramètre $pNomParamètre de l'évènement. Si le paramètre n'existe pas, la fonction retourne null.

Exemple

```
//l'évènement est lancé
_notify ('Evenement', array ('param'=>'valeur'));

//...
//dans le listener
function processEvenement ($pEvent, $pEventResponse){
   $paramValue = $pEvent->getParam ('param');//vaut 'valeur' pour l'évènement du dessus
}
```

CopixEventNotifier

Avant propos

Avant de lire cette section, assurez-vous d'avoir lu la présentation de la **gestion des évènements**.

Présentation

Lancer un évènement est un processus particulièrement simple. Il suffit en tout et pour tout d'indiquer le nom de l'évènement à lancer, et un ensemble de paramètres sous la forme d'un tableau.

exemple

```
//Sans aucun paramètre
_notify ('NomEvenement');

//Avec un paramètre
_notify ('NomEvenement', array ('parametre'=>'valeur'));
```

Syntaxe complète

CopixEventNotifier est la classe en charge de transmettre et ensuite de dispatcher les évènements aux différents **listeners**.

Elle dispose d'une seule méthode publique, "notify", dont le **raccourci** est _notify.

Cette méthode notify accepte deux paramètres

- L'évènement ou le nom de l'évènement
- Un tableau de paramètre pour l'évènement si le premier paramètre était le nom de l'évènement. (ce paramètre n'est pas pris en compte si le premier argument donné à la fonction est de type **CopixEvent**)

exemple avec un évènement

```
CopixEvent::notify (new CopixEvent ('NomEvenement', array ('parametre'=>'valeur')));
//ou
_notify (new CopixEvent ('NomEvenement', array ('parametre'=>'valeur')));
```

exemple avec le nom de l'évènement

```
CopixEvent::notify ('NomEvenement', array ('parametre'=>'valeur'));
//ou
_notify ('NomEvenement', array ('parametre'=>'valeur'));
```

CopixListener

Présentation

Pour pouvoir intercepter un **évènement**, il est nécessaire de passer par deux étapes :

- Indiquer dans le fichier **module.xml** que nous disposons d'un listener capable d'interpréter l'évènement
- créer un listener pour répondre à l'évènement.

Création du listener

Emplacement

Les listeners sont des classes qui seront stockées dans les répertoires /classes/ de vos modules. Les fichiers se nomment "nom.listener.php". La classe listener se nomme ListenerNom et hérite de CopixListener. Le listener comporte des méthodes nommées processNomEvenementATraiter, un peu comme les méthodes **actions** dans les **Actiongroup**.

exemple

```
//fichier classes/exemple.listener.php
class ListenerExemple extends CopixListener {
   function processEvenement ($pEvent, $pEventResponse){
      //traitement de l'évènement
   }
}
```

Paramètres des méthodes processXXX

$pEvent l'évènement donné en paramètre

$pEvent correspond à l'évènement (de type **CopixEvent**) qui a été **lancé**.

exemple

```
function processEvenement ($pEvent, $pEventResponse){
   if ($idNews = $pEvent->getParam ('id_nouvelle')){
      //traitement de la nouvelle d'id $idNews
   }
}
```

$pEventResponse pour apporter une réponse

Le paramètre $pEventResponse est de type **CopixEventResponse** et vous permet d'indiquer à l'appelant que vous avez effectué des traitements.

```
function processEvenement ($pEvent, $pEventResponse){
   //des traitements sont effectués
   //la réponse doit être donnée sous la forme d'un tableau clef / valeur
   $pEventResponse->add (array ('elementDeResponse'=>'valeur',
'elementDeReponse2'=>'valeur2'));
}
```

Enregistrement dans le fichier module.xml

Lorsque vous avez développé votre listener et que vous l'avez préparé pour répondre correctement aux évènements désirés, il vous suffit de l'enregistrer dans le fichier **module.xml** du module qui écoute les évènements en question (peu importe celui qui les lance) dans la section events / listeners.

Vous écrirez une balise listener pour chaque listener que vous déclarez. Vous indiquez une balise "event" pour chaque évènement auquel répond votre listener.

```
<events>
 <listeners>
   <listener name="NomListener">
    <event name="NomEvenement" />
    <event name="AutreEvenement" />
   </listener>
 </listeners>
</events>
```

Note : Pour que votre listener soit actif, il faut que le module auquel il appartient soit installé.

CopixEventResponse

Présentation

La classe CopixEventResponse permet d'apporter / lire une réponse d'un ou des **listeners** après avoir **lancé un évènement**.

Exemple (utilisé dans le module quicksearch)

```
//demande aux modules de lister leurs contenus.
$response = _notify ('ListContent');

//récupère l'ensemble des contenus donnés par les modules (dans url)
foreach ($response->getResponse () as $key=>$responses){
        if (isset ($responses['url']) && is_array ($responses['url'])){
                $ppo->arLinks = array_merge ($ppo->arLinks, $responses['url']);
        }
}

//pour fournir $responses['url'], les listener ont effectué le code suivant :
function processListContent ($pEvent, $pEventResponse){
        $pEventResponse->add (array ('url'=>$tableauDeLiens));
}
```

Méthodes

Ecriture d'une réponse avec add

La méthode add permet d'ajouter un ensemble de données à la réponse. Cette méthode accepte un seul paramètre qui doit être un tableau.

exemple

```
        $pEventResponse->add (array ('parametre'=>'valeur'));
        $pEventResponse->add (array ('parametre'=>'valeur', 'parametre2'=>'valeur2'));
```

Chaque appel à la méthode add ajoute un élément dans la réponse.

Lecture d'une réponse

utilisation de inResponse pour savoir si un élément est présent dans la réponse

La méthode inResponse permet de regarder si l'on a trouvé un élément donné dans une des réponses reçues.

Exemple

```
//Notification de suppression
$response = _notify ('DeletedContent', array ('id'=>$id));

//on regarde si tous les modules acceptent que l'on supprime l'élément
if ($response->inResponse ('failed', true)){
   //impossible de supprimer, un module à besoin de l'élément
}else{
   //supression effective
}

//Voici le code dans le listener pour indiquer failed :
function processDeletedContent ($pEvent, $pResponse){
   //traitement
   $pResponse->add (array ('failed'=>true));
}
```

parcours de toutes les réponses avec getResponse

La méthode getResponse retourne dans un tableau toutes les réponses apportées via la méthode add.

exemple

```
//demande des contenus du site
$response = _notify ('ListContent');

//récupère l'ensemble des contenus donnés par les modules (dans url)
foreach ($response->getResponse () as $key=>$responses){
        if (isset ($responses['url']) && is_array ($responses['url'])){
                $ppo->arLinks = array_merge ($ppo->arLinks, $responses['url']);
        }
}

//pour fournir $responses['url'], les listener ont effectué le code suivant :
function processListContent ($pEvent, $pEventResponse){
        $pEventResponse->add (array ('url'=>$tableauDeLiens));
}
```

Templates

Template_principal

Introduction

Ce document a pour but de présenter les éléments du template principal et la façon de le configurer.

Préambule

Copix utilise un pattern de type "2 Step View ", c'est à dire que le contenu final est (la plupart du temps) généré en deux temps.

Le premier temps est accordé à votre Action qui aura pour rôle de générer la partie centrale du contenu.

Le second temps est accordé au contrôleur de Copix, qui aura pour objectif d'assigner la charte globale du projet (éléments de menu, en tête, pied de page et plus généralement tous les éléments systématiques dans la charte de votre application).

Ainsi, vous utiliserez toujours deux templates

- Le template de l'action que vous spécifiez à chaque retour (return _arPpo ($ppo, 'templateDeLAction.tpl));
- Le template principal (dont il est question ici) configuré pour être choisi automatiquement

Configuration

En standard

Avec l'installation par défaut de Copix, le template principal correspond au fichier *project/modules/stable/default/templates/main.php* comme template principal.

Ce dernier est configuré dans le fichier de configuration **copix.conf.php**

```
//Template principal
$config->mainTemplate   = 'default|main.php';
```

Rappel: la syntaxe 'default|main.php' est une syntaxe qui se retrouve souvent dans Copix et qui désigne un élément dans un module particulier. Ici, on désigne l'élément main.php du module default.

Pour tester

Faites une copie du fichier main.php dans main_test.php

Si vous souhaitez tester l'utilisation d'un autre template principal, n'hésitez pas à copier le fichier *project/modules/stable/default/templates/main.php* vers *project/modules/stable/default/templates/main_test.php*.

Dans le fichier de configuration, modifiez la ligne correspondant au template principal pour refléter ce changement.

```
//Template principal
$config->mainTemplate   = 'default|main_test.php';
```

Modifiez le contenu du fichier *main_test.php* (inutile d'être trop inventif) et constatez que les changements sont bien pris en compte dans votre application.

Note: Vous verrez plus tard qu'il existe un système de thème graphique livré avec Copix pour vous permettre d'alterner automatiquement entre plusieurs chartes graphiques.

Contenu du fichier

Le template principal est proposé avec des variables pré-configurées que vous avez tout loisir d'utiliser (et qu'il est recommandé de placer à bon escient dans la plupart des cas).

TITLE_BAR

Cette variable contient ce que le développeur souhaite mettre dans la barre de titre du navigateur. Ainsi, la plupart du temps, il est recommandé de la placer dans la section <title> du template.

Dans le template

```
<head>
 <title><?php echo $TITLE_BAR; ?></title>
</head>
...
```

Dans l'action

```
//dans un ActionGroup
public function processMonAction (){
   $ppo = new CopixPpo ()
   $ppo->TITLE_BAR = 'Contenu de la barre de titre';
   return _arPpo ($ppo, 'template_interieur.php');
}
```

Valeurs par défaut

Si le développeur ne spécifie aucune valeur pour TITLE_BAR, alors TITLE_BAR utilisera le paramètre 'default|titleBar' pour se remplir.

Vous pouvez modifier cette valeur via le module admin, dans l'administration des paramètres.

Par défaut, cette valeur est configurée avec [Copix]{$TITLE_PAGE}, ce qui signifie qu'au cas où aucune valeur n'est indiquée pour TITLE_BAR, alors Copix affichera pour TITLE_BAR la chaîne [Copix] suivie de la valeur renseignée pour TITLE_PAGE (voir plus loin la description de TITLE_PAGE).

Nous vous conseillons de mettre comme paramètre par défaut [Nom de votre application]{$TITLE_PAGE} pour cette valeur.

TITLE_PAGE

Cette variable contient ce que le développeur souhaite donner comme nom à sa page. C'est en gros la description en une phrase de ce que représente la page.

Généralement, nous vous conseillons de la placer dans un "h1" (qui sémantiquement correspond bien au titre de niveau 1)

Dans le template

```
...
<body>
<h1><?php echo $TITLE_PAGE; ?></h1>
...
</body>
```

Dans l'action

```
//dans un ActionGroup
public function processMonAction (){
   $ppo = new CopixPpo ()
   $ppo->TITLE_PAGE = 'Mon action';
   return _arPpo ($ppo, 'template_interieur.php');
}
```

Valeurs par défaut

Par défaut, si le développeur à oublié d'indiquer un nom de page pour son action, Copix utilise le paramètre 'default|titlePage'. En standard, ce paramètre vaut "Copix Framework".

En général, nous vous conseillons de toujours spécifier un titre de page pour vos actions.

HTML_HEAD

Cette variable est indispensable si vous souhaitez utiliser la classe **CopixHtmlHeader** et certains **tags** dans vos **templates**.

En effet, Copix, lorsque vous faites appel à certaines fonctions, s'occupe pour vous de préparer l'inclusion des bonnes bibliothèques javascript / css pour vous.

C'est la variable HTML_HEAD du template principal qu'il va utiliser pour ce faire, d'où son caractère indispensable.

Elle devra se situer dans la partie "head" de votre template (généralement à la fin)

Dans le template

```
<head>
...
<?php echo $HTML_HEAD; ?>
</head>
```

MAIN

La partie MAIN correspond à l'emplacement où le contenu généré par votre action sera affiché.

Si vous souhaitez que votre action puisse afficher du contenu, cette variable est bien évidemment indispensable.

Dans le template

```
<body>
...
<?php echo $MAIN; ?>
...
</body>
```

Dans l'action

```
//dans un ActionGroup
public function processMonAction (){
   $ppo = new CopixPpo ()
   //...
   return _arPpo ($ppo, 'template_interieur.php');//template_interieur sera affiché
            ✂ dans la partie MAIN du template principal
}
```

Aller plus loin

- Si vous voulez que soient assignées automatiquement d'autres variables à votre template principal, modifiez le ProjectController.
- Allez voir comment créer des thèmes graphiques.
- Tout savoir sur PPO.

tags

Présentation

Les tags sont des extensions qui vous permettent en une ligne de réaliser la génération d'un code HTML long et rébarbatif.

Dans Copix, les tags sont généralement disponibles sous deux formes, Smarty et PHP.

Blocs

- .CopixHtmlHeader.
- .PopupInformation.
- **WikiEditor**

Fonctions

- .AutoComplete.
- .Calendar.
- .CopixLogo.
- .CopixPicture.
- .CopixResource.
- **CopixTag**
- .CopixTips.
- .CopixUrl.
- .CopixZone.
- **CSV**
- **CurrentUrl**
- .cycle.
- .ErrorMsg.
- .Escape.
- **FavIcon**
- .FormFocus.
- .HTMLEditor.
- .i18n.
- .InputText.
- **JSSubmitForm**
- .LinkBar.
- .MooTools.
- .RadioButton.
- .Select.
- .MultipleSelect.
- .ULLI.

Modificateurs

- .DateI18N.
- .DateTimeI18N.
- .Hour_Format.
- .Time.

- ToArray
- Url
- Var_Dump

Voir aussi

Dans les templates Smarty, il est bien évidemment possible d'utiliser tous les tags et modificateurs livrés avec Smarty. Pour plus d'informations, veuillez vous reporter au manuel Smarty.

CopixHTMLHeader

Introduction

La classe CopixHTMLHeader a pour objectif de permettre la manipulation de la partie entête du fichier HTML (head) généré par Copix à la fin du processus de rendu.

Tout ce qui aura été ajouté via cette classe sera assigné au template principal dans une variable spéciale nommée "$HTML_HEAD".

On se sert souvent de cette classe lorsqu'il faut inclure des codes javascripts ou styles spécifiques à certains écrans. Copix utilise cette fonctionnalité pour plusieurs balises.

API

CopixHtmlHeader

addJsLink ($fichier, $parametresSupplementaires)

Cette fonction ajoute un lien vers un fichier javascript. Si le même fichier est inclus deux fois, il ne sera déclaré qu'une seule fois dans l'entête.

Le tableau de paramètres supplémentaires peut servir à rajouter des options lors de l'inclusion du fichier javascript, ces options seront rajoutées dans la balise "script".

exemple

```
//en utilisant le système de ressources
CopixHTMLHeader::addJsLink (_resource ('js/mon_fichier_js.js'));
//avec une url absolue
CopixHTMLHeader::addJsLink ('http://www.copix.org/js/mon_fichier_js.js');
```

addCSSLink ($fichier, $parametresSupplementaires)

Cette fonction ajoute un lien vers un fichier CSS.

Si le même fichier est inclus deux fois, il ne sera déclaré qu'une seule fois dans l'entête.

Le tableau de paramètres supplémentaires peut servir à rajouter des options lors de l'inclusion du fichier.

Exemple

```
//en utilisant le système de ressources
CopixHTMLHeader::addCSSLink (_resource ('css/mon_fichier_js.css'));
//avec une url absolue
CopixHTMLHeader::addCSSLink ('http://www.copix.org/css/mon_fichier_css.css');
```

addStyle ($selector, $def = null)

Ajoute la déclaration d'un style CSS qui sera ajouté dans un ensemble "style" dans l'entête.

$def doit contenir l'ensemble des propriétés de style sous la forme d'une chaîne de caractères (sans les accolades).

Si plusieurs appels sont effectués à cette fonction avec le même sélecteur, seul le premier appel est pris en compte.

Exemple

```
CopixHtmlheader::addStytle ('h2', 'border-bottom: 1px solid #000; font-size: 1em;');
```

Si seul $selector est spécifié, alors il devra contenir un ensemble de déclarations CSS valides, par exemple

```
CopixHtmlheader::addStytle ('h2 {border-bottom: 1px solid #000; font-size: 1em}');
```

addOthers ($content, $key=null)

Ajoute un contenu dans l'en tête HTML, ce contenu sera ajouté tel quel, sans aucune modification.

Si une clef est donnée ($key), CopixHTMLHeader s'en servira comme identifiant unique et n'ajoutera le code en question qu'une seule fois par clef (le dernier appel écrasant les précédents).

```
CopixHTMLHeader::addOthers ('<link rel="icon" href="icone.jpg" />');
```

addJSCode ($code, $key = null)

Ajoute une portion de code Javascript dans l'en tête HTML. Le code javascript ajouté ne doit pas contenir les balises de déclaration Javascript.

Si une clef est donnée ($key), CopixHTMLHeader s'en servira comme identifiant unique et n'ajoutera le code en question qu'une seule fois par clef (le dernier appel écrasant les précédents).

addFavIcon ($pPicturePath)

Ajoute une icone à votre page. $pPicturePath est le chemin de l'image.

```
CopixHTMLHeader::addFavIcon (_resource ('img/tools/favicon.jpg'));
```

get ()

Retourne une chaîne de caractères qui correspond à l'en tête du fichier HTML. Cette méthode est appelée par Copix lors du rendu du fichier final.

clear ($what)

Permet de supprimer de l'entête HTML certains éléments. $what est un tableau de valeurs qui peut contenir 'CSSLink', 'Styles', 'JSLink', 'JSCode', 'Others'.

Tag Popupinformation

Présentation

Le tag popupinformation vous permet de réaliser un "popup javascript" en utilisant des DIV, avec une image ou un texte déclencheur

Exemple d'appel avec Smarty

```
{popupinformation} Contenu du popup {/popupinformation}
```

Avec un contenu HTML

```
{popupinformation} <p style="color: #F00; font-weight: bold;">Test de style</p>
{/popupinformation}
```

Liste des paramètres

Spécifier un contenu texte alternatif pour l'image

```
{popupinformation alt="Sur l'image"} Contenu du popup avec alt {/popupinformation}
```

Ajouter un texte après l'image

```
{popupinformation text="Du texte après l'image"} Contenu du popup avec texte
{/popupinformation}
```

Sans image

```
{popupinformation displayimg=false text="Du texte"} Contenu du popup avec texte et sans
image {/popupinformation}
```

En utilisant une autre image que celle par défaut

```
{copixresource path=img/tools/loupe.png assign=imgPath} {popupinformation img=$imgPath}
✂Contenu du popup avec une image alternative {/popupinformation}
```

En spécifiant une classe CSS pour le popup

```
{copixhtmlheader kind="style"} .divClass { border: 1px solid #F00; }
✂{/copixhtmlheader} {popupinformation divclass=divClass} Contenu du popup
✂avec une classe css spécifique {/popupinformation}
```

Déclencher le popup sur un click plutot qu'au survol

```
{popupinformation handler=onclick} Contenu du popup avec onclick {/popupinformation}
```

Charger le contenu d'une zone en ajax

```
{popupinformation handler=onclick zone='module|zone'}{/popupinformation}
```

Tag Autocomplete

Présentation

Le tag AutoComplete vous permet de faire un "input" pour votre formulaire qui bénéficiera de l'autocompletion, facilement paramétrable.

Forme Smarty

```
{*avec view*}
{autocomplete dao="ma_table" field="caption" view="caption;champ1;champ2"}

{*sans view*}
{autocomplete dao="ma_table" field="caption"}
```

Forme PHP

```
// avec view
_tag ('autocomplete', array ('field'=>'caption', 'dao'=>'ma_table',
'view'=>'caption;champ1;champ2'));
// sans view
_tag ('autocomplete', array ('field'=>'caption', 'dao'=>'ma_table'));
```

Paramétrage

Seuls deux paramètres sont obligatoires :

- dao
- field

la source de données pour l'autocompletion : datasource

Le tag autocomplete utilise des "datasources" comme sources de données. Par défaut, le datasource est positionné à "dao" pour indiquer que la source de données est un DAO.

Aujourd'hui, seul le datasource "dao" est pris en compte.

le DAO à utiliser : dao

Pour spécifier le DAO, il suffit de donner un paramètre "dao" au tag.

exemple :

```
{autocomplete dao="ma_table" field="caption"}
```

Spécifier une connexion autre que celle par défaut avec ct

Si vous utilisez un DAO comme source de données (datasource), vous pouvez demander à Copix d'utiliser une connexion différente de celle par défaut avec le paramètre optionnel ct.

```
{autocomplete dao="ma_table" field="caption" ct="nom_connexion"}
```

le champ à utiliser : field

Pour spécifier à quel champ se rapporte l'autocompletion, vous utiliserez le paramètre "field".

Indiquer les champs visibles dans l'autocompletion

Le paramètre "view" permet d'indiquer les champs que l'on souhaites voir lors de l'autocompletion. Pour spécifier les champs, il suffit d'en faire une liste séparée par des points virgules ";".

Nombre de caractères pour déclencher l'autocompletion

Par défaut, l'autocompletion est déclenchée dès la saisie du premier caractère. Si vous souhaitez modifier ce comportement, vous pouvez le faire en spécifiant le paramètre "length".

```
//La completion ne se déclenchera qu'au bout de 3 caractères
{autocomplete dao="ma_table" field="caption" length="3"}
```

Tag Calendar

Présentation

Le tag calendar permet d'afficher une zone de saisie assistée pour les dates.

Exemple PHP

```
<?php _eTag ('calendar', array ('name'=>'date_publication')); ?>
```

Exemple Smarty

```
{calendar name="date_publication"}
```

Paramètres

name

La paramètre name (obligatoire) permet de spécifier le nom du champ de formulaire.

```
{calendar name="date_publication"}
```

image

Le paramètre image permet de spécifier une image alternative pour le calendrier.

```
{copixresource path=img/tools/loupe.png assign=path}
{calendar name=cal image=$path}
```

value

Le paramètre value permet d'indiquer le contenu par défaut du champ. Le texte sera affiché sans transformation.

```
{calendar name=cal value="01/02/2017"}
```

yyyymmdd

Le paramètre yyyymmdd permet d'indiquer la valeur du champ depuis un format YYYYMMDD. La valeur sera affichée au format date local (dd/mm/aaaa pour la France).

yyyymmdd prend le pas sur value si les deux sont indiqués.

```
{calendar name=cal yyyymmdd="20170201"}
```

timestamp

Le paramètre timestamp permet d'indiquer la valeur du champ depuis un timestamp. La valeur sera affichée au format date local (dd/mm/aaaa pour la France)

timestamp prend le pas sur yyyymmdd et sur value s'il est spécifié.

extra

Extra permet de rajouter des éléments dans la balise input type="text" générée. Le contenu sera placé tel quel.

```
{calendar name=cal extra='style="background-color: #ccc;"'}
```

size

Size permet de spécifier le nombre de caractères saisissables dans le champ de saisie. Par défaut 8.

```
{calendar name=cal size=6}
```

lang

Lang permet d'indiquer dans quelle langue seront affichés les informations du calendrier. Cette option n'a aucune influence sur le format de la date. Par défaut la langue courante.

```
{calendar name=cal lang=en}
```

format

Format permet d'indiquer dans quel format sera affiché la date. Par défaut le format de la langue courante.

```
{calendar name=cal format=yyyymmdd}
```

sizeday

Sizeday permet d'indiquer le nombre de caractères que cous souhaitez afficher pour les noms des jours de la semaine. Par défaut 3.

```
{calendar name=cal sizeday=1}
```

beforeyear & afteryear

Beforeyear & afteryear vous permettent d'indiquer le nombre d'années qui apparaitront dans la liste déroulante respectivement avant et après l'année courante (par défaut 10).

```
{calendar name=cal beforeyear=2 afteryear=2}
```

duration

Duration représente le temps en millisecondes que le calendrier mettra pour apparaitre. Par défaut 500.

```
{calendar name=cal duration=0}
```

tabindex

Tabindex représente la position du champ dans l'ordre des tabulations.

```
{calendar name=cal tabindex=2}
```

Voir aussi

- Le module tutorials "tags_demo"

Tag Copixlogo

Présentation

Ce tag à pour seul objet de faire de la publicité à Copix :-)

Il génère de différentes façon un contenu qui permet d'indiquer aux autres que vous avez utilisé Copix pour faire votre application.

Tag Copixpicture

Présentation

Le tag copixpicture est généralement utilisé pour afficher une image.

Exemple Smarty

```
{copixpicture resource="img/logo.png"}
```

Exemple PHP

```
<?php _etag ('copixpicture', array ('resource'=>"img/logo.png")); ?>
```

Paramètres

resource

Le paramètre resource permet de spécifier le nom du fichier image. Cette valeur sera ainsi rajoutée au code HTML produit par Smarty.

Par exemple, le code Smarty suivant :

```
{copixpicture resource="img/logo.png"}
```

Produira le code HTML suivant :

```
<img src="http://monsite.com/themes/default/img/logo.png" alt="" title=""  />
```

id

Si le module pictures est activé, vous pouvez utiliser le paramètre id à la place du paramètre resource. Ce paramètre permet de préciser l'identifiant de l'image choisie.

width

Le paramètre width permet de préciser la largeur de l'image. Cette valeur sera ainsi rajoutée au code HTML produit par Smarty.

Par exemple, le code Smarty suivant :

```
{copixpicture resource="img/logo.png" width="100"}
```

Produira le code HTML suivant :

```
<img src="http://monsite.com/themes/default/img/logo.png" alt="" title=""
        ✂ width="100" />
```

height

Le paramètre height permet de préciser la hauteur de l'image. Cette valeur sera ainsi rajoutée au code HTML produit par Smarty.

Par exemple, le code Smarty suivant :

```
{copixpicture resource="img/logo.png" height="100"}
```

Produira le code HTML suivant :

```
<img src="http://monsite.com/themes/default/img/logo.png" alt="" title=""
        ✂ height="100" />
```

title

Le paramètre width permet de préciser le titre de l'image. Cette valeur sera ainsi rajoutée au code HTML produit par Smarty (attributs **alt** et **title**).

Par exemple, le code Smarty suivant :

```
{copixpicture resource="img/logo.png" title="MonImage"}
```

Produira le code HTML suivant :

```
<img src="http://monsite.com/themes/default/img/logo.png" alt="MonImage"
        ✂ title="MonImage"  />
```

assign

Le paramètre assign permet d'indiquer le nom de la variable à laquelle sera assigné le code HTML produit par Smarty. Attention, en précisant ce paramètre, l'image ne sera pas affichée mais seulement assignée à une variable.

```
{copixpicture resource="img/logo.png" assign="maVariable"}

{* et si l'on souhaite ensuite l'utiliser *}
{$maVariable}
```

Voir aussi

- Le tag CopixResource

Tag Copixresource

Présentation

La balise copixresource n'est disponible que dans les templates de type Smarty. C'est un moyen pour les designers d'utiliser les méthodes _resource / CopixUrl::getResource () directement dans les templates, afin de profiter du système de thèmes graphiques et en particulier la personnalisation des ressources.

Exemple

```
{copixresource path="img/test.png"}
```

Pour rappel, Copix va chercher la ressource img/test.png, dans l'ordre et par priorité, dans les emplacements suivants :

- www/themes/nom_theme_courant/img/test.png
- www/themes/default/img/test.png
- www/img/test.png

Paramètre assign

Le paramètre assign permet de demander à assigner le retour de la balise à une variable de template plutôt que de l'afficher directement.

```
{copixresource path="img/test.png" assign=nomVariable}
{* Plus loin, utilisation de la variable *}
{$nomVariable}
```

Tag Copixtips

Présentation

CopixTips est un tag dont nous nous servons en interne (dans le module d'administration par exemple) pour afficher des listes de trucs et astuces d'importance diverses.

Tag Copixurl

Présentation

Le tag {copixurl} est destiné aux utilisateurs de Smarty pour qu'ils puissent générer des url de type Copix à l'intérieur de leurs templates.

Cette balise converti les appels en **CopixUrl**::get (xxx).

Cette balise n'est pas disponible via "_tag" du fait que cela ne présente aucun intérêt face à **CopixUrl::get** ou la fonction **raccourci** _url.

ATTENTION : Le tag {copixurl} retourne l'url avec les caractères spéciaux HTML échapés (par exemple & devient &) alors que _url et CopixUrl::get () retournent l'url sans ces échappements. Vous pouvez utiliser le paramètre notxml=true dans le tag {copixurl} pour avoir le même comportement que CopixUrl::get. De la même façon, vous pouvez demander à CopixUrl:get d'échapper les caractères avec le troisième paramètre (optionnel) à true.

Exemple d'utilisation

```
{* équivalent de CopixUrl::get () *}
{copixurl}

{* équivalent de CopixUrl::get ('#') *}
{copixurl dest="#"}

{* équivalent de CopixUrl::get ('module|groupe|action') *}
{copixurl dest="module|groupe|action"}

{* équivalent de CopixUrl::get ('module|groupe|action', array ('parametre'=>1)) *}
{copixurl dest="module|groupe|action" parametre=1}

{* équivalent de CopixUrl::get ('module|groupe|action', array ('parametre'=>1,
'parametre2'=>2)) *}
{copixurl dest="module|groupe|action" parametre=1 parametre2=2}
```

Assignation de copixurl dans une variable JavaScript

Pour que la page soit validée par le W3C, le tag copixurl écrit les caractères "&" (séparateur de paramètre) avec "&". Ceci est totalement transparent lors d'une utilisation du tag dans un <a href>, mais lors d'une assignation dans une variable JavaScript, cela pose problème, car les "&" ne seront pas compris comme des séparateurs de paramètres, mais comme faisant partie intégrante de la chaine de caractère.

Pour remédier à ce problème, il existe un paramètre : notxml=true, à ajouter dans le tag :

```
{literal}
<script type="text/javascript">
var monUrl = '{/literal}{copixurl dest="module|actionGroup|action"
monParam1="maValeur" monParam2="maValeur2" notxml=true}{literal}';
</script>
{/literal}
```

Tag Copixzone

Présentation

CopixZone est un tag essentiellement destiné à permettre aux designers de templates d'appeler des zones depuis Smarty.

Exemple Smarty

```
{copixzone process=idZone param1=value param2= value2}

{* équivaut à l'appel PHP *}
{* CopixZone::process ('idZone', array ('param1'=>'value', 'param2'=>'value2')); *}
```

Paramètres supplémentaires

assign

Ce paramètre permet d'assigner le retour de la zone à une variable de template plutôt que de l'afficher directement. Ce paramètre n'est disponible que dans la version Smarty du tag.

```
{copixzone process=idZone param1=value param2= value2 assign=variable}

{* plus loin *}
{$variable}
```

required

Ce paramètre permet de rendre la zone facultative. Si required vaut false et que le module dont la zone fait parti n'existe pas, alors aucune erreur ne sera générée lors de l'appel du tag.

```
{* si le module modulePasInstalle n'est pas installé, alors rien ne sera affiché ici *}
{copixzone process="modulePasInstalle|idZone" required=false}
```

NOTE : Le paramètre required n'agit que sur la présence du MODULE auquel appartient la zone, et pas sur la zone en elle même. Si vous spécifiez une zone qui n'existe pas dans un module actif, une erreur sera tout de même générée.

ajax

Ce paramètre permet de charger le contenu de la zone en Ajax. Ce paramètre peut entrainer des paramètres qui lui son propre.

{copixzone process="module|zoneAjax" ajax=true}

id

Cette id permet d'identifier cette zone. le but etant après le chargement de la page de pouvoir faire

```
//Charge le contenu du div sans l'afficher (le div est en display:none)
$('monId').fireEvent('load');

//Charge si besoin et affiche le div
$('monId').fireEvent('display');
```

auto

En le mettant a true, ce paramètre permet de charger et d'afficher le contenu de la zone directement après que l'arborescence DOM est fini de se charger.

text

Ce paramètre permet d'afficher un text qui au click affichera ou cachera le contenu de la zone (en chargeant le contenu au premier affichage)

idClick

Ce paramètre permet de définir un objet (par son id) qui sera clickable pour afficher cacher la zone

onComplete

Correspond à un code javascript a executer au moment du onComplete de resultat Ajax

onHide

Un code javascript effectuer au moment ou le div se cache

onDisplay

Un code javascript effectuer au moment ou le div s'affiche

NOTE : il existe 3 evenements que le div peut catcher

- load Charge le contenu
- display Affiche le contenu (et le charge si ce n'est pas deja fait)
- hide Cache le contenu

Tag Cycle

Présentation

Le tag cycle vous permet d'afficher successivement des données parmi une liste de valeurs. C'est une balise très pratique pour par exemple alterner les couleurs des lignes d'un tableau.

Forme Smarty

Se référer à la **documentation Smarty officielle**.

Forme PHP

La forme PHP reprends l'ensemble des paramètres de la forme Smarty, avec la syntaxe habituelle.

```
_tag ('cycle', array ('values'=>"#eeeeee,#d0d0d0"));
```

Tag Errormsg

Présentation

Le tag errormsg est généralement utilisé pour afficher un message d'erreur.

Exemple Smarty

```
{errormsg message="Grosse Erreur !" class="classeErreur"}
```

Exemple PHP

```
<?php _etag ('errormsg', array ('message'=>"Grosse Erreur !")); ?>
```

Paramètres

message

Le paramètre message (obligatoire) permet de spécifier le message à afficher avec un style CSS par défaut.

Par exemple :

```
{errormsg message="Grosse Erreur !" }
```

Produira le code suivant :

```
<p  style="color: #FF2222;font-weight:bold;">Grosse Erreur !</p>
```

class

Le paramètre class permet de préciser le nom de la classe CSS à appliquer au message d'erreur.

```
{errormsg message="Grosse Erreur !" class="classeErreur"}
```

Produira le code suivant :

```
<p class="classeErreur">Grosse Erreur !</p>
```

assign

Le paramètre assign permet d'indiquer le nom de la variable à laquelle sera assigné le code HTML produit par Smarty. Attention, en précisant ce paramètre, le message ne sera pas affichée mais seulement assigné à une variable.

```
{errormsg message="Grosse Erreur !" assign="maVariable"}

{* et si l'on souhaite ensuite l'utiliser *}
{$maVariable}
```

Tag Escape

Présentation

Le modificateur "escape" permet de convertir les caractères spéciaux en leur équivalent HTML (accents, <, >, ...).

C'est un alias à la fonction html_entities qui prendra en compte le jeu de caractères configuré.

Forme Smarty

```
{* si $var contient é, affichera &eacute; *}
{$var|escape}
```

Forme PHP

```
// si $var contient é, affichera &eacute;
_eTag ('escape', $var);
```

Tag Formfocus

Présentation

Le tag formfocus vous permet de donner le focus à un élément particulier de votre formulaire.

Vous devez pour cela spécifier le paramètre "id" qui correspondra à l'id de l'élément de formulaire à qui donner le focus.

Exemple en PHP

```
<input type="text" id="premier">
<input type="text" id="second">
<!-- on va donner le focus par défaut au deuxième élément du formulaire -->
<?php _eTag ('formfocus', array ('id'=>'second')); ?>
```

Exemple avec Smarty

```
<input type="text" id="premier">
<input type="text" id="second">
{* on va donner le focus par défaut au deuxième élément du formulaire *}
{formfocus id='second'}
```

Tag Htmleditor

Présentation

Le tag htmleditor permet de proposer à l'internaute un éditeur WYSIWYG pour des contenus textes.

Tag i18n

Présentation

Le tag i18n permet d'afficher une chaîne localisée en fonction de la langue courante du navigateur. Pour plus d'informations, consultez la documentation sur CopixI18N.

Exemple PHP

```
<?php _etag ('i18n', array ('key'=>'monModule|monModule.maCle')); ?>
```

Exemple Smarty

```
{i18n key="monModule|monModule.maCle" lang="fr"}
```

Paramètres

key

Le paramètre key (obligatoire) permet de spécifier la clé correspondante à la chaîne localisée requise.

```
{i18n key="monModule|monModule.maCle"}
```

lang

Le paramètre lang permet de préciser la langue (ou langue / pays) dans laquelle doit être récupérée la chaîne.

```
{i18n key="monModule|monModule.maCle" lang="fr"}

{* il est aussi possible de spécifier le pays dans ce paramètre *}
{i18n key="monModule|monModule.maCle" lang="en_US"}
```

assign

Le paramètre assign permet d'indiquer le nom de la variable à laquelle sera assignée la chaîne. Attention, en précisant ce paramètre, la chaîne ne sera pas affichée mais seulement assignée à une variable. Ce paramètre n'est disponible que dans la version Smarty.

```
{i18n key="monModule|monModule.maCle" assign="maVariable"}

{* et si l'on souhaite ensuite l'utiliser *}
{$maVariable}
```

noEscape

Le paramètre noEscape permet d'éviter la transformation html_entities effectuée par défaut sur la chaîne récupérée.

Par exemple, avec la clef suivante :

```
monModule.maCle = 18 < 20 ? oui !
```

```
{i18n key="monModule|monModule.maCle" lang="fr"}
{* génère 18 &lt; 20 ? oui ! *}

{i18n key="monModule|monModule.maCle" lang="fr" noEscape=1}
{* génère 18 < 20 ? oui ! *}
```

Cas d'une chaîne paramétrée

Vous pouvez préciser d'autres paramètres afin de personnaliser à la volée une chaîne paramétrée.

Par exemple, avec le fichier monModule.properties suivant :

```
monModule.maCle = Il est %s heure %s
```

On utilisera

```
{i18n key="monModule|monModule.maCle" lang="fr" param1="9" param2="30"}
```

Pour afficher :

```
Il est 9 heure 30
```

Tag Inputtext

Présentation

Le tag inputtext permet d'afficher une champ texte HTML (balise < input type="text" >)

Exemple Smarty

```
{inputtext id="monChamp" value=$valeur}
{inputtext id="monChamp" value="valeur par defaut"}
```

Paramètres

défini par l'utilisateur

Vous pouvez spécifier n'importe quel autre paramètre HTML pour ce champ texte.

Par exemple :

```
{inputtext id="monChamp" value=$ppo->valeur}
```

Produira le code suivant :

```
    <input text id="monChamp" name="monChamp" value="valeur de $ppo->valeur" />
```

maxlength

Le paramètre maxlength permet de définir la longueur maximale du champ (en caractères).

Par exemple :

```
  {inputtext id="monChamp" maxlength="3"}
```

Produira le code suivant :

```
    <input type="text" id="monChamp" name="monChamp" maxlength="3"  />
```

next et previous

Le paramètre next permet de définir le champ suivant au niveau du focus tandis que le paramètre previous permet de définir le champ précédent au niveau du focus. Le paramètre next doit obligatoirement être accompagné du paramètre maxlength pour fonctionner.

Par exemple :

```
  {inputtext id="monChamp2" maxlength="3" previous="monChamp1" next="monChamp3"}
```

Produira le code suivant :

```
    <input type="text" id="monChamp2" name="monChamp2" maxlength="3"
onKeyDown="javascript:focusid(this,3,event,'monChamp3','monChamp2');"  />

    <!-- + Fonctions JavaScript de contrôle -->
```

assign

Le paramètre assign permet d'indiquer le nom de la variable à laquelle sera assigné le code HTML produit par Smarty. Attention, en précisant ce paramètre, le champ ne sera pas affichée mais seulement assigné à une variable.

```
  {inputtext id="monChamp" assign="maVariable"}

  {* et si l'on souhaite ensuite l'utiliser *}
  {$maVariable}
```

name & id

Les paramètres name et id servent à spécifier une valeur pour les paramètres name et id de l'input généré. L'un ou l'autre doit être donné.

Si id est donné mais pas name, alors name aura la valeur de id. Si name est donné mais pas id, alors id aura la valeur de name. Si les deux paramètres sont donnés, alors ils garderont bien sûr leur valeur propre.

```
{inputtext id="monChamp"}
//génère <input type="text" id="monChamp" name="monChamp" />

{inputtext name="monAutreChamp"}
//génère <input type="text" id="monAutreChamp" name="monAutreChamp" />

{inputtext name="champName" id="champId"}
//génère <input type="text" id="champId" name="champName" />
```

Tag Linkbar

Présentation

Le tag linkbar permet de générer facilement une barre de navigation numérotée, telle qu'on peut en voir lorsque vous souhaitez permettre à un utilisateur de naviguer au travers de plusieurs enregistrements dans une très grande liste.

Exemple d'utilisation Smarty

```
{linkbar url="http://www.copix.org/index.php/list" nbLink=5 pageNum=$ppo->pageNum
          ✂ nbTotalPage=$ppo->nbPage}
```

Tag Mootools

Présentation

Le tag Mootools vous permet de demander à Copix d'inclure les bibliothèques Mootools avec / sans plugins.

Ce tag accepte un paramètre "plugin" qui peut être un tableau ou une chaine de caractère, qui indique les plugins mootools que vous souhaitez inclure.

Ce tag utilise CopixHTMLHeader en interne pour réaliser l'inclusion.

Si vous appelez plusieurs fois le tag, les inclusions ne seront réalisées qu'une seule fois.

Forme Smarty

```
{* sans plugin *}
{mootools}

{* avec des plugins *}
{mootools plugin="pngfix;elementmover;transcorners;toolbar"}
```

Forme PHP

```
//avec une chaine dans plugin
_tag ('mootools', array ('plugin'=>'pngfix;elementmover;transcorners;toolbar'));
//avec un tableau
_tag ('mootools', array ('plugin'=>array ('pngfix', 'elementmover', 'transcorners',
            ✂ 'toolbar')));
```

Tag Radiobutton

Présentation

Le tag radiobutton permet d'afficher un groupe de boutons radios (balise < input type="radio" />)

Exemple Smarty

```
   {radiobutton name="monBouton" values="1=>OUI;2=>NON"|toarray}
```

Exemple PHP

```
<?php _eTag ('radiobutton', array ('name'=>'monBouton', 'values'=>array ('oui',
            ✂ 'non'))); ?>
```

Paramètres

name

Le paramètre (obligatoire) name permet de définir le nom du groupe de boutons radios.

values

Le paramètre values permet à partir d'un tableau clé/valeur de définir la valeur et le label respectifs de chacun des boutons radios du groupe.

Par exemple :

```
{radiobutton name="monBouton" values="1=>OUI;2=>NON"|toarray}
```

Produira le code suivant :

```
<input type="radio" name="monBouton"  value="1" />OUI
<input type="radio" name="monBouton"  value="2" />NON
```

Voir aussi : La documentation du **modificateur toarray**.

objectMap

Si vous disposez d'un tableau d'objets pour définir le couple valeur/label de chacun des boutons radios, il vous suffit de le passer au paramètre values. Il est alors nécessaire de spécifier à quels attributs correspondent ces deux informations. C'est le rôle du paramètre objectMap.

Le paramètre objectMap permet de définir le mappage "attributValeur;attributLabel" à appliquer à l'objet reçu dans le paramètre values.

Par exemple, avec l'objet suivant :

```
class MonObjetRadio {
    var $valeur;
    var $label;

    public function __construct($valeur, $label) {
            $this->valeur = $valeur;
            $this->label = $label;
    }
}

//A passer au template
$ppo->monObjet = array(new MonObjetRadio("1", "Oui"), new MonObjetRadio("2",
        ✂ "Non"));
```

Le code Smarty suivant:

```
{radiobutton name="monBouton" values=$ppo->monObjet objectMap="valeur;label"}
```

Produira le code suivant :

```
<input type="radio" name="monBouton"  value="1" />Oui
<input type="radio" name="monBouton"  value="2" />Non
```

selected

Le paramètre selected permet de définir l'identifiant de l'élément pré-sélectionné par défaut. Cet identifiant s'applique à une valeurs définies via le paramètre values.

Par exemple :

```
{radiobutton name="monBouton" values="1=>OUI;2=>NON"|toarray selected="2"}
```

Produira le code suivant :

```
<input type="radio" name="monBouton"  value="1" />OUI
<input type="radio" name="monBouton"  value="2" checked="checked"  />NON
```

extra

Le paramètre extra sert à définir un texte libre qui sera inséré dans le code de la balise HTML produit par Smarty. Cela vous permet donc de rajouter à votre guise des paramètres supplémentaires à votre groupe de boutons radios comme un **id**. Dans notre exemple, nous spécifions une classe CSS. Notez cependant que la plupart des styles ne fonctionnent pas très bien avec les boutons radio.

Par exemple :

```
{radiobutton name="monBouton" values="1=>OUI;2=>NON"|toarray
          ✂ extra='class="classeRadio"'}
```

Produira le code suivant :

```
<input type="radio" name="monBouton" class="classeRadio" value="1" />OUI
<input type="radio" name="monBouton" class="classeRadio" value="2" />NON
```

assign

Le paramètre assign permet d'indiquer le nom de la variable à laquelle sera assigné le code HTML produit par Smarty. Attention, en précisant ce paramètre, le bouton ne sera pas affiché mais seulement assigné à une variable.

```
{radiobutton name="monBouton" assign="maVariable"}

{* et si l'on souhaite ensuite l'utiliser *}
{$maVariable}
```

Voir aussi

- Le tag select

Tag Select

Présentation

Le tag select vous permet de générer une combobox (select/options) facilement à partir d'un tableau de valeurs ou d'un tableau d'objets.

Exemple Smarty

```
{select name="champ" values=$arData selected=1}
```

Exemple PHP

```
_eTag ('select', array ('name'=>"champ", 'values'=>$arData 'selected'=>1));
```

Présentation des paramètres supplémentaires

Spécifier les clefs/valeurs

```
{select name=select values="1=>MySQL;2=>Postgres;3=>SQLite;4=>SQLServer;
           ✂ 5=>Oracle"|toarray}
```

Indiquer l'élément sélectionné avec selected

```
{select name=select selected=2 values="1=>MySQL;2=>Postgres;3=>SQLite;4=>SQLServer;
           ✂ 5=>Oracle"|toarray}
```

Changer le libellé de la valeur vide

```
{select emptyValues="--Aucun--" name=select values="1;2;3;4;5"|toarray}
```

Changer le libellé et la valeur de la valeur vide

```
{select emptyValues="KO=>--Aucun--" name=select values="1;2;3;4;5"|toarray}
```

Ne pas afficher de valeur vide

```
{select emptyShow=false name=select values="1;2;3;4;5"|toarray}
```

Spécifier un id différent du name

```
{select name=select id=autrechose values="1;2;3;4;5"|toarray}
```

Utiliser un tableau d'objet et spécifier les clefs / valeurs

Ici nous avons un tableau d'objet avec les propriétés id/caption. Nous allons indiquer à la balise que id est la valeur de l'option et que caption est son libellé dans le paramètre objectMap.

le tableau d'objet est déclaré comme suit :

```
$arObjects = array ();

$obj = new StdClass ();
$obj->id = '1';
$obj->caption = 'libellé 1';
$arObjects[] = $obj;

$obj = new StdClass ();
$obj->id = '2';
$obj->caption = 'libellé 2';
$arObjects[] = $obj;

//On passe le tableau au template via PPO
$ppo->arObjects = $arObjects;
```

```
{select name=select id=autrechose values=$ppo->arObjects objectMap="id;caption"}
```

Paramètre extra pour rajouter des informations à la balise

```
{select extra='style="background-color: #ccc;"' name=select values="1;2;3;4;5"|toarray}
```

(spécifique à Smarty) assigner le retour dans une variable

Il existe des cas ou vous pouvez souhaiter ne pas afficher directement le select et mettre le retour du tag dans une variable, que vous utiliserez plus tard.

Utilisez le paramètre "assign".

```
{select name="champ" values=$arData assign=variable}
{if $condition}
 {$variable}{* on affiche la sélection uniquement si $condition *}
{/if}
```

Tag Multipleselect

Présentation

Le tag multipleselect vous permet de générer une liste qui ressemble a la combobox select mais qui permet de cocher plusieurs valeurs

Exemple Smarty

```
{mulipleselect name="champ" values=$arData selected=1}
```

Exemple PHP

```
_eTag ('multipleselect', array ('name'=>"champ", 'values'=>$arData 'selected'=>1));
```

Présentation des paramètres supplémentaires

Spécifier les clefs/valeurs

```
{multipleselect name=select values="1=>MySQL;2=>Postgres;3=>SQLite;4=>SQLServer;
            ✂ 5=>Oracle"|toarray}
```

Indiquer l'élément sélectionné avec selected

```
{mulipleselect name=select selected=2
            ✂ values="1=>MySQL;2=>Postgres;3=>SQLite;4=>SQLServer;5=>Oracle"|toarray}
{mulipleselect name=select selected="2;3;4"|toarray
            ✂ values="1=>MySQL;2=>Postgres;3=>SQLite;4=>SQLServer;5=>Oracle"|toarray}
```

Spécifier un id différent du name

```
{multipleselect name=select id=autrechose values="1;2;3;4;5"|toarray}
```

Utiliser un tableau d'objet et spécifier les clefs / valeurs

Ici nous avons un tableau d'objet avec les propriétés id/caption. Nous allons indiquer à la balise que id est la valeur de l'option et que caption est son libellé dans le paramètre objectMap.

le tableau d'objet est déclaré comme suit :

```
$arObjects = array ();

$obj = new StdClass ();
$obj->id = '1';
$obj->caption = 'libellé 1';
$arObjects[] = $obj;

$obj = new StdClass ();
$obj->id = '2';
$obj->caption = 'libellé 2';
$arObjects[] = $obj;

//On passe le tableau au template via PPO
$ppo->arObjects = $arObjects;
```

```
{multipleselect name=select id=autrechose values=$ppo->arObjects
            ✂ objectMap="id;caption"}
```

Paramètre extra pour rajouter des informations à la balise

```
{multipleselect extra='style="background-color: #ccc;"' name=select
            ✂ values="1;2;3;4;5"|toarray}
```

(spécifique à Smarty) assigner le retour dans une variable

Il existe des cas ou vous pouvez souhaiter ne pas afficher directement le select et mettre le retour du tag dans une variable, que vous utiliserez plus tard.

Utilisez le paramètre "assign".

```
{multipleselect name="champ" values=$arData assign=variable}
{if $condition}
 {$variable}{* on affiche la sélection uniquement si $condition *}
{/if}
```

Tag Ulli

Présentation

ulli est un tag qui permet de générer facilement une liste de type ul / li à partir d'un tableau de valeurs.

Exemple Smarty

```
{ulli values="premier;second;troisième"|toarray}
```

Qui va générer

```
<ul>
 <li>premier</li>
 <li>second</li>
 <li>troisième</li>
</ul>
```

Exemple PHP

Avec un tableau de tableaux

```
$values = array ('1', array ('21', '22', '23'), '3');
_eTag ('ulli', array ('values'=>$values));
```

Qui va générer

```
<ul>
 <li>1</li>
 <ul>
  <li>21</li>
  <li>22</li>
  <li>23</li>
 </ul>
 <li>3</li>
</ul>
```

Tag Datei18n

Présentation

Ce modificateur de variable, disponible uniquement dans les templates Smarty, transforme une date au format YYYYMMDD en Date locale.

Exemple

```
{* avec $monDateTime = 20070201 *}
{$monDateTime|datei18n}
{* va afficher 01/02/2007 *}
```

Il est possible de demander à afficher la date/heure au format texte, exemple

```
{* avec $monDateTime = 20070720 *}
{$monDateTime|datei18n:text}
{* va afficher vendredi 20 juillet 2007 *}
```

Voir aussi

- CopixDateTime

Tag Datetimei18n

Présentation

Ce modificateur de variable, disponible uniquement dans les templates Smarty, transforme une date au format YYYYMMDDHHIISS en DateTime local.

Exemple

```
{* avec $monDateTime = 20070201143015 *}
{$monDateTime|datetimei18n}
{* {* va afficher 01/02/2007 14:30:15*}
```

Il est possible de demander à afficher la date/heure au format texte, exemple

```
{* avec $monDateTime = 20070720152700 *}
{$monDateTime|datetimei18n:text}
{* va afficher Vendredi 20 Juillet 2007 15:27:00 *}
```

Voir aussi

- CopixDateTime

Tag Hour_format

Présentation

La balise hour_format est un modificateur de variable destiné à être utilisé dans les templates Smarty uniquement. Son rôle est de convertir une chaîne de caractères au format "HHMMSS" dans un format spécifié en paramètre.

Le format attendu devra comporter les champs :

- "%H" pour les heures
- "%i" pour les minutes
- "%s" pour les secondes

Par défaut, il sera **"%H:%i:%s"**.

Par exemple, le code Smarty suivant :

```
{assign var="heure" value="153020"}
{$heure|hour_format:"%H:%i:%s"}
```

Produira le code HTML suivant :

```
15:30:20
```

Tag Time

Présentation

La balise time est un modificateur de variable destiné à être utilisé dans les templates Smarty uniquement. Son rôle est de convertir une chaîne de caractères du format "HHMMSS" au format Heure établi par la langue courante ("HH:MM:SS" en France).

Par exemple, le code Smarty suivant :

```
{assign var="heure" value="153020"}
{$heure|time}
```

Produira le code HTML suivant :

```
15:30:20
```

Si la chaîne initiale est mal formatée, elle ne sera pas transformée.

Voir aussi

Pour plus d'informations, consultez la documentation sur CopixDateTime et plus particulièrement sa méthode **CopixDateTime::hhmmssToTime**.

Tag Toarray

Présentation

La balise toarray est un modificateur de variable destiné à être utilisé dans les templates Smarty uniquement.

Son rôle est de convertir une chaîne de caractères en tableau, si elle respecte un format particulier.

exemple

```
{select name=selectbox values="1=>OUI;2=>NON"|toarray}
```

Format de la chaine

- Chaque élément du tableau doit être séparé par un point virgule (;).
- Vous pouvez spécifier une clef pour chaque élément, pour cela il faut écrire clef=>valeur pour l'élément.

exemples

```
{select name=selectbox values="OUI;NON"|toarray}

{select name=selectbox values="2=>NON;1=>OUI"|toarray}

{select name=selectbox values="OUI;2=>NON;PEUT-ETRE;6=>ON VERRA"|toarray}
```

Tag url

Présentation

La balise url est un modificateur de variable destiné à être utilisé dans les templates Smarty uniquement. Son rôle est de convertir une chaîne de caractères en url.

Par exemple, le code Smarty suivant :

```
{assign var="url" value="http://www.copix.org"}
{$url|url}
```

```
<a href="http://www.copix.org">http://www.copix.org</a>
```

Produira le code HTML suivant :

Formatage optionnel

Il est également possible de préciser le texte qui recevra le lien, celui-ci pouvant donc être différent.

Par exemple, le code Smarty suivant :

```
{assign var="url" value="Copix|http://www.copix.org"}
{$url|url}
```

Produira le code HTML suivant :

```
<a href="http://www.copix.org">Copix</a>
```

Webservices

WSServer

Introduction

Cette page est destiné à décrire en détail le module WSServer qui a pour but de vous fournir une interface pour exporter n'importe quelle classes de vos modules en Web Services.

Note : Si vous n'êtes pas familier avec SOAP où que vous avez quelques questions concernant l'implémentation de Web Services en PHP, nous vous invitons à consulter la page concernant les **Webservices et PHP**.

L'extension SOAP ne permettant pas de générer automatiquement les fichiers WDSL, le module intègre la classe WSDL_Gen.php pour ce faire. Celle-ci se base sur les commentaires pour identifier les types des paramètres et des retour des fonctions. Veillez donc à bien commencer vos fonction de la manièdre suivante :

```
/**
 * @param int $pParamInt
 * @param int $pParamInt
 * @return string
 */

function ExempleParams ($pParamInt, $pParamString) {
    /**
     * Code
     */
    return "OK";
}
```

Installation du module

Condition préalable : votre serveur doit disposer de l'extension SOAP de PHP5. L'installation est classique. Le module installe une table wsservices qui contient les infos sur les services exportés.

Architecture

Le module se trouve dans project/modules/public/stable/tools/wsserver/

Paramètres du module

- exportedModule : Nom du module Copix contenant la classe à exporter par défaut
- exportedClassFile : Nom du fichier contenant la classe à exporter par défaut
- exportedClass : Nom de la classe à exporter par défaut.

Administration

Il est possible de configurer le module de deux manières complémentaires

La première consiste à modifier les paramètres du module pour exporter une classe. Dans ce cas le fichier wsdl est disponible à l'adresse suivante : http://www.votresite.com/index.php/wsserver/default/wsdl/

L'adresse du serveur SOAP est : http://www.votresite.com/index.php/wsserver/

Cette première méthode vous permettra facilement d'exporter une classe. Elle est recommandé dans le cas d'un web service ponctuel.

La deuxième, plus élaborée, va vous permettre d'exporter les classes de vos modules. Pour celà rendez-vous sur l'interface d'administration et cliquez sur "Gestion des exportations WS". Vous obtenez alors la liste complète des Modules installés sur votre serveur et les classes leur appartenant.

Cliquez ensuite sur la classe que vous compter exporter. Par exemple : la classe sample.services.class.php du module "Mise en place d'un serveur SOAP".

Vous obtenez alors un formulaire prérempli, contenant les informations sur la classe que vous voulez exporter. Indiquez alors le nom que vous voulez donner à votre service.

Exemple : SampleWebService. Cliquez sur valider. Votre web service est créé et vous obtenez les infos récapitulatives sur votre Web Services, à savoir l'URL du Web Services et l'URL du fichier WSDL. Exemple

```
Url du Webservice :
http://localhost/alptis/www/index.php/wsserver/default/default/SampleWebService
Url du fichier wsdl :
http://localhost/alptis/www/index.php/wsserver/default/wsdl/SampleWebService
```

A partir de ces informations vous pourrez créer votre Client SOAP.

Outils

CopixEMail

Objectifs

Ici, nous allons voir comment envoyer des mails au format texte ou au format HTML, ainsi que la façon d'y adjoindre des pièces jointes.

Envoyer un mail au format texte avec CopixTextEMail

```
$message = ... // une chaîne de caractères
$spam_texte = new CopixTextEmail($destinataire,
                                 $copie,
                                 $copiecachee,
                                 $sujet,
                                 $message);
```

Envoyer un mail au format HTML

```
$destinataire = "admin@copix.org";
$copie = "team@copix.org";
$copiecachee = "uncopain@ailleurs.net";
$sujet = "Je veux la version 4 dans 2 jours";

// on veut l'envoyer au format HTML
$message = ... // une chaîne de caractères au format HTML
$remplacement = ... // une autre chaîne de caractères si le mailer ne supporte pas le
            ✂ HTML

// le paramètre $remplacement est optionnel
$spam_html = new CopixHTMLEmail($destinataire,
                                $copie,
                                $copiecachee,
                                $sujet,
                                $message,
                                $remplacement);
```

Ajouter des pièces jointes

Maintenant que le courriel est fait, nous pouvons avoir besoin de pièces jointes. Pour cela, il existe la méthode addAttachment. Cette méthode prend les données, un nom de fichier, le type mime et un encodage.

```
// on veut envoyer des données brutes. Par défaut, addAttachement donne un nom de
fichier vide, un type mime application/octet-stream et un encodage base64.
$spam_texte->addAttachement($données);

// si par contre, on veut envoyer une image
$nomFichier = 'toto.gif';
$mimeType = 'image/gif';
$encoding = 'base64';
$spam_texte->addAttachement($image, $nomFichier, $mimeType, $encoding);
```

Envoi du message proprement dit

```
$spam_texte->send ();
```

Si vous voulez envoyer le message à d'autres expéditeurs que ceux annoncés au départ lors de la création de CopixHTMLEmail ou CopixTextEMail, vous pouvez les spécifier lors de l'envoi

```
$spam_texte->send('le patron', // le nom de l'expéditeur
                  'jamaiscontent@direction.copix.org' // son adresse
);
```

Configuration de Copix Mail

Copix va récupérer les informations dont il a besoin dans le module *default*. Ce dernier contient dans son répertoire *resources* un fichier nommé *parameter.properties*. Ainsi, si dans la méthode *send*, expliquée plus haut, vous ne donnez pas le nom de l'expéditeur et/ou son adresse ne sont pas précisés, Copix va aller chercher les chercher dans ce fichier.

parameter.mailEnabled

Indique à Copix si l'envoi de courriel est autorisé depuis le site. Si il vaut 1, c'est le cas. Toute autre valeur signifie une interdiction d'envoi.

parameter.mailFrom

Indique l'adresse de l'expéditeur du courrier envoyé par l'application. Exemple

```
parameter.mailFrom = admin@monsite.eu
```

parameter.mailFromName

Indique le nom de l'expéditeur du courrier envoyé par l'application. Exemple

```
parameter.mailFromName = Jean Claude Dupont
```

parameter.mailMethod

Indique la méthode souhaitée pour l'envoi du courriel. Vous avez le choix entre 2 possibilités:

- **mail:** le courriel est envoyé avec la méthode mail de php.
- **smtp:** le courriel est envoyé à travers un serveur smtp. Dans ce cas, il faut préciser aussi le paramètre *mailSmtpHost*.

parameter.mailSmtpHost

Indique l'adresse du serveur smtp qui servira à l'envoi du courriel si la méthode choisie dans le paramètre *mailMethod* est smtp.

CopixCache

Rôle

CopixCache est la classe de base de Copix qui vous permet de sauvegarder des informations afin de les ressortir en un temps éclair lorsque vous en aurez de nouveau besoin.

Il est possible de paramétrer plusieurs caches avec plusieurs stratégies de sauvegardes.

CopixCache peut en standard fonctionner avec APC, le système de fichier, ou simplement se comporter comme une variable globale à la page.

Pour chaque cache, il est possible de déterminer des durées de vie ainsi que des sous type.

Utilisation

Ecrire dans le cache

```
CopixCache::write ($id, $content, $type);
```

- $id correspond à l'identifiant de l'élément à stocker dans le cache. L'identifiant peut être un tableau, un objet, une chaine, ...
- $content est le contenu de l'élément à mettre en cache. Le contenu peut être d'un type quelconque.
- $type est le type de cache ou l'on veut stocker notre élément. Le type est facultatif, si vous ne le précisez pas, le type 'default' sera utilisé.

Exemples

```
//Sauvegarde du résultat d'une requête
CopixCache::write ($filtre, _dao ('ma_table')->findBy ($filtre));

//Sauvegarde du contenu d'une page web distante
CopixCache::write ('http://www.copix.org', file_get_contents
('http://www.copix.org'));

//Sauvegarde d'une donnée dans le cache de type "data" (qui devra être configuré)
CopixCache::write ('data_1', $monObjet->get ('data_1'), 'data');
```

Tester l'existence d'un élément dans le cache

```
CopixCache::exists ($id, $type);
```

- $id correspond à l'identifiant de l'élément dont on vérifie la présence. L'identifiant peut être un tableau, un objet, une chaine, ...
- $type est le type de cache dans lequel on vérifie la présence de l'élément. Le type est facultatif, si vous ne le précisez pas, le type 'default' sera utilisé.

La méthode retourne true ou false.

Exemple

```
if (!CopixCache::exists ($filtre)){
   //traitement
}
```

Lire depuis le cache

```
$my_content = CopixCache::read ($id, $type);
```

- $id correspond à l'identifiant de l'élément à récupérer depuis le cache. L'identifiant peut être un tableau, un objet, une chaine, ...
- $type est le type de cache depuis lequel on veut récupérer l'élément. Le type est facultatif, si vous ne le précisez pas, le type 'default' sera utilisé.

Effacer les éléments d'un cache

```
CopixCache::clear ($id, $type);
```

- $id correspond à l'identifiant de l'élément à supprimer du cache. L'identifiant peut être un tableau, un objet, une chaine, ... Si vous souhaitez supprimer l'ensemble des éléments du cache, mettez "null".
- $type est le type de cache duquel on souhaite supprimer l'élément. Le type est facultatif, si vous ne le précisez pas, le type 'default' sera utilisé.

Configuration

Pour configurer les types de cache, il suffit d'utiliser l'interface de configuration du module admin nommée "administration des types de cache".

Les stratégies de Cache

CopixCache peut utiliser plusieurs types de stratégies pour écrire/lire dans le cache. En général, une stratégie est un moyen / lieu ou sauvegarder les éléments.

Les stratégies standard

APC

La stratégie APC utilise l'extension PHP (PECL) APC.

Concrètement, elle utilise les méthodes apc_store et apc_fetch pour écrire / lire depuis le cache. C'est une méthode particulièrement efficace si vous utilisez cette extension sur votre serveur.

Nous recommandons l'utilisation de cette stratégie pour des données volatiles et peu volumineuses.

File

La stratégie File utilise le système de fichier pour sauvegarder les données dans le cache.

Les fichiers seront sauvegardés dans le répertoire désigné par **COPIX_CACHE_PATH** soit en standard temp/cache/*.

Il existera un répertoire par type de cache.

Nous recommandons l'utilisation de cette stratégie pour des données à longue persistance ou volumineuses.

System

La stratégie System utilise la mémoire du script pour sauvegarder ses données. Ainsi, toute information mise dans le cache avec la stratégie System est perdue à la fin du script.

Nous recommandons l'utilisation de cette stratégie pour des pré-calculs de données qui peuvent être utilisées au travers de plusieurs objets de façon asynchrones.

Creation d'une stratégie

Vous pouvez egalement créer des strategies de cache.

Pour cela, créez un module contenant une classe utilisant l'interface CopixCacheApi

Vous devrez ensuite des déclarer les 6 méthodes obligaroires (write, read, clear, exists, isEnabled et le constructeur __construct)

Votre strategie pourra être configurée dans un de vos cache en mettant la strategie sous la forme 'module|classeAdaptator'

Voir aussi

- Tutoriel sur le Cache

CopixZone

Introduction

Les Zones dans Copix sont des objets dont l'objectif est de prendre en charge une partie d'écran, simple et autonome qui peuvent être réutilisés dans un module, dans un même projet ou même dans un autre projet.

On peut vouloir faire des zones pour créer des composants graphiques réutilisables, que l'on utilisera un peu partout dans un site (exemples: Un zone de login, une zone de recherche, une zone des dernières nouvelles, une zone de syndication, ...)

Les zones peuvent nativement tirer parti du cache.

Classes & Emplacement

Les zones héritent de *CopixZone*, elles sont stockées dans des fichiers *nomzone.zone.php* dans les répertoires *zones* des modules et se nomment *ZoneNom*

Exemple

```
//dans le fichier zones/exemple.zone.php
class ZoneExemple extends CopixZone {
   function _createContent (&$toReturn){
      $toReturn = 'Contenu de ma zone';
      return true;
   }
}
```

La fonction _createContent

La fonction "_createContent" est en charge de la génération du contenu de la zone. Elle reçoit en paramètre une chaîne de caractères (par référence) dont elle à la charge de remplir le contenu.

Ce contenu sera plus tard affiché là où la zone est appelée.

La fonction _createContent doit retourner un booléen qui indique si la génération du contenu s'est bien passée. Ce paramètre est important car les données ne seront mises en cache que si la fonction _createContent retourne "true".

Zone & Paramètres

Il est possible de passer des paramètres à une zone. Ces paramètres seront récupérés dans la zone via la méthode $this->getParam ().

Exemple

```
class ZoneExempleParam extends CopixZone {
   function _createContent (&$toReturn){
      $toReturn = 'La valeur du paramètre PARAM est '.$this->getParam ('PARAM');
      return true;
   }
}
```

Note : Il est possible de donner une valeur par défaut aux paramètres si jamais l'appelant ne les spécifie pas, par exemple :

```
class ZoneExempleParam extends CopixZone {
   function _createContent (&$toReturn){
      $toReturn = 'La valeur du paramètre PARAM est '.$this->getParam ('PARAM', 'Si
            ✂ paramètre non donné');
      return true;
   }
}
```

Appel des zones

depuis un code PHP

```
$contenuObtenu = CopixZone::process ('mon_module|Exemple');//appel de la zone
            ✂ ZoneExemple dans mon_module/zones/exemple.zone.php

//avec des paramètres
$contenuObtenu = CopixZone::process ('module|NOM_ZONE', array ('PARAM'=>'Valeur du
            ✂ paramètre'));//appel de la zone ZoneExempleParam dans
            ✂ mon_module/zones/exemple.zone.php avec un paramètre PARAM
```

depuis un template Smarty

```
{copixzone process=exemple}

{copixzone process=exempleparam PARAM="Valeur du paramètre PARAM"}
```

Exemple concret, ZoneEphemeride

Par exemple, dans un module, vous avez une partie de l'affichage qui affiche la date et le saint du jour. Cette partie est typiquement une zone. Elle ne sert qu'à l'affichage et n'effectue pas elle même de traitements lourds.

Le code de la zone

```
class ZoneEphemeride extends CopixZone {
   function _createContent (& $toReturn) {
      $tpl = new CopixTpl ();

      $date = getdate();
      $jours = array("dimanche", "lundi", "mardi", "mercredi", "jeudi", "vendredi",
"samedi");
      $mois = array("janvier", "février", "mars", "avril", "mai", "juin", "juillet",
"août", "septembre", "octobre", "novembre", "décembre");

      $tpl->assign('DATE', $jours[$date["wday"]]." ".$date["mday"]."
".$mois[$date["mon"]-1]." ".$date["year"]);
      $tpl->assign('SAINT', getTodaysSaint());

      $toReturn = $tpl->fetch('navigation.tpl');
      return true;
   }

   private function getTodaysSaint()
   {
      // on regarde le saint du jour et on le retourne
      return TodaysSaint;
   }
}
```

Le code du template navigation.tpl

```
<p>{$DATE}</p>
<p>{$SAINT}</p>
```

Voir aussi

- CopixHTMLHeader Qui vous permettra de demander à Copix d'injecter des contenus javascript ou CSS nécessaires à la zone dans le template principal.

CopixFile

Introduction

CopixFile est une classe qui permet la lecture et l'écriture de contenus.

Son principal rôle est bien sûr d'écrire des fichiers, sans avoir à se soucier des concurrences d'accès.

Cette classe ne gère pas les problématiques de modification (c'est un remplacement de contenu qui est effectué).

read ($filePath)

Lit le contenu du fichier $filePath ($filePath est le chemin du fichier sur le serveur). read () retourne "false" en cas d'échec de lecture.

Exemple

```
$objectReader = new CopixFile ();
if (($content = $objectReader->read ($filePath)) !== false){
   $tpl->assign ('texte', $content);
}else{
   $tpl->assign ('texte', CopixI18N::get ('error.cannotReadFile'));
}
```

write ($filePath, $content)

Cette méthode va créer ou remplacer le fichier $filePath ($filePath est le chemin du fichier sur le serveur) et y placer le contenu $content (de type chaîne de caractères). Cette méthode est sûre, il n'existe pas de risque de concurrence d'accès.

Note : Sous un système de type Windows, la concurrence d'accès n'est pas assurée du fait que le système n'est pas capable de prendre en charge le "remplacement d'un fichier" (élément technique mis en oeuvre dans la méthode write). Toutefois, le risque est particulièrement minime de voir échouer un remplacement. Sous unix/linux, cette manipulation est parfaitement sûre.

Cette méthode est capable de créer le chemin menant à $filePath si ce dernier n'existe pas.

Exemple

```
$objectWriter = new CopixFile ();
return $objectWriter->write ($fileName, 'Contenu du fichier');
```

createDir ($chemin)

La méthode createDir permet de créer un répertoire dont le chemin est $chemin.

Si les sous chemins n'existent pas, ils seront crées.

La méthode retourne vrai ou faux selon la réussite de la création de l'arborescence.

search ($pPattern, $pPath, $pRecursiveSearch = true)

La méthode search permet de rechercher un ou plusieurs fichiers qui respectent le pattern de recherche $pPattern dans une arborescence qui commence dans le chemin $pPath.

Le dernier paramètre indique si vous souhaitez étendre votre recherche dans les sous répertoires (par défaut vrai).

La méthode vous retourne un tableau de chaines de caractères qui représentent l'ensemble des fichiers trouvés qui correspondent à la recherche.

trailingSlash ($pPath)

Cette méthode retourne $pPath en s'assurant que la chaine se termine par un slash.

removeDir ($pDirectory, $pStopOnFailure = false)

Cette méthode supprime le répertoire $pDirectory et ses sous répertoires. Elle retourne vrai en cas de succès et un tableau de noms de fichiers qu'il à été impossible de supprimer en cas d'échec.

Si vous spécifiez $pStopOnFailure, alors la supression sera interrompue au premier échec de suppression.

Si $pDirectory se termine par un slash "/", seul le contenu du répertoire est supprimé. Si $pDirectory ne se termine par un slash "/", le répertoire lui même est également supprimé.

removeFileFromPath ($pDirectory, $pStopOnFailure = false)

Cette méthode supprime l'ensemble des fichiers (et non les répertoires) qui sont contenus dans le répertoire $pDirectory et ses sous répertoires.

Elle retourne vrai en cas de succès et un tableau de noms de fichiers qu'il à été impossible de supprimer en cas d'échec.

Si vous spécifiez $pStopOnFailure, alors la supression sera interrompue au premier échec de suppression.

extractFileName ($pPath)

Extrait le nom du fichier du chemin passé en paramètre.

extractFilePath ($pPath)

Extrait le chemin (sans le nom du fichier) du chemin passé en paramètre.

extractFileExt ($pPath)

Extrait l'extension du fichier du chemin passé en paramètre.

getIcon ($pFilePath)

Récupère le chemin de l'icône (pour être passé à la fonction **CopixUrl::getRessource ( )**) correspondant au type de fichier. Cette correspondance est basée sur l'extension du fichier et n'est pas exhaustive.

Cette fonction est disponible depuis Copix 3.0.3+

CopixTimer

Introduction

CopixTimer est une classe utilitaire capable de mesurer le temps écoulé entre deux appels à "start" et "stop". Plusieurs compteurs peuvent être menés simultanéements.

Utilisation

```
//instanciation du timer
$timer = new CopixTimer ();

$timer->start ();
//exécution de code
echo $timer->stop (), "secondes ont étés nécessaires pour éxécuter le code.";

//-------------------
//Compteurs multiples
$timer->start ();

$timer->start ();//démarre un souscompteur
//exécution de code 1
echo $timer->stop (), "ont étés nécessaires pour code 1 <br />";

$timer->start ();
//exécution de code 2
echo $timer->stop (), "ont étés nécessaires pour code 2 <br />";

echo $timer->stop (), 'ont étés nécessaires au total pour code 1 et 2 <br />';
```

CopixDateTime

Manipulation de dates et d'heures

CopixDateTime permet de manipuler des dates, des heures et des timestamps au sein de votre application Copix. Grâce à cette classe, vous pouvez par exemple convertir un timestamp en date en respectant le formatage induit par une langue donnée.

Dates

Date => YYYYMMDD

Entrée : vous disposez d'une date formatée contenant 3 champs (JJ, MM, YYYY) rangés dans un certain ordre et séparés par un caractère donné (/ par défaut).

Sortie : vous obtenez une date respectant le formatage YYYYMMDD

```
// CopixDateTime::dateToYYYYMMDD ($date, $separator);

$maDate = CopixDateTime::dateToYYYYMMDD ('31/12/2006', '/'); //retourne 20061231

$maDate = CopixDateTime::dateToYYYYMMDD ('', '/'); //retourne null
$maDate = CopixDateTime::dateToYYYYMMDD (null, '/'); //retourne null

$maDate = CopixDateTime::dateToYYYYMMDD ('31/12', '/'); //retourne false
$maDate = CopixDateTime::dateToYYYYMMDD ('310/120/2006', '/'); //retourne false
$maDate = CopixDateTime::dateToYYYYMMDD ('3/1/2', '/'); //retourne false
```

Date => Timestamp

Entrée : vous disposez d'une date formatée contenant 3 champs (JJ, MM, YYYY) rangés dans un certain ordre et séparés par un caractère donné (/ par défaut).

Sortie : vous obtenez une date sous forme d'un timestamp UNIX (**http://fr.php.net/date**)

```
// CopixDateTime::dateTotimestamp ($date, $separator);

$maDate = CopixDateTime::dateTotimestamp ('31/12/2006', '/'); //retourne 1167516000

$maDate = CopixDateTime::dateTotimestamp('', '/'); //retourne false
$maDate = CopixDateTime::dateTotimestamp (null, '/'); //retourne false
$maDate = CopixDateTime::dateTotimestamp ('31/12', '/'); //retourne false
$maDate = CopixDateTime::dateTotimestamp ('310/120/2006', '/'); //retourne false
$maDate = CopixDateTime::dateTotimestamp ('3/1/2', '/'); //retourne false
```

YYYYMMDD => Date

Entrée : vous disposez d'une date respectant le formatage YYYYMMDD

Sortie : vous obtenez une date formatée contenant 3 champs (JJ, MM, YYYY) rangés dans l'ordre établi par la langue courante et séparés par un caractère donné (/ par défaut).

```
// CopixDateTime::yyyymmddToDate ($yyyymmdd, $separator);

$maDate = CopixDateTime::yyyymmddToDate ('20061231', '/'); //en France, retourne
          ✂ 31/12/2006

$maDate = CopixDateTime::yyyymmddToDate ('', '/'); //retourne null
$maDate = CopixDateTime::yyyymmddToDate (null, '/'); //retourne null

$maDate = CopixDateTime::yyyymmddToDate ('2006120310', '/'); //retourne false
$maDate = CopixDateTime::yyyymmddToDate ('213', '/'); //retourne false
```

YYYYMMDD => Texte

Entrée : vous disposez d'une date respectant le formatage YYYYMMDD

Sortie : vous obtenez une date sous forme d'un texte lisible en fonction de la langue courante

```
// CopixDateTime::yyyymmddToText ($yyyymmdd);

$maDate = CopixDateTime::yyyymmddToText ('20061231');
//en France, retourne Dimanche 31 décembre 2006

$maDate = CopixDateTime::yyyymmddToText (''); //retourne null
$maDate = CopixDateTime::yyyymmddToText (null); //retourne null

$maDate = CopixDateTime::yyyymmddToText ('2006120310'); //retourne false
$maDate = CopixDateTime::yyyymmddToText ('213'); //retourne false
```

YYYYMMDD => Timestamp

Entrée : vous disposez d'une date respectant le formatage YYYYMMDD

Sortie : vous obtenez une date sous forme d'un timestamp UNIX (**http://fr.php.net/mktime**)

```
// CopixDateTime::yyyymmddToTimestamp ($yyyymmdd);

$maDate = CopixDateTime::yyyymmddToTimestamp ('20061231'); //retourne 1167516000

$maDate = CopixDateTime::yyyymmddToTimestamp (''); //retourne false
$maDate = CopixDateTime::yyyymmddToTimestamp (null); //retourne false
$maDate = CopixDateTime::yyyymmddToTimestamp ('213'); //retourne false
```

Timestamp => YYYYMMDD

Entrée : vous disposez d'une date sous forme d'un timestamp UNIX (**http://fr.php.net/strftime**)

Sortie : vous obtenez une date respectant le formatage YYYYMMDD

```
// CopixDateTime::timestampToyyyymmdd($timestamp);

$maDate = CopixDateTime::timestampToyyyymmdd('1167563702'); //retourne 20061231
```

Timestamp => Date

Entrée : vous disposez d'une date sous forme d'un timestamp UNIX (**http://fr.php.net/date**)

Sortie : vous obtenez une date formatée contenant 3 champs (JJ, MM, YYYY) rangés dans l'ordre établi par la langue courante et séparés par un caractère donné (/ par défaut).

```
// CopixDateTime::timestampToDate ($timestamp, $separator);

$maDate = CopixDateTime::timestampToDate ('1167563702', '/');
//en France, retourne 31/12/2006
```

Heures

Heure => HHMMSS

Entrée : vous disposez d'une heure formatée contenant 3 champs (HH, MM, SS) rangés dans l'ordre établi par la langue courante et séparés par un caractère donné (: par défaut)

Sortie : vous obtenez une heure respectant le formatage HHMMSS

```
// CopixDateTime::timeToHHMMSS ($time, $separator);

$monHeure = CopixDateTime::timeToHHMMSS ('13:15:02', ':'); //retourne 131502

$monHeure = CopixDateTime::timeToHHMMSS ('', ':'); //retourne null
$monHeure = CopixDateTime::timeToHHMMSS (null, ':'); //retourne null

$monHeure = CopixDateTime::timeToHHMMSS ('131502', ':'); //retourne false
$monHeure = CopixDateTime::timeToHHMMSS ('25:15:02', ':'); //retourne false
$monHeure = CopixDateTime::timeToHHMMSS ('13:75:02', ':'); //retourne false
$monHeure = CopixDateTime::timeToHHMMSS ('13:15:72', ':'); //retourne false
$monHeure = CopixDateTime::timeToHHMMSS ('a:b:c', ':'); //retourne false
```

HHMMSS => Heure

Entrée : vous disposez d'une heure respectant le formatage HHMMSS

Sortie : vous obtenez une heure formatée contenant 3 champs (HH, MM, SS) rangés dans l'ordre établi par la langue courante et séparés par un caractère donné (: par défaut).

```
// CopxDateTime::hhmmssToTime ($hhmmss, $separator);

$monHeure = CopxDateTime::hhmmssToTime ('131502', ':'); //en France, retourne 13:15:02

$monHeure = CopxDateTime::hhmmssToTime ('', ':'); //retourne null
$monHeure = CopxDateTime::hhmmssToTime (null, ':'); //retourne null

$monHeure = CopxDateTime::hhmmssToTime ('130150020', ':'); //retourne false
$monHeure = CopxDateTime::hhmmssToTime ('110', ':'); //retourne false
```

Dates et Heures

YYYYMMDDHHIISS => DateHeure

Entrée : vous disposez d'une chaîne date/heure respectant le formatage YYYYMMDDHHIISS

Sortie : vous obtenez une chaîne date/heure formatée contenant 6 champs (DD, MM, YYYY, HH, II, SS) rangés dans l'ordre établi par la langue courante et séparés par un caractère donné (/ par défaut).

```
// CopixDateTime::yyyymmddhhiissToDateTime($pParam, $separator);

$monDateHeure = CopixDateTime::yyyymmddhhiissToDateTime('20061231131502', '/');
//en France, retourne 31/12/2006 13:15:02

$monDateHeure = CopixDateTime::yyyymmddhhiissToDateTime('', '/'); //retourne null
$monDateHeure = CopixDateTime::yyyymmddhhiissToDateTime(null, '/'); //retourne null

$monDateHeure = CopixDateTime::yyyymmddhhiissToDateTime('20061231131502000', '/');
//retourne false
$monDateHeure = CopixDateTime::yyyymmddhhiissToDateTime('2006123113', '/'); //retourne
            ✂ false
$monDateHeure = CopixDateTime::yyyymmddhhiissToDateTime('20061231251502', '/');
            ✂ //retourne false
$monDateHeure = CopixDateTime::yyyymmddhhiissToDateTime('20061231137502', '/');
            ✂ //retourne false
$monDateHeure = CopixDateTime::yyyymmddhhiissToDateTime('20061231131572', '/');
            ✂ //retourne false
```

YYYYMMDDHHIISS => Texte

Entrée : vous disposez d'une chaîne date/heure respectant le formatage YYYYMMDDHHIISS

Sortie : vous obtenez une chaîne date/heure formatée en texte pour la partie date (voir YYYMMDD=>Text) et l'heure à coté.

```
// CopixDateTime::yyyymmddToText ($yyyymmdd);

$maDate = CopixDateTime::yyyymmddhhiissToText ('20061231143020');
//en France, retourne Dimanche 31 décembre 2006 14:30:20

$maDate = CopixDateTime::yyyymmddhhiissToText (''); //retourne null
$maDate = CopixDateTime::yyyymmddhhiissToText (null); //retourne null

$maDate = CopixDateTime::yyyymmddhhiissToText ('2006120310'); //retourne false
$maDate = CopixDateTime::yyyymmddhhiissToText ('213'); //retourne false
```

YYYYMMDDHHIISS => Timestamp

Entrée : vous disposez d'une chaîne date/heure respectant le formatage YYYYMMDDHHIISS

Sortie : vous obtenez une chaîne date/heure sous forme d'un timestamp UNIX (**http://fr.php.net/date**)

```
// CopixDateTime::yyyymmddhhiissToTimeStamp($pParam);

$monDateHeure = CopixDateTime::yyyymmddhhiissToTimeStamp('20061231131502');
            ✂ //retourne 1167563702

$monDateHeure = CopixDateTime::yyyymmddhhiissToTimeStamp(''); //retourne null
$monDateHeure = CopixDateTime::yyyymmddhhiissToTimeStamp(null); //retourne null

$monDateHeure = CopixDateTime::yyyymmddhhiissToTimeStamp('20061231251502000');
            ✂ //retourne false
$monDateHeure = CopixDateTime::yyyymmddhhiissToTimeStamp('200612312'); //retourne false
$monDateHeure = CopixDateTime::yyyymmddhhiissToTimeStamp('20061231251502');
            ✂ //retourne false
$monDateHeure = CopixDateTime::yyyymmddhhiissToTimeStamp('20061231136502');
            ✂ //retourne false
$monDateHeure = CopixDateTime::yyyymmddhhiissToTimeStamp('20061231131562');
            ✂ //retourne false
```

Timestamp => YYYYMMDDHHIISS

Entrée : vous disposez d'une chaîne date/heure sous forme d'un timestamp UNIX (**http://fr.php.net/date**)

Sortie : vous obtenez une chaîne date/heure respectant le formatage YYYYMMDDHHIISS

```
// CopixDateTime::timeStampToyyyymmddhhiiss($timestamp);

$monDateHeure = CopixDateTime::timeStampToyyyymmddhhiiss('1167563702');
//retourne 20061231131502
```

DateHeure => YYYYMMDDHHIISS

Entrée : vous disposez d'une chaîne date/heure formatée contenant 6 champs (DD, MM, YYYY, HH, II, SS) rangés dans un ordre donné et séparés par un caractère donné (/ par défaut).

Sortie : vous obtenez une chaîne date/heure respectant le formatage YYYYMMDDHHIISS.

```
// CopixDateTime::DateTimeToyyyymmddhhiiss($DateTime,$separator);

$monDateHeure = CopixDateTime::DateTimeToyyyymmddhhiiss('31/12/2006 13:15:02', '/');
//retourne 20061231131502
```

DateHeure ISO-8601 => DateHeure Local

Entrée : vous disposez d'une chaîne date/heure sous forme d'un chaîne au format ISO-8601 (YYYY-MM-DD hh:ii:ss ou YYYY-MM-DDThh:ii:ssZ)

Sortie : vous obtenez une chaîne date/heure formatée contenant 6 champs (DD, MM, YYYY, HH, II, SS) rangés dans l'ordre établi par la langue courante et séparés par un caractère donné (/ par défaut).

```
// CopixDateTime::ISODateTimeToDateTime($pParam, $separator);

$monDateHeure = CopixDateTime::ISODateTimeToDateTime('2006-12-31 13:15:02', '/');
//en France, retourne 31/12/2006 13:15:02

$monDateHeure = CopixDateTime::ISODateTimeToDateTime('', '/'); //retourne null
$monDateHeure = CopixDateTime::ISODateTimeToDateTime(null, '/'); //retourne null

$monDateHeure = CopixDateTime::ISODateTimeToDateTime('20-1-3 13:15:02', '/');
            ✂ //retourne false
$monDateHeure = CopixDateTime::ISODateTimeToDateTime('2006-120-310 13:15:02', '/');
            ✂ //retourne false
$monDateHeure = CopixDateTime::ISODateTimeToDateTime('2006-12-31 25:15:02', '/');
            ✂ //retourne false
$monDateHeure = CopixDateTime::ISODateTimeToDateTime('2006-12-31 13:75:02', '/');
            ✂ //retourne false
$monDateHeure = CopixDateTime::ISODateTimeToDateTime('2006-12-31 13:15:72', '/');
            ✂ //retourne false
```

CopixUrl

Présentation

CopixUrl est une classe qui vous permet de générer des urls d'appels au format Copix. Les URL générées prendront en compte :

- Le mode d'appel (URL Significative ou non);
- L'url de base de vos scripts (noms de domaine, nom du fichier, mode sécurisé ou non, ...);
- Les gestionnaires d'url (**URLHandler**) que vous aurez développé pour personnaliser les appels.

En une ligne, selon la configuration utilisée, cette classe peut générer des chaines de la forme

http(s)://(www.)domaine.ext/chemin/monsite/articles/Un-Article-Interressant.htm ou

http(s)://(www.)domaine.ext/chemin/monsite/index.php?module=articles&action=get&id=Un-Article-Interressant

Méthodes de CopixUrl

getRequestedScript ()

Retourne le script demandé, sans les informations infopath ou paramètres de la requête

```
//Si appelé depuis http://mysite.com/subdir/index.php/mypath/myaction?myparams=myvalues
echo CopixUrl::getRequestedScript ();
//affiche "/subdir/index.php"
```

getRequestedScriptPath ()

Retourne le chemin du script demandé.

```
//Si appelé avec http://mysite.com/subdir/index.php/mypath/myaction?myparams=myvalues
echo CopixUrl::getRequestedScriptPath ();
//affiche "/subdir/"
```

getRequestedScriptName ()

Retourne le nom du script demandé

```
//Si appelé avec http://mysite.com/subdir/index.php/mypath/myaction?myparams=myvalues
echo CopixUrl::getRequestedScriptName ();
//affiche "index.php"
```

getRequestedDomain ()

Récupère le nom de domaine du script demandé.

```
//si appelé avec http://mysite.com/subdir/index.php/mypath/myaction?myparams=myvalues
echo CopixUrl::getRequestedDomain ();
//affiche mysite.com
```

getRequestedBasePath ()

Récupère le début de l'url

```
//Si appelé avec http://mysite.com/subdir/index.php/mypath/myaction?myparams=myvalues
echo CopixUrl::getRequestedScriptPath ();
//affiche "mysite.com/subdir/"
```

getRequestedBaseUrl ()

Récupère le chemin allant jusqu'au script exclus en incluant le protocole utilisé.

```
//Si appelé avec http://mysite.com/subdir/index.php/mypath/myaction?myparams=myvalues
echo CopixUrl::getRequestedScriptPath ();
//affiche "http://mysite.com/subdir/"
```

getRequestedProtocol ()

Récupération du protocole (http/https pour le moment)

```
//Si appelé avec http://www.copix.org
echo CopixUrl::getRequestedProtocol ();
//affiche "http://"
```

getRequestedPathInfo ()

Récupère le pathinfo

```
//Si appelé via http://localhost/copix_3/test.php/stuff/stuff2/stuff3?test=simpletest|
echo CopixUrl::getRequestedPathInfo ();
//affiche "/stuff/stuff2/stuff3"
```

getCurrentUrl ($pForXML = false)

Récupère l'url demandée

$pForXML (boolean) : si l'on génère en XML ou non (& au lieu de &)

getRequestedUrl ($pForXML = false)

Récupère l'url demandée

$pForXML (boolean) : si l'on génère en XML ou non (& au lieu de &)

valueToUrl ($pName, $pValue, $pStartWithAmp = false, $pForXml = false)

Convertit une valeur en sa représentation dans une URL.

Si $pName est null, alors $pValue doit être un tableau associatif avec $pName=>$pValue

$pName (string) : le nom de la variable à convertir

$pValue (string/int/array) : La valeur de la variable $pName

(boolean) : oolean $start

$pForXml (boolean) : Si l'on doit générer l'url pour du XML ou non

appendToUrl ($pUrl, $pParams = array (), $pForXML = false)

Ajoute des paramètres à l'url donnée

Si jamais l'url donnée contient déja les paramètres que l'on a demandé de rajouter, les paramètres donnés remplaçeront les exisant

$pUrl (string) : l'url à qui on veut ajouter des paramètres

$pParams (array) : les paramètres à ajouter à l'url

$pForXML (boolean) : si l'on ajoute à une URL destinée XML ou non

```
echo CopixUrl::appendToUrl ('http://www.copix.org/index.php?module=test', array
            ✂ ('group'=>'value'));
//affiche http://www.copix.org/index.php?module=test&group=value
echo CopixUrl::appendToUrl ('http://www.copix.org/index.php?group=test', array
            ✂ ('group'=>'value'));
//affiche http://www.copix.org/index.php?group=value
```

parse ($pUrl, $pFromString = false, $pFromXML = false)

Analyse de la requête donnée et retourne la liste des paramètres envoyés au script

$pUrl (string) : l'url à analyser

$pFromString (boolean) : Indique si la requête est arrivée uniquement sous la forme de chaine (on utilisera pas les variables d'environnement pour l'analyser (_GET, _POST)

$pFromXml (boolean) : Indique si la requête donnée est au format XML (& au lieu de &)

get ($pDest = null, $pParams = array (), $pForXML = false)

Gets the url string from parameters

$pDest (string) : the module|dest|action string

$pParams (array) : an associative array with the parameters

$pForXML (boolean) : the string has to be for html

escapeSpecialChars ($pString)

Supprime les champs spéciaux (irréversible, utilisable a de simples fin de présentation)

$pString (string) : la chaine dont on veut supprimer les caractères spéciaux

removeParams ($pUrl, $pParams, $pForXML = false)

Supprime de l'url $pUrl les paramètres $pParams

Le paramètre $pForXml est pris en compte à la fois pour l'interprétation de la requête d'origine et à la fois pour la génération. Il n'est pas possible d'avoir un fonctionnement dissocié

$pUrl (string) : url a nettoyer

$pParams (array) : les pièces à supprimer de l'url (array ('param', 'param2', 'param3');

$pForXML (boolean) : si l'on génère une sortie html ou non (& au lieu de &)

extractParams ($pUrl, $pFromXML)

Extrait les paramètres d'une requête $pUrl

$pUrl (string) : l'url à analyser

$pFromXML (boolean) : si l'on analyse une chaine provenant de HTML ou non

getResource ($pResourcePath)

Récupère un chemin de ressource (situé dans www)

Ira chercher dans l'ordre de priorité dans ./nom_theme/lang_COUNTRY/$path ./nom_theme/lang/$path ./nom_theme/$path ./default/lang_COUNTRY/$path ./default/lang/$path ./default/$path ./$path

$resourcePath (string) : le chemin du fichier que l'on souhaites récupérer

```
//on souhaites récupérer la feuille de style
$path = CopixURL::getRessource ('styles/copix.css');
//$path ==
http://www.domaine.fr/chemin/vers/le/script/themes/nom_du_theme/styles/copix.css si le
fichier existe
```

getResourcePath ($pResourcePath)

Récupère un chemin de ressource (situé dans www)

Ira chercher dans l'ordre de priorité dans ./nom_theme/lang_COUNTRY/$path ./nom_theme/lang/$path ./nom_theme/$path ./default/lang_COUNTRY/$path ./default/lang/$path ./default/$path ./$path

$resourcePath (string) : le chemin du fichier que l'on souhaites récupérer

```
//on souhaites récupérer la feuille de style
$path = CopixURL::getRessourcePath ('styles/copix.css');
//$path == /var/www/themes/nom_du_theme/styles/copix.css si le fichier existe
```

Voir aussi

- La balise {copixurl}
- CopixRequest Pour récupérer les paramètres envoyés au script
- Comprendre la forme des URL
- Trigramme module group action pour décrire les URL

CopixCSV

Présentation

CopixCSV est une classe vous permettant de manipuler les fichiers de type CSV.

Créer un fichier CSV

Pour commencer et créer un fichier CSV, il convient d'instancier un objet de type CopixCSV et prenant pour paramètre le nom du fichier. Si celui ci n'existe pas, il sera créé. Il existe deux paramètres optionnels permettant de définir le séparateur de champ et le délimiteur de texte.

Par défaut le séparateur de champ sera , et le délimiteur de texte '

```
// Déclaration d'un objet csv
$objectCsv = new CopixCSV ('fichier.csv');
// Déclaration d'un objet csv dont le séparateur de champ est ;
$objectCsv = new CopixCSV ('fichier.csv', ';');

// Déclaration d'un objet csv dont le séparateur de champ est ; et le délimiteur "
$objectCsv = new CopixCSV ('fichier.csv', ';', '"');


Pour ajouter une ligne la classe met à disposition une méthode addLine prenant en
paramètre un tableau contenant les éléments à ajouter dans le fichier.


$arElems = array("champ1", "champ2", "champ3");
$objectCsv->addLine($arElems);
```

Paramètres optionnels

Il est possible, au moment de l'instanciation de votre classe, de définir le délimiteur et le séparateur de texte de votre fichier CSV :

```
// Création d'un fichier CSV ayant pour séparateur un | et délimiteur de texte "
$objectCsv = new CopixCSV ('fichier.csv', '|', '"');
```

Parcourir un fichier CSV

Il existe un itérateur pour parcourir vos fichiers CSV.

- CopixCsvIterator : pour parcourir un fichier CSV de la première à la dernière ligne.

La classe CopixCSV fournit deux méthodes pour récupérer ces itérateurs

- getIterator()

exemple avec getIterator ()

Parcourir le code en sens normal :

```
$objectCsv = new CopixCSV ('fichier.csv', '|', '"');

// $csvIterator contient l'itérateur sur le fichier CSV déclaré par $objectCsv
$csvIterator = $objectCsv-> getIterator();

// $line est un tableau indexé numériquement  contenant les champs de chaque ligne du
fichier CSV.
foreach ($csvIterator as $line) {
    // On affiche le premier champ
    echo $line[0];
}
```

Utilisation de la première ligne comme en-tête (Copix 3.0.2 et suivante)

Depuis la version 3.0.2 il est possible d'indiquer à l'iterator que les index des tableaux renvoyés sont indexés par les champs de la première ligne du fichier csv. Dans ce cas la première ligne ne sera pas renvoyés. Pour celà il faut passé une valeur vrai à la méthode getIterator. Dans Copix 3.0.3 des constantes remplaceront le booléen.

fichier.csv

```
champ1;champ2;champ3
info1;info2;info3
```

En PHP (Copix 3.0.2):

```
$objectCsv = new CopixCSV ('fichier.csv', ';');

// $csvIterator contient l'itérateur sur le fichier CSV déclaré par $objectCsv
$csvIterator = $objectCsv-> getIterator(true);

// $line est un tableau indexé numériquement  contenant les champs de chaque ligne du
fichier CSV.
foreach ($csvIterator as $line) {
    // On affiche le premier champ
    echo $line['champ1'];
}
```

Utilisation dans Copix

La classe CopixCSV est utilisée dans la stratégie de log fichier : */copix/utils/copix/log/CopixLogFileStrategy.class.php*

CopixCookie

Qu'est-ce qu'un cookie ?

Il est possible de manipuler des cookies avec Copix. Avant tout, qu'est-ce qu'un cookie ?

Un cookie est une valeur que vous pourrez stocker sur le poste du client, et plus exactement sur le navigateur. Cette valeur à un temps de vie, et reste lisible même après fermeture du navigateur client, à l'inverse d'une variable de session.

Cela peut-être très pratique pour - par exemple - garder en mémoire le nom que l'utilisateur à saisi dans un formulaire, ou même pour l'authentifier directement !

Avec la classe CopixCookie, cela reste très facile:

Enregistrer et lire

Pour enregistrer une valeur, il suffit d'utiliser la méthode "set" avec 3 paramètres:

- le premier paramètre est le nom de la variable
- le second est la valeur à enregistrer (peut-être n'importe quel type, même un objet)
- le troisième paramètre correspond à un namespace (en réalité, le namespace sera appelé "default")
- le temps de vie du cookie en secondes (200 ans si non donnée)

Voici comment enregistrer un cookie valable 1 jour:

```
CopixCookie::set('test',"La valeur est cette chaine", null ,60 * 60 * 24)) ; // 24h
```

Désormais, pour lire la valeur:

```
$val = CopixCookie::get('test');
```

$val contiendra alors "La valeur de cette chaine".

Namespace

Vous pouvez indiquer un namespace au cookie. Cela est intéressant dans le cas où vous ne voulez pas risquer d'écraser une valeur existante.

Par exemple:

```
//deux variables de cookie nommées "foo"
CopixCookie::set("foo","valeur 1");
CopixCookie::set("foo","valeur 2","bar");

//les deux variables se récupèrent selon leur namespace
$val1 = CopixCookie::get('foo'); //$val1 = "valeur 1" car namespace par défaut
$val2 = CopixCookie::get('foo','bar'); //$val2 = "valeur 2" car namespace 'bar' utilisé
```

L'implémentation est donc très proche de **CopixSession**.

Suppression

La méthode "delete" prend en paramètre le nom de la variable et le namespace. Si le namespace n'est pas donné, ce sera le namespace par défaut qui sera utilisé.

```
CopixCookie:delete('foo'); //supprime la variable 'foo' du namespace par défaut
CopixCookie:delete('foo','bar'); //supprime la variable 'foo' du namespace 'bar'
```

Anti_Spam

Objectifs

Ce module a pour utilité de se protéger contre les robots qui remplissent automatiquement les formulaires.

Protéger un formulaire avec un captcha (image de charactère)

En tout premier lieux installez le module *antispam*.

Pour générer une image il faut appeller le tag correspondant :

```
$id = uniqid();
_etag('imageprotect', array ('id'=>$id));
```

ou en smarty

```
{imageprotect id="$id"}
```

Puis pendant la validation du formulaire il suffit de tester si la valeur saisie est bien celle affiché dans le captcha, **$id** étant l'identifiant utilisé à la création de l'image.

```
if (!ImageProtect::getCode($id, $code)) {
    throw new CopixException ('Mauvaise valeur saisie'));
}
```

Après la première utilisation de la méthode **ImageProtect::getCode()**, *elle renvoie automatiquement* **false** *si elle est réutilisée avec le même identifiant (par soucis de sécurité).*

Classe Copix PPO

CopixPPO

Présentation

CopixPPO est une classe "vide" dont l'objectif est de transporter des données, principalement utilisée dans les **actions** de type **PPO**.

Demander la récupération d'une propriété inexistante d'un objet de type PPO ne génère pas de notice.

Exemples

```
$ppo = new CopixPPO ();
$ppo->data = 'contenu';
$ppo->data_autre = 'contenu autre';
//...

echo $ppo->existe_pas; //ne génère pas de notice
```

Il est possible de passer un tableau associatif au constructeur pour initialiser l'objet.

exemple

```
$ppo = new CopixPPO (array ('TITLE_PAGE'=>'Titre de ma page'));
$ppo->data = 'contenu';
$ppo->data_autre = 'contenu autre';
//...
```

CopixActionReturn_PPO

Présentation

Le code retour PPO est utilisé pour demander à Copix d'afficher des données préparées dans un **CopixPPO** dans un template.

C'est le code classique qui est retourné par vos **actions** lorsque vous souhaitez effectuer un affichage de type texte (html, css, xml, ...)

Exemple classique

```
function processUneActionPpo (){
   $ppo = new CopixPPO ();
   $ppo->TITLE_PAGE  = 'Titre de ma page';
   $ppo->infoEnPlus  = 'Une information complémentaire de mon choix';
   $ppo->autresInfos = "Encore une variable, ce n'est pas limité";
   return new CopixActionReturn (CopixActionReturn::PPO, $ppo,
'template_pour_ppo.tpl');
   //ou simplement avec le raccourcis
   return _arPpo ($ppo, 'template_pour_ppo.tpl');
}
```

avec le template suivant

```
<p>{$ppo->infoEnPlus}</p>
<p>{$ppo->autresInfos}</p>
```

Votre template template_pour_ppo.tpl sera inclus dans le **template principal**, dans la variable {$MAIN}.

Note : PPO signifie "Plain PHP Object".

Options supplémentaires

Le code retour PPO accepte plusieurs options. Ces options sont transmises dans un tableau.

Spécifier le template à utiliser avec template

Vous devez spécifier le template à utiliser avec votre PPO. Cette option est l'option transmise par défaut.

```
function processUneActionPpo (){
   $ppo = new CopixPPO ();
   $ppo->TITLE_PAGE  = 'Titre de ma page';
   return new CopixActionReturn (CopixActionReturn::PPO, $ppo,
            'template_pour_ppo.tpl');
   //ou simplement avec le raccourcis
   return _arPpo ($ppo, 'template_pour_ppo.tpl');
   //ou sous la forme d'un tableau d'options
   return new CopixActionReturn (CopixActionReturn::PPO, $ppo, array
            ('template'=>'template_pour_ppo.tpl'));
   //Sous la forme d'un tableau avec le raccourcis
   return _arPpo ($ppo, array ('template'=>'template_pour_ppo.tpl'));
}
```

Spécifier le type MIME avec content-type

Vous pouvez spécifier le type de contenu que vous affichez. Pour ce faire, il vous faut donner l'option content-type, par défaut positionnée à text/html.

Exemple

```
function processUneActionQuiGenereDuXML (){
    $ppo = new CopixPpo ();
    $ppo->arData = _dao ('nomTable')->findAll ();
    return _arPPO ($ppo, array ('template'=>'template.xhtml.tpl', 'content-type'
           ✂ =>'text/xml', 'mainTemplate'=>null));
}
```

Note : Ici, on utilise également l'option mainTemplate à null pour indiquer que nous ne souhaitons pas utiliser de template principal et directement afficher notre template "template.xhtml.tpl" avec les données.

Spécifier le charset utilisé avec charset

Vous pouvez spécifier le charset que vous avez utilisé pour générer votre page. Par défaut, si vous ne spécifiez rien, Copix utilisera **CopixI18N**::getCharset () pour connaitre le charset courant.

exemple

```
function processUneActionUTF8 (){
    $ppo = new CopixPpo ();
    $ppo->arData = _dao ('nomTable')->findAll ();
    return _arPPO ($ppo, array ('template'=>'template.xhtml.tpl', 'charset'=>
           ✂ 'UTF-8'));
}
```

Spécifier le template principal à utiliser avec mainTemplate

Vous pouvez utiliser un autre **template principal** que celui configuré par défaut dans l'application pour une action donnée. Pour cela, il suffit d'utiliser l'option mainTemplate.

```
function processUneActionPopUP (){
    $ppo = new CopixPpo ();
    $ppo->arData = _dao ('nomTable')->findAll ();
    return _arPPO ($ppo, array ('template'=>'template.xhtml.tpl',
           ✂ 'mainTemplate'=>'principal_pour_popup.tpl'));
}
```

Ne pas utiliser de template principal avec mainTemplate

Si vous souhaitez que votre action génère un contenu directement, sans passer par le **template principal**, vous pouvez le faire en mettant l'option mainTemplate à null.

```
function processUneActionDirecte (){
    $ppo = new CopixPpo ();
    $ppo->arData = _dao ('nomTable')->findAll ();
    return _arPPO ($ppo, array ('template'=>'template.xhtml.tpl',
            ✂ 'mainTemplate'=>null));
}
```

Il existe également un raccourcis pour ce faire, _arDirectPpo

```
function processUneActionDirecte (){
    $ppo = new CopixPpo ();
    $ppo->arData = _dao ('nomTable')->findAll ();
    return _arDirectPPO ($ppo, 'template.xhtml.tpl');
}
```

Spécifier cache-control

Vous pouvez spécifier la valeur de l'en tête http Cache-Control avec l'option cache-control.

```
function processUneActionSansCache (){
   $ppo = new CopixPPO ();
   $ppo->TITLE_PAGE  = 'Titre de ma page';
   return _arPpo ($ppo, array ('template'=>'template_pour_ppo.tpl', 'cache-control'
            ✂ =>'no-cache'));
}
```

CopixActionReturn_REDIRECT

Présentation

Le type REDIRECT est utilisé pour demander à Copix de faire une redirection vers une autre URL dans le code retour de vos **actions**.

Exemple :

```
function processVaSurCopixPointOrg (){
   return new CopixActionReturn (CopixActionReturn::REDIRECT, 'http://www.copix.org');
   //ou le raccourcis
   return _arRedirect ('http://www.copix.org');

   //ou en utilisant des url Copix
   return _arRedirect (_url ('||'));//va sur la page par défaut de notre site
}
```

CopixActionReturn_FILE

Présentation

Le code retour CopixActionReturn::FILE sert à demander à Copix de télécharger un contenu qui existe sous la forme d'un fichier sur le serveur vers le poste client.

Ce code est pratique lorsque le contenu est disponible sur le serveur et que vous souhaitez utiliser Copix pour le fournir (par exemple pour contrôler si l'utilisateur à le droit de visualiser ce dernier).

Exemple :

```
return _arFile ('/tmp/monfichier.zip');
//ou avec la syntaxe complète
return new CopixActionReturn (CopixActionReturn::FILE, '/tmp/monfichier.zip');
```

Options supplémentaires

Il est possible de spécifier des options supplémentaires lorsque l'on demande le téléchargement de contenu, soit :

- la disposition du fichier (content-disposition)
- le nom du fichier (filename) (déterminé automatiquement à partir du chemin du fichier si non spécifié)
- le type mime du contenu (content-type, option par défaut) (déterminé automatiquement à partir du nom du fichier si non spécifié)
- le mode de transfert du fichier (content-transfer-encoding)

Les options sont passées sous la forme d'un tableau associatif.

Disposition du fichier

Cette option peut prendre deux valeurs :

- inline (valeur par défaut si vous ne spécifiez rien)
- attachement

Exemple :

```
return _arFile ('/tmp/monfichier.zip', array ('content-disposition'
        ✂ =>'attachement'));
//et avec inline
return _arFile ('/tmp/monfichier.zip', array ('content-disposition'=>'inline'));
```

Spécifier un nom de fichier

Cette option permet d'indiquer au navigateur le nom du fichier à télécharger. Cela évite que le fichier prenne le nom du script php qui génère le contenu.

Exemple :

```
return _arFile ('/tmp/monfichier.zip', array ('filename'=>'archive.zip'));
```

Spécifier le type MIME du fichier

Cette option permet d'indiquer le type du contenu que nous sommes en train d'envoyer au navigateur.

Si rien n'est spécifié et qu'aucun nom de fichier (filename) n'a été spécifié, le type sera déterminé à partir du chemin du fichier sur le serveur. Si rien n'est spécifié et qu'un nom de fichier à été donné, le type sera déterminé en fonction de l'extension du nom de fichier demandé. Sinon, le type sera celui spécifié.

Exemples :

```
//Le type MIME sera celui du zip car le chemin du fichier indique "monfichier.zip"
return _arFile ('/tmp/monfichier.zip');
//Le type MIME sera celui de tar.gz car on a spécifié un nom de fichier avec tar.gz
return _arFile ('/tmp/monfichier.zip', array ('filename'=>'archive.tar.gz'));
//le type MIME sera celui du "gz"" car on le spécifie explicitement
return _arFile ('/tmp/archive.zip', array ('content-type'
        ✂ =>CopixMIMEType::getFromExtension ('.gz'))));
```

Note Ici, on utilise **CopixMIMETypes** pour récupérer le type MIME du fichier. **CopixMIMETypes** prend en charge la plupart des formats connus.

Note Le type du fichier est la seule option qui n'a pas besoin d'être passée sous la forme de tableau, ainsi vous pouvez écrire :

```
return _arFile ('/tmp/monfichier.zip', CopixMIMEType::getFromExtension ('.zip')));
```

Mode de transfert

Cette option permet de spécifier au navigateur comment transférer le contenu que l'on lui envoie.

Par défaut, Copix utilise "binary".

Exemple:

```
return _arFile ('/tmp/monfichier.zip', array ('content-transfer-encoding'
        ✂ =>'binary')));
```

CopixActionReturn_CONTENT

Présentation

Le code retour CopixActionReturn::CONTENT sert à demander à Copix de télécharger un contenu vers le poste client.

Ce code est pratique lorsque le contenu est généré à la volée et que vous ne souhaitez pas le conserver sur le serveur.

Exemple :

```
return _arContent ($generateurDeFichier->generate ($data, 'zip'));
//ou avec la syntaxe complète
return new CopixActionReturn (CopixActionReturn::CONTENT,
            ✂ $generateurDeFichier->generate ($data, 'zip'));
```

Options supplémentaires

Il est possible de spécifier des options supplémentaires lorsque l'on demande le téléchargement de contenu, soit :

- la disposition du fichier (content-disposition)
- le nom du fichier (filename)
- le type mime du contenu (content-type, option par défaut)
- le mode de transfert du fichier (content-transfer-encoding)

Les options sont passées sous la forme d'un tableau associatif.

Disposition du fichier

Cette option peut prendre deux valeurs :

- inline (valeur par défaut si vous ne spécifiez rien)
- attachement

Exemple :

```
return _arContent ($generateurDeFichier->generate ($data, 'zip'),
            ✂ array ('content-disposition'=>'attachement'));

//et avec inline
return _arContent ($generateurDeFichier->generate ($data, 'zip'),
            ✂ array ('content-disposition'=>'inline'));
```

Spécifier un nom de fichier

Cette option permet d'indiquer au navigateur quel sera le nom du fichier à télécharger. Cela évite que le fichier à télécharger prenne le nom du script php qui génère le contenu.

Exemple :

```
return _arContent ($generateurDeFichier->generate ($data, 'zip'),
        ✂ array ('filename'=>'archive.zip'));
```

Spécifier le type MIME du fichier

Cette option permet d'indiquer le type du contenu que nous somme en train d'envoyer au navigateur.

Si rien n'est spécifié et qu'aucun nom de fichier (filename) n'a été spécifié, le type sera "application/octet-stream". Si rien n'est spécifié et qu'un nom de fichier à été donné, le type sera déterminé en fonction de l'extension. Sinon, le type sera celui spécifié.

Exemples :

```
//Le type MIME sera application/octet-stream car on ne sait pas comment le
        ✂ déterminer
return _arContent ($generateurDeFichier->generate ($data, 'zip'));
//Le type MIME sera celui du zip grâce au nom du fichier
return _arContent ($generateurDeFichier->generate ($data, 'zip'),
        ✂ array ('filename'=>'archive.zip'));
//le type MIME sera celui du zip car on le spécifie explicitement
return _arContent ($generateurDeFichier->generate ($data, 'zip'),
        ✂ array ('content-type'=>CopixMIMEType::getFromExtension ('.zip')));
```

Note Ici, on utilise **CopixMIMETypes** pour récupérer le type MIME du fichier. **CopixMIMETypes** prend en charge la plupart des formats connus.

Note Le type du fichier est la seule option qui n'a pas besoin d'être passée sous la forme de tableau, ainsi vous pouvez écrire :

```
return _arContent ($generateurDeFichier->generate ($data, 'zip'),
        ✂ CopixMIMEType::getFromExtension ('.zip'));
```

Mode de transfert

Cette option permet de spécifier au navigateur comment transférer le contenu que l'on lui envoie.

Par défaut, Copix utilise "binary".

Exemple:

```
return _arContent ($generateurDeFichier->generate ($data, 'zip'),
        ✂ array ('content-transfer-encoding'=>'binary'));
```

Gestion des utilisateurs

CopixAuth

Présentation

CopixAuth est la classe de base qui vous permet de récupérer l'utilisateur courant ainsi que de le supprimer.

Cette classe est en quelque sorte un point d'entrée vers le système d'authentification de Copix.

Elle propose les deux méthodes statiques suivantes :

- getCurrentUser qui retourne l'utilisateur courant (**CopixUser**)
- destroyCurrentUser qui détruit les informations sur l'utilisateur courant (et de ce fait le déconnecte complètement)

Note Pour déconnecter proprement un utilisateur, il est préférable d'avoir recours à la méthode logout () de **CopixUser**.

Note Il existe un **raccourcis** pour CopixAuth::getCurrentUser () nommé _currentUser ()

Voir aussi

- **Système d'authentification et de droits** dans Copix

CopixUser

Présentation

CopixUser est l'objet qui représente un utilisateur sous Copix.

Vous pouvez accéder à l'utilisateur courant avec le code suivant :

```
$user = _currentUser ();
//ou avec la syntaxe complète
$user = CopixAuth::getCurrentUser ();
```

Vous pouvez grâce à cet objet connaitre la liste des groupes de l'utilisateur, son/ses logins, son/ses identifiants et bien sûr tester ses droits.

Connexion / Déconnexion

Pour demander à un utilisateur de se connecter / déconnecter, vous pouvez utiliser la méthode login et logout.

Connexion

Cette méthode accepte un seul paramètre sous la forme d'un tableau.

Il existe des paramètres réservés dont voici la liste :

- "login" qui sert à spécifier le login que l'on souhaite utiliser pour se connecter
- "password" qui sert à spécifier le mot de passe éventuel à utiliser pour se connecter
- "forceNoPassword" qui sert à demander au handler de ne pas prendre en compte le password (c'est une demande de connexion forcée)

En fonction des handlers, ces paramètres seront ou non pris en compte.

La méthode retourne true si au moins un handler est parvenu à se connecter et false si aucune tentative de connexion n'a réussi.

Pour permettre à vos utilisateurs de se connecter, vous utiliserez en temps normal le module d'authentification fourni avec Copix (index.php/auth/).

Toutefois, si vous souhaitez vous même appeler la méthode de connexion, voici un exemple d'appel :

```
//On essaye de connecter l'utilisateur courant avec le login test et le mot de passe
           ✂ "test_password"
_currentUser ()->login (array ('login'=>'test', 'password'=>'test_password'));
```

Déconnexion

Cette méthode demande de déconnecter l'utilisateur courant.

Comme la fonction de connexion, elle accepte un paramètre sous la forme d'un tableau. Il n'existe à l'heure actuelle aucun paramètre réservé dans ce tableau.

Pour permettre à vos utilisateurs de se déconnecter, vous utiliserez généralement le module d'authentification fourni avec Copix (index.php/auth/default/out).

Toutefois, si vous souhaitez vous même appeler la méthode de déconnexion, voici un exemple d'appel :

```
//on demande de déconnecter l'utilisateur courant
_currentuser ()->logout ();
```

Récupération des informations

Les groupes auquel l'utilisateur appartient

La méthode getGroups récupère dans un tableau la liste des groupes auxquels l'utilisateur appartient.

Ces groupes sont retournés sous la forme d'un tableau a deux dimensions.

- La première dimension contient le handler concerné
- La seconde dimension contient l'identifiant du groupe.
- La valeur contient le libellé du groupe

Voir aussi **les droits de type "group:".**

Savoir si l'utilisateur est connecté

La méthode isConnected indique si l'utilisateur est actuellement connecté. Dans le cas ou plusieurs gestionnaires d'authentification cohabitent, cette méthode indique simplement si "au moins" un gestionnaire a accepté l'utilisateur.

Dans le cas ou la méthode retourne false, alors cela signifie que tous les gestionnaire ont refusé l'authentification.

Si vous souhaitez savoir si votre utilisateur est bien connecté avec un gestionnaire particulier, vous devrez utiliser la méthode "isConnectedWith" (voir plus loin)

Savoir si l'utilisateur est connecté avec un gestionnaire particulier

La méthode isConnectedWith ($nomDuGestionnaire) vous permet de savoir si un utilisateur est correctement connecté à l'application via un gestionnaire d'authentification particulier.

Son utilité est avérée si vous utilisé plusieurs gestionnaires d'authentification simultanéement, sans quoi isConnected devrait être suffisant.

Cette méthode existe depuis Copix 3.0.2+.

L'ancien nom de cette méthode est "isLoggedWith" et fonctionne de façon strictement identique.

Connaitre l'identifiant de l'utilisateur

La méthode getId () retourne l'identifiant de l'utilisateur. Si l'utilisateur n'est pas connecté, alors elle retourne null.

ATTENTION: Dans le cas ou vous utilisez plusieurs gestionnaire d'authentification, cette méthode retourne l'identifiant de l'utilisateur dans le **premier** gestionnaire d'authentification qui a accepté la connexion.

Connaitre le login de l'utilisateur

La méthode getLogin () retourne l'identifiant de l'utilisateur. Si l'utilisateur n'est pas connecté, alors elle retourne null.

ATTENTION: Dans le cas ou vous utilisez plusieurs gestionnaire d'authentification, cette méthode retourne le login de l'utilisateur dans le **premier** gestionnaire d'authentification qui a accepté la connexion.

Connaitre le "libellé" de l'utilisateur

La notion de libellé corresponds au "nom complet" de l'utilisateur, pas simplement son login. Tous les gestionnaires ne sont pas obligés de le gérer, auquel cas cette méthode retourne le login de l'utilisateur.

La méthode getCaption () retourne donc le libellé de l'utilisateur. Si l'utilisateur n'est pas connecté, alors elle retourne null.

ATTENTION: Dans le cas ou vous utilisez plusieurs gestionnaire d'authentification, cette méthode retourne le libellé de l'utilisateur dans le **premier** gestionnaire d'authentification qui a accepté la connexion.

Récupération des informations de connexion

Lorsqu'un utilisateur se connecte, les gestionnaires d'authentification apportent des réponses sous la forme de **CopixUserLogResponse**.

Si vous souhaitez connaitre ce qu'ils ont répondu lors de l'acceptation de la connexion, la méthode getResponses () retourne l'ensemble des réponses des gestionnaires qui ont répondu favorablement.

Si vous ne souhaitez connaitre la réponse que d'un seul gestionnaire en particulier, vous pouvez utiliser la méthode getHandlerResponse ($nomDuGestionnaire). Seuls les réponses des gestionnaires ayant répondu favorablement sont accessibles via cette méthode.

Tester les droits de l'utilisateur

Lorsque vous voulez tester les droits de l'utilisateur courant, vous utilisez les méthodes testCredential et assertCredential.

Toute deux acceptent en paramètre une chaine de droit en fonction des **gestionnaires de droits** configurés dans l'application.

- testCredential retourne un booléen indiquant si l'utilisateur dispose ou non des droits.
- asserCredential lance une exception de type **CopixCredentialException** si jamais l'utilisateur ne dispose pas des droits demandés.

Authentification

Présentation

Le système d'authentification et de droit dans Copix est architecturé autours de 3 éléments, tous interchangeables, autours du principe d'une chaine de responsabilité.

Ce système est particulièrement puissant et vous permet de remplacer (ou simplement compléter) tout ou partie des gestionnaires de base livrés avec Copix, pour les adapter parfaitement à vos besoins applicatifs.

Avant de rentrer dans une explication détaillée, disons en une simple phrase qu'il existe des utilisateurs, rassemblés dans des groupes, groupes auxquels sont associés un certain nombre de droits.

Gestionnaires d'authentification

Les gestionnaires d'authentification permettent de gérer les utilisateurs (objets **CopixUser**). C'est leur seul et unique rôle.

Les gestionnaires d'authentification implémentent tous l'interface **ICopixUserHandler**.

Ils gèrent ainsi les méthodes permettant de se connecter et se déconnecter. En plus de ces deux méthodes, ils permettent la récupération d'informations complémentaires sur l'utilisateur, ainsi que la recherche parmi une liste et selon certains critères.

Gestionnaires de groupes

Les gestionnaires de groupes permettent de regrouper un ensemble d'utilisateurs (quels que soient les gestionnaires d'authentification dont ils sont issus).

Les gestionnaires de groupes implémentent tous l'interface **ICopixGroupHandler** qui permet de récupérer les groupes auquel appartient un utilisateur donné ainsi que des informations sur les groupes en eux même.

Gestionnaires de droits

Les gestionnaires de droit permettent de tester les droits d'un utilisateur. Dans copix, il est possible de définir des types de droits différents. En standard, Copix gère les droits de type **group**, **basic**, **module** et **dynamic**.

Les gestionnaires de droits implémentent tous l'interface **ICopixCredentialHandler** et doivent définir une seule méthode qui vérifie les droits d'un utilisateur.

ICopixUserHandler

Présentation

ICopixUserHandler est l'interface que devront respecter l'ensemble des gestionnaires d'authentification que vous serez amené à développer.

```
interface ICopixUserHandler {
    public function login  ($pParams);
    public function logout ($pParams);
    public function getInformations ($pUserId);
    public function find ($pParams = array ());
}
```

Connexion / Déconnexion

Les méthodes de connexion / déconnexion acceptent en entrée un paramètre sous la forme d'un tableau associatif.

A l'heure actuelle, seule la méthode de login dispose de clefs réservées, à savoir :

- login : le login demandé pour la connexion
- password : le mot de passe en clair demandé pour la connexion

Les méthodes de connexion / déconnexions retourne une réponse sous la forme d'un objet de type CopixUserLogResponse.

Informations sur l'utilisateur

La méthode getInformations accepte en paramètre l'identifiant de l'utilisateur pour le handler donné.

Elle retourne un objet de type quelconque avec autant de propriétés que d'informations supplémentaires à connaitre sur notre utilisateur.

Recherche d'utilisateurs

La méthode find est chargée de rechercher des utilisateurs en fonction de plusieurs critères.

Les critères ne sont pas figés et vous pouvez en prendre en charge autant que vous le souhaitez. Toutefois, votre méthode se doit d'être capable de rechercher les utilisateurs en fonctions des critères suivants :

- id pour l'identifiant de l'utilisateur
- login pour le login de l'utilisateur
- caption pour le nom complet de l'utilisateur

ICopixGroupHandler

Présentation

L'interface ICopixGroupHandler est l'interface de base de tous les gestionnaires de groupe.

```
interface ICopixGroupHandler {
        public function getUserGroups ($pUserId, $pUserHandler);
        public function getInformations ($pGroupId);
}
```

Les gestionnaires de groupes doivent être capable de gérer des utilisateurs en provenance de plusieurs gestionnaire d'authentification.

Récupération des groupes d'un utilisateur

La méthode getUserGroups accepte deux paramètres :

- $pUserId est l'identifiant de l'utilisateur dont on souhaite connaitre les groupes
- $pUserHandler est le gestionnaire d'authentification auquel appartient l'utilisateur dont on souhaite connaitre les groupes.

Elle retourne un tableau, avec pour clefs les identifiants des groupes, et pour valeurs les noms des groupes.

Depuis Copix 3.0.2+, cette méthode est appelé pour les utilisateurs anonymes. Dans ce dernier cas, null est passé pour $pUserId et $pUserHandler. C'est une méthode pratique pour vous permettre de gérer des groupes "publics" afin d'assigner des droits à tout internaute sur votre application.

Récupération des informations sur un groupe

La méthode getInformations permet de récupérer l'ensemble des informations connues sur un groupe d'identifiant donné. Le retour de la méthode est un objet sans type prédéfini avec autant de propriétés publiques que d'informations à récupérer.

ICopixCredentialHandler

Présentation

ICopixCredentialHandler est l'interface de base des gestionnaires de droit.

```
interface ICopixCredentialHandler {
        public function assert ($pStringType, $pString, $pUser);
}
```

Tester les droits

Les gestionnaires de droits n'ont pour tâche que le test des droits d'un utilisateur donné.

Pour ce faire, il doivent implémenter la méthode "assert" qui retourne un booléen true ou false si l'utilisateur dispose ou non des droits en question.

La méthode reçoit trois paramètres :

- $pStringType le type de droit (en standard **basic**, **group**, **module**, ou **dynamic**)
- $pString La chaine de droit à tester (sans le type de droit)
- $pUser L'utilisateur courant sous la forme d'un objet **CopixUser**

En cas de tests multiples pour un même couple utilisateur / clef, Copix se souvient des résultats et n'appellera votre gestionnaire qu'une seule fois.

Exemples de tests de droits

```
_currentUser ()->assertCredential ('basic:registered');//on demande à ce que
            ✂ l'utilisateur soit connecté
_currentUser ()->assertCredential ('group:[admin]');//on demande à ce que l'utilisateur
            ✂ appartienne à un groupe nommé "admin"
_currentUser ()->assertCredential ('module:write@news');//on demande à ce que
            ✂ l'utilisateur dispose du droit "write" déclaré dans le module "news"
```

CopixCredentialException

Présentation

CopixCredentialException est l'exception qui est lancée par la méthode assertCredential de l'objet **CopixUser** lorsque l'utilisateur ne dispose pas des droits demandés.

CopixUserLogResponse

Présentation

CopixUserLogResponse est la classe de base des objets qui sont retournés par les **gestionnaires d'authentification** lors des opérations de connexion et déconnexion.

construction

```
__construct ($pOk, $pHandler, $pId, $pLogin, $pExtra = array ()){
```

- $pOk indique si l'opération à réussi
- $pHandler indique le handler qui a pris en charge l'opération
- $pId indique l'utilisateur qui a été touché par l'opération
- $pLogin indique le login de l'utilisateur qui a été touché
- $pExtra est un tableau d'informations complémentaires

Credential Group

Présentation

Les **chaines de droit** de type groupe vous permettent de définir des droits "par groupe", c'est à dire que le droit va vérifier que votre utilisateur appartient bien à un groupe d'utilisateur donné.

C'est un système facile d'utilisation dont voici quelques exemples :

```
_currentUser ()->testCredential ('group:1');//on regarde si l'utilisateur courant
            ✂ appartient a un groupe d'identifiant 1

_currentUser ()->testCredential ('group:[admin]');//on regarde si l'utilisateur
            ✂ courant appartient à un groupe dont le libellé est "admin"

_currentUser ()->testCredential ('group:1@dbgrouphandler');//on regarde si
            ✂ l'utilisateur courant appartient au groupe d'identifiant 1 dans le
            ✂ gestionnaire de groupe de type 'dbgrouphandler'

_currentUser ()->testCredential ('group:[admin]@dbgrouphandler');//on regarde si
            ✂ l'utilisateur courant appartient à un groupe dont le libellé est
            ✂ "admin" dans le gestionnaire de groupe de type "dbgrouphandler"
```

Annexe

Constantes Copix

Copix défini plusieurs constantes pour vous faciliter la tâche et pour des besoins internes. En voici la liste :

- COPIX_PROJECT_PATH - Emplacement de votre projet, par défaut "project/"
- COPIX_PATH - Emplacement ou se situe le répertoire copix, par défaut "utils/copix/"
- COPIX_CORE_PATH - Emplacement ou se situent les classes du core copix, par défaut "utils/copix/core"
- COPIX_UTILS_PATH - Emplacement ou se situent les classes utilitaires de copix, par défaut "utils/copix/utils"
- COPIX_VERSION - Représente la version de Copix actuellement utilisée.

projectincphp

Rôle du fichier

Ce fichier, inclus depuis le fichier *www/index.php*, à pour rôle de déclarer les constantes propres au projet ainsi que de définir le **ProjectController**.

Les constantes déclarées dans ce fichier servent avant tout à pouvoir personnaliser les différents emplacement des répertoires Copix, ce qui pourra être nécessaire pour des installations avancées.

Déclaration des constantes projet

COPIX_PROJECT_PATH

COPIX_PROJECT_PATH indique simplement où se situe le fichier project.inc.php. Il est fréquent d'avoir recours à cette constante dans le fichier de configuration **copix.inc.php** pour paramétrer d'autres chemins.

COPIX_TEMP_PATH

COPIX_TEMP_PATH correspond à la racine des répertoires temporaires qui seront utilisés par Copix. Copix utilise le répertoire temporaire pour placer les fichiers "transformés" de DAO, les fichiers de paramètres, les fichiers i18n, la liste des modules installés, ...

COPIX_CACHE_PATH

COPIX_CACHE_PATH correspond à la racine du répertoire où seront stockés les fichiers de cache générés via **CopixCache**. Si vous souhaitez que deux applications Copix profitent du même cache, vous pouvez faire pointer cette constante sur le même répertoire.

COPIX_LOG_PATH

COPIX_LOG_PATH correspond à la racine du répertoire ou seront stockés les fichiers de log (utilisant la stratégie de sauvegarde "FILE") via **CopixLog** (ou le [[shortcut|raccourcis] _log ( ))

COPIX_VAR_PATH

COPIX_VAR_PATH correspond à la racine du répertoire où, par convention, Copix stocke l'ensemble des documents non temporaires manipulés par l'application. Si vous développez des modules qui génèrent des documents, qui permettent à l'utilisateur d'uploader un certain nombre d'éléments, c'est un bon emplacement pour le faire. Ce répertoire contient par exemple les fichiers de configuration que vous manipulez via l'interface d'administration (/config/_db_profiles, _log_profiles, _plugins)

Déclaration du ProjectController

Le ProjectController est la classe principale en charge du lancement de l'application. Sa surcharge vous laisse l'opportunité d'en modifier le comportement.

Voir la documentation du **ProjectController**.

copixconfphp

Présentation

Le fichier copix.conf.php gère les options du framework et se situe dans *project/config/copix.conf.php*.

Fonctionnement des URL

Aujourd'hui, il existe deux modes de fonctionnement pour les URL :

- default (envois de paramètres)
- prepend (utilisations de chemins)

URLs classiques (default)

Par exemple : *index.php?module=nom&desc=default&action=do*

```
$config->significant_url_mode = 'none'; // Utilisation des url "normales"
```

URLs significatives (prepend)

Par exemple : *index.php/nom/default/do*

```
$config->significant_url_mode = 'prepend'; // Utilisation des url significatives
        ✂ type "prepend"
```

Gestionnaire de cache

Copix propose en standard une classe capable de gérer des caches de contenu : CopixCache. Vous pouvez activer ou non ce cache. Par exemple, sur un serveur de développement, vous souhaitez que le cache ne fonctionne jamais pour être sûr que vos fonctions dynamiques sont systématiquement appelées :

```
$config->cacheEnabled = false; // Le cache est toujours inactif
```

Les autres opérations de configuration du cache se configurent dans les interfaces livrées avec le module admin.

Langues, Pays & charset

Ressources & charset

Vous pouvez indiquer la langue et le charset par défaut de votre site :

```
$config->default_language = 'fr';
$config->default_country = 'FR';
$config->default_charset = 'ISO-8859-1';
```

Copix n'intègre en standard que les langues anglais (en) et français (fr). Si vous souhaitez utiliser d'autres langues, il vous faudra traduire les fichiers de ressources. Voir le chapitre **internationalisation** pour plus de détails.

Note : Si vous souhaitez que la langue soit détectée automatiquement, vous pouvez enregistrer le plugin i18n dédié à cet effet.

Sélection des templates en fonction de la langue

Vous pouvez demander à Copix de sélectionner les templates spécialisés en fonction de la langue courante. Activer cette option consomme des ressources, ainsi, nous recommandons de la désactiver si vous n'en profitez pas.

```
$config->i18n_path_enabled = false;
```

Erreurs de clefs

Vous pouvez demander à Copix de générer des erreurs lorsqu'on lui demande une clef i18n inexistante, ou simplement lui demander de retourner comme chaine le nom de la clef demandée.

```
//Provoquera une erreur si l'on demande une clef inexistante. Mettre "null" en
        ✂ production pour mettre ces erreurs sous silence.
$config->missingKeyTriggerErrorLevel = E_USER_ERROR;
```

Templates

Copix utilise Smarty en standard comme moteur de templates. Pour configurer Smarty, Copix publie les seules options dont vous aurez besoin. En standard, vous n'aurez pas à modifier ces paramètres :

```
$config->template_dir = COPIX_PROJECT_PATH.'templates/'; // Le répertoire ou sont
        ✂ stockés les templates
$config->compile_dir = COPIX_CACHE_PATH.'tpl_compile/'; // Le répertoire ou sont
        ✂ compilés les templates
$config->config_dir = './configs'; // Le répertoire ou sont situés les fichiers de
        ✂ configuration
$config->caching = 0; // Le mode de fonctionnement du cache
$config->cache_dir = COPIX_CACHE_PATH.'tpl_cache/'; // Le répertoire de cache
```

Note : Consultez la documentation de Smarty pour plus de renseignements sur ces paramètres de configuration.

Template principal de l'application

Dans Copix, vous utilisez un **template principal** pour votre application. Pour définir le template principal par défaut de votre application, vous utiliserez le paramètre $mainTemplate :

```
$config->mainTemplate = 'default|main.ptpl';
```

Note : Le nommage du template principal reprend les principes des sélecteurs dans Copix (module|nomElement).

Paramètres de compilation

L'une des particularités de Copix est de compiler ses fichiers de configuration XML (**CopixConfig**), properties (**CopixI18N**), DAO (**CopixDAO**) en PHP. C'est à dire qu'une fois analysés, les fichiers de configuration sont transformés en vrai PHP et donc exécutés aussi rapidement que si c'était le développeur lui même qui avait écrit la fonctionnalité "en dur".

Copix est capable de détecter seul si votre fichier de configuration a été modifié et donc s'il doit de nouveau générer le source en conséquence. Toutefois, il vous est possible de modifier ce mode de fonctionnement (que nous nommons compile_check), en fonction de vos besoin.

Vérifier les mises à jour pour la compilation (compile_check, mode par défaut)

Par défaut, avant d'utiliser un élément qu'il à transformé en PHP, Copix vérifie si vos fichiers de configuration relatifs ont étés modifiés (en comparant simplement la date système des éléments en jeu).

Pour obtenir ce fonctionnement, voici les paramètres de compilation que vous devez avoir dans vos fichiers de configuration :

```
$config->force_compile  = false;
$config->compile_check  = true;
```

Note : c'est la configuration recommandée durant la phase de développement.

Forcer la compilation (force_compile)

Si pour une raison ou pour une autre vous pensez que le système n'est pas capable de savoir si vos fichiers ont étés modifiés depuis leurs dernière compilation (par exemple horloges systèmes décalées sur les différentes machines), vous pouvez demander à Copix de toujours compiler les éléments.

Pour obtenir ce fonctionnement, voici les paramètres de compilation que vous devez avoir dans vos fichiers de configuration :

```
$config->force_compile  = true;
$config->compile_check  = true;//comme force_compile est à vrai, ce paramètre à peu
           ✂ d'importance
```

Note : Forcer la compilation des éléments est très coûteux en terme de performances. N'activez jamais cet élément en production, quand bien même vous souhaitez "tout recompiler". En production, préférez vider les répertoires de cache. Les paramètres "force" ont la priorité sur les paramètres "check". C'est à dire que "check" n'est pas pris en compte si "force" est actif.

Ne jamais vérifier la compilation

A des fins d'optimisation et de gains de performances, vous pouvez vouloir désactiver la vérification de compilation. Vous gagnerez ainsi tous les temps de contrôle sur les fichiers résultants. Désactiver le contrôle ne signifie pas que les fichiers ne seront pas compilés, mais que si les fichiers compilés existent, Copix ne regardera pas s'il devrait les mettre à jour.

```
$config->force_compile  = false;
$config->compile_check  = false;
```

Note : Ne pas vérifier la compilation des éléments est un paramètre que nous recommandons de mettre en production. Les gains de performance sont significatifs.

Emplacement des modules

Copix est capable d'aller chercher les modules à différents emplacements. Ces emplacements sont configurables dans la variable arModulesPath.

Seuls les modules situés dans ces emplacements pourront être installés.

```
$config->arModulesPath = array (COPIX_PROJECT_PATH.'modules/public/stable/standard/',
COPIX_PROJECT_PATH.'modules/public/stable/cms/',
COPIX_PROJECT_PATH.'modules/public/devel/bench/',
COPIX_PROJECT_PATH.'modules/public/devel/tools/'
);
```

Restreindre la liste des modules exécutables

Sur certains sites, pour des raison de sécurité par exemple, vous pouvez être amené à vouloir désactiver l'exécution directe de certains modules (parametters, install, ..., ...).

Pour ce faire, vous pouvez activer le mode "trustedModules" qui n'autorisera alors l'exécution des modules que s'ils ont étés explicitement ajoutés à la liste des modules autorisés (arTrustedModules)

Dans l'exemple ci dessous, seul le module bench_news peut être appelé via l'installation courante.

```
$config->checkTrustedModules = true;
$config->arTrustedModules = array ('bench_news'=>true, 'default'=>true);
```

Authentification

Toujours vérifier les droits au moment de la demande (3.0.1+).

```
$config->copixauth_cache = false;
```

En standard, ce paramètre est positionné à true, ce qui fait que Copix n'interroge le système de droit qu'une fois par chaine et se souvient ensuite du résultat pour toute la session.

C'est très pratique pour optimiser les performances, mais cela peut être gênant lorsque vous êtes en phase de développement.

ProjectController

Présentation

La classe ProjectController est instanciée dans *www/index.php* et c'est elle qui prend en charge l'application complète. Sa déclaration est située dans project/**project.inc.php**.

C'est le Front Controller de l'application.

Cette classe hérite de **CopixController** et en standard ne surcharge que la méthode _processStandard.

Cette méthode processStandard a pour objectif de réaliser un certain nombre d'opérations communes à chaque **action d'affichage** (DISPLAY / PPO / DISPLAY_IN) et vous donne ainsi l'opportunité d'assigner des variables par défaut à votre template principal, ou alors de gérer un certain nombre d'élément de contexte (les menus, les pieds de page, ...)

Nous ne recommandons pas de surcharger d'autres méthodes que *_processStandard*.

_processStandard

Cette méthode reçoit en paramètre les données du **template principal** avant qu'on n'en demande l'affichage.

Ainsi, vous avez une dernière opportunité d'en modifier les données ou de rajouter des éléments à ce template.

En standard, vous verrez que c'est ici que nous assignons les valeurs par défaut de TITLE_PAGE et TITLE_BAR.

```
        function _processStandard ($tplObject) {
                $tplVars = $tplObject->getTemplateVars ();

                if (! isset ($tplVars['TITLE_PAGE'])) {
                        $tplVars['TITLE_PAGE'] = CopixConfig::get ('|titlePage');
                        $tplObject->assign ('TITLE_PAGE', $tplVars['TITLE_PAGE']);
                }

                if (! isset ($tplVars['TITLE_BAR'])) {
                        $tplVars['TITLE_BAR'] = str_replace ('{$TITLE_PAGE}',
                                ✂ $tplVars['TITLE_PAGE'], CopixConfig::get
('|titleBar'));
                        $tplObject->assign ('TITLE_BAR', $tplVars['TITLE_BAR']);
                }
                //...
```

Plus loin, vous voyez également que nous nous servons de ce processStandard pour assigner "en dur" quelques éléments de menu ($menuItems).

Ce n'est pas la seule façon de procéder

Cette méthode processStandard n'est pas la seule opportunité que vous avez de placer des éléments à votre template principal, même si c'est la méthode la plus rapide et efficace pour préparer des données systématiques propres à votre projet.

Vous pouvez également utiliser

- des plugins (portables et capables de modifier un ensemble de données sur votre page avant affichage)
- Des zones appelées depuis votre template principal (pour générer une partie du contenu).

URLHandler

Présentation

La classe CopixURLHandler est la classe de base des gestionnaires d'url que vous pouvez déclarer dans vos modules pour créer des URL significatives.
Cette classe est abstraite.

En standard, Copix génère des URL de deux formes différentes :

- index.php?module=nomModule&group=nomGroup&action=nomAction en mode "default"
- index.php/module/group/action en mode "prepend"

Copix propose également un système de réécriture d'URL sous la forme de classes vous permettant de personnaliser complètement la forme de ces dernières dans vos applications.

Ces classes, par convention, se situent dans le répertoire classes du module et portent le nom "nommodule.urlhandler.php".

Ainsi, pour un module "exemple", le gestionnaire d'url sera situé dans exemple/classes/exemple.urlhandler.php et héritera de CopixUrlHandler :

```
class UrlHandlerExemple extends CopixUrlHandler {
   public function get ($dest, $parameters, $mode){
   }
   public function parse ($path, $mode){
   }
}
```

Méthode get

Cette méthode est en charge de générer les URL. Elle reçoit en paramètre :

- le trigramme module|group|action sous la forme d'une chaine de caractère
- le tableau des paramètres demandés
- le mode de gestion de URL

Elle devra retourner un objet de type CopixUrlHandlerGetResponse pour décrire complètement l'url.

Si votre gestionnaire souhaites indiquer qu'il ne gère pas l'URL reçue, vous retournerez "false".

Par exemple, si vous souhaitez générer l'URL http://www.votresite.org/chemin/vers/index.php/chemin/complet?parametre=valeur, vous auriez retourné cette réponse dans votre handler :

```
$obj = new CopixUrlHandlerGetResponse ();
$obj->path = array ('chemin', 'complet');
$obj->vars = array ('parametre'=>'valeur');
return $obj;
```

Méthode parse

Cette méthode est en charge d'analyser l'URL demandée par l'internaute et de la transformer en trigramme copix (module|group|action) avec éventuellement des paramètres supplémentaires.

L'exemple ci dessous permet d'indiquer à Copix qu'il faut, pour les appels aux urls de la forme "index.php/les_nouvelles/1/News-du-jour" réagir comme si l'internaute avait saisis index.php/les_nouvelles/front/read?news=1

```
class UrlHandlerExemple extends CopixUrlHandler {
   public function parse ($path, $mode){
      //Dans $path on obtient la liste
      if (count ($path) == 2){
         if (($path[0] == 'les_nouvelles') && is_integer ($path[1])){
            return array ('module'=>'les_nouvelles', 'group'=>'front',
'action'=>'read', 'news'=>1);
         }
      }
   }
}
```

CopixUrlHandlerGetResponse

Présentation

La classe CopixURLHandlerGetResponse décrit les composantes d'une URL lors de sa génération par les handlers spécifiques.

Lorsque vous développez un handler d'URL, vous pouvez renseigner différents paramètres, à savoir :

le chemin ($path)

Vous spécifiez le chemin via un tableau de chaines de caractères.

Par exemple, si vous souhaitez générer le chemin /module/et/son/chemin, alors vous initialiserez path avec array ('module', 'et', 'son', 'chemin').

Par défaut $path est initialisé tableau vide (array ()).

Note : Chaque élément du chemin subira un "urlencode" et verra ses espaces remplacés par des tirets "-".

les paramètres ($vars)

Vous pouvez spécifier les paramètres à envoyer à l'URL. Par exemple, si vous souhaitez avec les paramètres ?p1=v1&p2=v2, vous donnerez à $vars la valeur array ('p1'=>'v1', 'p2'=>'v2').

Par défaut $vars est initialisé à tableau vide (array ()).

Note : Chaque valeur des variables subira un "urlencode".

Spécifier le nom du script à utiliser ($scriptName)

Vous pouvez spécifier le script à appeler.

Par défaut le script est initialisé avec le script de l'url actuelle. (index.php par défaut dans les URL Copix)

Le chemin de base de l'URL ($basePath)

Vous pouvez spécifier le chemin qui mène à votre script, nom de domaine inclus.

Par exemple, si vous souhaitez que le chemin soit copix.org/monsite/, alors vous spécifierez 'copix.org/monsite/'.

Par défaut, ce chemin est initialisé au chemin de l'URL actuelle.

Note : Ce chemin doit comprendre tous les slashs et caractères spéciaux. (exemple chemin/de/base/)

Le protocole ($protocol)

Vous pouvez spécifier le protocole à utiliser, sous la forme d'une chaine de caractère. Par défaut, la valeur du protocole est le protocole utilisé par l'URL actuelle (http ou https).

Note : Le protocole doit comprendre les caractères spéciaux. (exemple: http://, https:, *ftp:*, ...)

Exemple complet

Si vous souhaitez générer l'URL http://www.copix.org/espace/index.php/le/chemin/complet?parametre=valeur, vous auriez retourné cette réponse dans votre handler :

```
$obj->protocol = 'http://';
$obj->basePath = 'www.copix.org/espace/';
$obj->scriptName = 'index.php';
$obj->path = array ('le', 'chemin', 'complet');
$obj->vars = array ('parametre'=>'valeur');
```

CopixLog

CopixLog

CopixLog est une classe qui vous permet de sauvegarder tout ce qui se passe sur votre site. Vous pouvez demander à Copix de loguer les éléments en base de données, dans des fichiers, simplement en session ou à l'écran.

Cette classe est donc très pratique en développement (pour déboguer vos applications) et en production (pour monitorer les erreurs qui surviennent sur votre site).

Configurer les logs

Configurer les logs se réalise via l'interface d'administration, dans la partie Administration des logs".

Une interface vous permet alors de créer vos profils de log.

Lorsque vous ajoutez un profil de log, il vous sera demandé plusieurs informations :

- Nom : Le nom que vous voulez donner à votre profil de log.
- Stratégie : La façon dont vous voulez que vos logs soient sauvegardés. En standard, 5 stratégies sont disponibles
  - Fichier (qui sauvegarde les données dans un fichier)
  - Base de données (dans la table CopixLog)
  - System (qui utilise la fonction PHP error_log)
  - Session (qui place les données en session)
  - Page courante (qui affiche directement les éléments sur la page).
  - Si aucune de ces stratégie vous convient, Copix vous laisse la possibilité de spécifier votre propre stratégie, auquel cas il vous faut spécifier un sélecteur Copix pour votre classe, qui doit implémenter **ICopixLogStrategy**.
- Les messages interceptés. Si vous ne voulez loguer que certains types de messages (vous pouvez spécifier des types de message lorsque vous envoyez un log).
- A partir de quel niveau vous souhaitez que le message soit logué (Information, avertissements, exceptions, erreurs, ...)
- et enfin si le log est actif ou non.

Utilisation

Ajouter un log

```
CopixLog::log ($message, $type, $niveau, $arInfos);
```

- $message correspond au message que vous voulez logger
- $type est une désignation du type de log afin de l'identifier parmis les autres messages (par exemple les requêtes sont loguées en tant que type "query").
- $niveau est le niveau d'importance du log, il en existe 6 :
  - CopixLog::INFORMATION
  - CopixLog::NOTICE
  - CopixLog::WARNING
  - CopixLog::EXCEPTION
  - CopixLog::ERROR
  - CopixLog::FATAL_ERROR
- arInfos (facultatif) permet de logger des informations supplémentaires ('file', 'line', 'function', 'class', 'user'). Veuillez noter que même si ces informations ne sont pas présentes, Copix tentera de les déterminer de lui même.

Note : Il existe une fonction raccourcis _log pour CopixLog::log ();

Lire les logs

Bien qu'il existe une interface (dans le module admin) pour vous permettre de parcourir les logs, il vous est possible d'utiliser du code pour se faire avec la méthode getLog.

```
$my_content = CopixLog::getLog ($profil, $niveau);
```

- $profil définit quel est le profil à récupérer.
- $niveau (facultatif), définit quel est le niveau à récupérer.

Vider un profil de log

Pour supprimer le contenu d'un profil de log, il vous suffit d'utiliser la méthode deleteProfil en spécifiant le nom du profil à supprimer.

```
CopixLog::deleteProfil ($profil)
```

ICopixLogStrategy

Présentation

L'interface ICopixLogstrategy vous permet de créer vos propres classes pour loguer de façon personnalisée les éléments de **CopixLog**.

Cette interface est définie comme suit :

```
interface ICopixLogStrategy {
    /**
    * Enregistre une ligne de log (dans le profil $pProfil, de type $pType, de niveau
$pLevel, à la date $pDate, avec comme message $pMessage et les informations $pArExtra)
    */
    public function log    ($pProfil, $pType, $pLevel, $pDate, $pMessage, $pArExtra);

    /**
    * Récupère le contenu d'un profil de log $pProfil
    */
    public function getLog ($pProfil);

    /**
    * Supprime le contenu d'un profil de log donné
    */
    public function deleteProfile ($pProfil);
}
```

CopixResource

CopixResources

Copix possède un système vous permettant de gérer plus facilement vos ressources (css, image, js ...). Ce système est simple, quand vous voulez afficher une image, linker un fichier css, etc... vous ne mettez jamais son chemin en dur, mais vous utilisez copixresources.

Exemple en php

```
CopixUrl::getResource ("styles/styles_copix.css");
// ou la syntaxe racourcie
_resource ("styles/styles_copix.css");
// retourne une chaine du type
            ✂ http://www.monsite.com/./themes/mon_theme/styles/styles_copix.css
```

Exemple en smarty

avec le **tag copixresource**

```
{copixresource path="styles/styles_copix.css"}
```

Récupérer le chemin physique vers le fichier

getResource renvoie une url vers le fichier passé en paramètre. Mais vous pouvez avoir besoin du chemin physique

```
CopixUrl::getResourcePath ('styles/styles_copix.css');
// ou la syntaxe raccourcie
_resourcePath ('styles/styles_copix.css');
// retourne une chaine du type ./themes/mon_theme/styles/styles_copix.css

// récupérer le chemin complet depuis la racine du disque dur
getcwd () . '/' . _resourcePath ('styles/styles_copix.css');
// retourne une chaine du type
            ✂ /var/www/html/mon_site/www/./themes/mon_theme/styles/styles_copix.css
```

uniquement depuis www, sans http://www.monsite.com/ devant. Dans ce cas, il faut utiliser getResourcePath :

L'appel au ressources de cette manière aura l'effet suivant :

Si vous êtes dans le thème par défaut, les fichiers utilisés seront dans www/themes/default, dans notre exemple nous utiliserons donc www/themes/default/style/styles_copix.css Si vous avez séléctionné un thème spécifiquement dans la configuration, il prendra en compte ce choix dans le chemin et vous donnera www/themes/votre_theme donc dans notre exemple www/themes/votre_theme/styles/styles_copix.css

Ce système, vous permet donc de gérer des affichages différents, sans pour autant changer les fichiers templates par défaut.

CopixI18N

Internationalisation des pages

CopixI18N permet d'internationaliser votre application Copix. Ainsi, à partir d'un seul template, vous êtes en mesure d'adapter le contenu des pages générées en fonction de la langue courante du navigateur.

Les fichiers properties

Il faut tout d'abord créer des fichiers pour contenir les différentes chaînes traduites. Il y aura un fichier par langue. Vous les rangerez dans le dossier "resources" de votre module.

Chaque fichier suivra la convention de nommage suivante : **groupe[_langue[_COUNTRY]].properties**

Prenons par exemple le fichier *modules/public/stable/default/resources/default_fr.properties*. Celui-ci contient :

```
default.welcome2Copix3 = Bienvenue dans Copix 3 !
default.configureDB = Configurer une base de données
default.configureMoreDB = Configurer une base de données supplémentaire
default.admin = Accéder à l'interface d'administration
default.officialLinks = Liens officiels
default.quickStart = Bien commencer
```

Sur une ligne, la première partie (*default.welcome2Copix3*) permet d'identifier la chaîne ; nous l'appellerons "clé". Notez que toutes les clés commencent par le nom du groupe utilisé pour nommer le fichier.

A la suite, après le signe =, on trouve la valeur de la chaîne traduite dans une langue donnée (ici le français).

Plus généralement, une clé suivra la convention de nommage suivante : **[groupe].[identifiantdechaine] = valeur**

Utilisation

Pour réutiliser une chaîne, il suffit donc de faire référence à sa clé.

Syntaxe de base

Dans du code PHP, pour récupérer le texte "Bienvenue dans Copix 3 !" dans la langue courante, c'est tout simple :

```
$maChaine = _i18n ('default.welcome2Copix3');
```

ou avec l'ancienne syntaxe :

```
$maChaine = CopixI18N::get ('default.welcome2Copix3');
```

Notez qu'il est possible de faire référence à une clé située en dehors du module courant. Il faut alors simplement préciser le nom du module contenant cette clé. Par exemple :

```
$maChaine = _i18n ('default|default.welcome2Copix3');
```

Syntaxe avancée

Supposons un fichier *default2.properties* contenant :

```
default2.messageSurX = voici un message sur %s
default2.messageSurXetY = voici un message sur %s et %s
```

Vous remarquerez que les chaînes contiennent un code spécial "%s". Cela permet de les personnaliser en passant un tableau en paramètre supplémentaire à la syntaxe de base. Ce tableau contiendra les valeurs respectives des "%s" des chaînes.

Pour plus d'informations sur le formatage de chaîne, reportez-vous au manuel PHP : **http://fr.php.net/sprintf**

```
_i18n ('default2.messageSurX', 'valeur de X');
//ou _i18n ('default2.messageSurX', array ('valeur de X'));

_i18n ('default2.messageSurXetY', array ('valeur de X', 'valeur de Y'));
```

Il est même possible de forcer une langue ou un charset :

```
$langue = 'fr'; //ou $langue = 'fr_fr';
$maChaine = _i18n ('default2.messageSurXetY', array ('valeur de X', 'valeur de Y'),
$langue);
```

Utilisation dans un template PHP

Dans un template PHP, on utilise la syntaxe qui convient le mieux, par exemple :

```
<p>
    <?php echo _i18n ('default.welcome2Copix3'); ?>
</p>
```

Utilisation dans un template Smarty

Dans un template Smarty, il vous suffit d'utiliser le tag dédié :

```
<p>
    {i18n key='default.welcome2Copix3'}
</p>
```

Les autres fonctionnalités de CopixI18N

CopixI18N permet également de rendre bien d'autres services. Découvrons-les ensemble.

Langue courante

Pour obtenir le code correspondant à la langue courante du navigateur :

```
$langue = CopixI18N::getLang ();
```

Pour définir le code correspondant à la langue courante :

```
//on demande à utiliser le français
CopixI18N::setLang ('fr');
```

Charset courant

Pour obtenir le code correspondant au charset courant :

```
$charset = CopixI18N::getCharset ();
```

Pour définir le code correspondant au charset courant :

```
CopixI18N::setLang ('fr_fr');
```

Pays courant

Pour obtenir le code correspondant au pays courant :

```
$charset = CopixI18N::getCountry ();
```

Pour définir le code correspondant au pays courant :

```
//on demande à utiliser le pays france
CopixI18N::setCountry ('FR');
```

Formats de date

CopixI18N permet de générer le formatage à appliquer aux dates ou heures en fonction de la langue courante. La chaîne récupérée pourra être utilisée avec les fonctions PHP habituelles (date, time, ...)

Pour manipuler les dates, il est néanmoins recommander d'utiliser CopixDateTime qui emploie d'ailleurs en interne CopixI18N afin d'obtenir le bon format.

Formater uniquement la date

```
$separateur = '/';
$format = CopixI18N::getDateFormat ($separateur);

echo "Format: ". $format ." ";
echo "Date: ". date($format);
```

Pour la langue courante "fr" (français), ceci affichera :

```
Format: d/m/Y
Date: 26/06/2007
```

Formater la date et l'heure

```
$separateur = '/';
$format = CopixI18N::getDateTimeFormat ($separateur);

echo "Format: ". $format ."
";
echo "Date: ". date($format);
```

Pour la langue courante "fr" (français), ceci affichera :

```
Format: d/m/Y H:i:s
Date: 26/06/2007 18:40:00
```

Masque de formatage

```
$separateur = '/';
$mask = CopixI18N::getDateTimeMask ($separateur);
```

Voir aussi

- Les fichiers .properties

Les fichiers properties

Présentation

Les fichiers properties servent à stocker les messages que vous souhaitez internationaliser dans votre application par l'intermédiaire de CopixI18N.

Il est possible de spécifier un fichier (liste classée par ordre de priorité)

- une langue et un pays (exemple groupe_fr_FR.properties pour le français en France, groupe_en_US.properties pour l'anglais aux Etats Unis)
- une langue (exemple groupe_fr.properties pour le français, groupe_en.properties pour l'anglais)
- par défaut (groupe.properties)

Ainsi, lorsque vous demandez une clef donnée, CopixI18N regarde si cette dernière est déclarée dans un fichier langue_PAYS, dans le cas contraire utilise celle du fichier langue, pour enfin tenter de la retrouver dans le fichier par défaut.

Exemples

Il est bien sûr possible de faire autant de fichiers properties que vous le souhaitez.

En supposant que vous êtes dans la configuration langue = fr et country = fr.

- la clef default.message sera recherchée dans l'ordre dans les fichiers default_fr_FR.properties, default_fr.properties et enfin default.properties
- la clef admin.message.admin sera recherchée dans l'ordre dans les fichiers admin_fr_FR.properties, admin_fr.properties et enfin admin.properties

Si maintenant vous êtes dans la configuration langue = fr et country = BE, alors

- la clef default.message sera recherchée dans l'ordre dans les fichiers default_fr_BE.properties, default_fr.properties et enfin default.properties
- la clef admin.message.admin sera recherchée dans l'ordre dans les fichiers admin_fr_BE.properties, admin_fr.properties et enfin admin.properties

Si vous êtes dans la configuration langue = en et country = EN

- la clef default.message sera recherchée dans l'ordre dans les fichiers default_en_EN.properties, default_en.properties et enfin default.properties
- la clef admin.message.admin sera recherchée dans l'ordre dans les fichiers admin_en_EN.properties, admin_en.properties et enfin admin.properties

De la sorte, vous constatez qu'il est possible d'avoir des clefs communes à toutes les langues (fichier.properties), par langue (fichier_langue.properties) ou spécifiques par langue/pays (fichier_langue_PAYS.properties).

CopixSession

Présentation

CopixSession est une classe vous permettant de manipuler des données dans la session avec quelques fonctionnalités supplémentaires, telles :

- La gestion de "namespace"
- La gestion automatique des objets avec CopixSessionObject (transparent pour vous)

Exemple d'utilisation

```
//Sauvegarde un objet record en session.
CopixSession::set ('le|chemin|de|stockage', _record ('MonDAO'));

//récupération de l'objet
$record = CopixSession::get ('le|chemin|de|stockage');
```

Lecture / Ecriture

La méthode set permet de définir des éléments dans la session :

```
//types simples
CopixSession::set ('variable', 1);
CopixSession::set ('une|variable', 'valeur');
CopixSession::set ('une|autre|variable', array (1, 2, 3));

//Gestion automatique des objets Copix
CopixSession::set ('unDAO', _dao ('monDAO'));//les dao
CopixSession::set ('unRecord', _record ('monDAO'));//les records
CopixSession::set ('classe', _class ('monModule|classId'));//les objets récupérés avec
CopixClassesFactory
```

La méthode get permet de lire les éléments de la session

```
CopixSession::get ('variable');
CopixSession::get ('une|variable');
CopixSession::get ('une|autre|variable');
```

Si l'élément demandé n'existe pas, null est retourné.

Raccourcis

Il existe un raccourcis pour la méthode CopixSession::get et CopixSession::set nommé _session.

Si vous donnez deux arguments à _session, alors elle se comporte comme CopixSession::set ($argument1, $argument2); Si vous ne donnez qu'un seul argument à _session, alors elle se comportera comme CopixSession::get ($argumentUnique);

```
_session ('copixtest|key', 'value');//définition de la valeur de copixtest|key à value
$value = _session ('copixtest|key');//$value vaut 'value'
_session ('copixtest|key', null);//réinitialisation de copixtest|key en session.
```

Note : Cette fonction raccourcis existe depuis Copix 3.0.3+

CopixSessionObject

Présentation

PHP, en standard, n'est pas capable de lire des objets depuis la session s'ils ne sont pas connus avant le démarrage de cette dernière.

Ainsi, s'il existe dans la session un objet "foo" qui n'a pas été déclaré avant l'appel à session_start, alors PHP ne sait pas relire l'objet en question.

En PHP5, nous disposons d'un système d'autoload pour permettre à PHP de charger automatiquement certaines classes, simplement en réalisant une méthode capable de charger la définition d'une classe à partir de son nom. Cette méthode peut toutefois sembler lourde lorsque l'objet que l'on souhaite mettre en session n'est pas habituel.

Ainsi, Copix propose un objet CopixSessionObject qui va vous permettre de placer des objets inconnus à PHP en session, tout en lui donnant un système pour relire cet objet de façon transparente.

En standard, vous pouvez passer à CopixSessionObject n'importe quel DAO, Record ou classe Copix standard, ce dernier se débrouillera tout seul (dans 80% des cas) pour savoir comment inclure sa définition.

Utilisation avec des DAO

Avec des DAO

Voilà comment placer un objet **DAO** en session.

```
$dao = new CopixSessionObject (_ioDAO ('monDAO'));
$dao->findAll ();//toutes les méthodes de l'objet sont appelables directement via
            ✂ CopixSessionObject.
$_SESSION['QUELQUE_PART'] = $dao;
```

Tous les appels (méthodes et propriétés) que vous pourrez faire à l'objet CopixSessionObject seront automatiquement transmis au "vrai" DAO, rendant l'utilisation de l'objet de session complètement transparent.

Vous pouvez également, si vous le souhaitez, spécifier explicitement le sélecteur de DAO que vous avez utilisé, par exemple :

```
$dao = new CopixSessionObject (_ioDAO ('monDAO'), 'monModule|monDAO');//on spécifie
            ✂ comment on a obtenu le DAO
$_SESSION['QUELQUE_PART'] = $dao;
```

Avec des records

Tous les éléments valables pour les DAO le sont également pour les record.

exemple

```
$_SESSION['QUELQUE_PART'] = new CopixSessionObject (_record ('monDAO'));
$_SESSION['QUELQUE_PART'] = new CopixSessionObject (_record ('monModule|monDAO'),
            ✂ 'monModule|monDAO');
```

Avec des classes métiers

Par défaut, lorsque vous passez un objet (qui n'est pas un DAO ou un record) à CopixSessionObject, celui ci considère que c'est un objet que vous avez instancié via **CopixClassesFactory** et se servira de cette dernière fabrique pour le récupérer.

```
$_SESSION['QUELQUE_PART'] = new CopixSessionObject (_class ('maClasse'));
//Ou en spécifiant explicitement la façon de le charger via le sélecteur
$_SESSION['QUELQUE_PART'] = new CopixSessionObject (_class ('monModule|maClasse'),
            ✂ 'monModule|maClasse');
```

Note : Encore une fois, toutes les méthodes et propriétés de l'objet sont manipulables directement depuis l'objet session.

Avec des objets autres

En dernier lieu, si vous souhaitez placer un objet PHP en session qui n'est ni un DAO, ni un record, ni une classe instanciée via CopixClassesFactory, vous pouvez simplement spécifier le chemin de la définition de l'objet.

exemple :

```
require_once ('bibliotheque_tierce.php');
$_SESSION['QUELQUE_PART'] = new CopixSessionObject (new ObjetTiers (),
            ✂ 'bibliotheque_tierce.php');
```

Avec des objets gérés par l'autoload

Si votre objet est géré par l'autoload, alors vous pouvez spécifier à CopixSessionObject de ne pas tenter des manipulations partiulières pour le prendre en charge.

```
new CopixSessionObject (new ObjetAutoloaded (), CopixSessionObject::AUTOLOADED);
```

Note : Pour toutes les classes Copix standard (dont le nom commence par 'Copix'), il est inutile de spécifier explicitement cette information.

Méthode supplémentaire

CopixSessionObject dispose d'une seule méthode propre qui est getSessionObject ().

Cette méthode retourne l'objet encapsulé par CopixSessionObject.

exemple

```
$objetSession = new CopixSessionObject ($dao = _dao ('daoAuto'));
$dao === $objetSession->getSessionObject ();//OUI
$dao === $objetSession; //NON
```

Voir aussi

- CopixSession

CopixUploadedFile

Présentation

La classe CopixUploadedFile permet de gérer les opérations courantes lorsque vous souhaitez permettre à l'internaute d'uploader des fichiers vers votre application.

Exemple avec le formulaire suivant :

```
<form action="{copixurl dest="upload"}" enctype="multipart/form-data" method="POST">
 Fichier : <input type="file" name="fichier">
</form>
```

ATTENTION Pour pouvoir uploader un fichier, il est nécessaire d'avoir les paramètres enctype="multipart/form-data" method="POST" dans la déclaration de la balise form.

et le code de l'action

```
//dans l'actiongroup
public function processUpload (){
   //on va copier le fichier quelque part
   //ou alors en utilisant CopixUploadedFile directement
   if (($fichier = CopixUploadedFile::get ('fichier')) !== false){
      $fichier->move ($cheminDestination);
   }
   //... on effectue la suite des traitements
}
```

A savoir qu'il est également possible d'utiliser **CopixRequest** pour ce faire

```
public function processUpload (){
   //on va copier le fichier quelque part
   //en utilisant CopixRequest::getFile
   if (CopixRequest::getFile ('fichier', $cheminDestination) !== false){
      //la copie s'est bien passée
   }
   //... on effectue la suite des traitements
}
```

Méthodes

getTempPath

Cette méthode retourne le chemin temporaire dans lequel le fichier à été placé en attendant d'être déplacé.

Exemple

```
if ((($fichier = CopixUploadedFile::get ('fichier')) !== false){
   $fichier->getTempPath ();
}
```

getType

Cette méthode retourne le type MIME du fichier tel que le navigateur du client nous l'a présenté.

Cette information n'est pas vérifiée par PHP ni par Copix. Vous ne pouvez donc pas vous y fier de façon certaine.

Exemple

```
if ((($fichier = CopixUploadedFile::get ('fichier')) !== false){
   $fichier->getType ();
}
```

getName

Cette méthode retourne le nom du fichier tel que le navigateur du client nous l'a présenté.

Exemple

```
if ((($fichier = CopixUploadedFile::get ('fichier')) !== false){
   $fichier->getName ();
}
```

getSize

Cette méthode retourne la taille du fichier en octet.

Exemple

```
if ((($fichier = CopixUploadedFile::get ('fichier')) !== false){
   $fichier->getSize ();
}
```

getError

Cette méthode statique permet de connaître l'erreur de téléchargement qui est survenue sur un fichier.

Exemple

```
if (($fichier = CopixUploadedFile::get ('fichier')) === false){
   //récupère l'erreur de téléchargement
   CopixUploadedFile::getError ('fichier');
}
```

move ($pPath, $pFileName = null)

Cette méthode déplace le fichier téléchargé vers l'emplacement $pPath.

Il est possible de spécifier un nom de fichier différent que celui proposé par le navigateur en donnant le paramètre $pFileName.

Si $pFileName n'est pas donné, le nom par défaut (getName) sera utilisé.

Exemple

```
if (($fichier = CopixUploadedFile::get ('fichier')) !== false){
   $fichier->move ($cheminDestination, $nomDeFichierSpecifique);
}
```

Voir aussi

- CopixRequest méthode getFile

Trigramme module|group|action

Présentation

Lorsque vous utilisez la classe **CopixUrl** / la balise smarty **CopixUrl** pour générer une URL ou que vous décrivez des **actions**, vous respectez le trigramme Copix composé des éléments "module|group|action".

Ce trigramme indique à Copix quelle est l'action que vous souhaitez lancer, dans quel ActionGroup elle est implémentée et enfin dans quel module se situe le tout.

Ne pas spécifier l'ensemble des éléments

Vous n'êtes pas tenu de toujours spécifier les trois éléments, ainsi :

- Si vous spécifiez deux éléments, ce seront "group|action". Le module considéré sera le module en cours d'exécution.
- Si vous spécifiez un seul élément, ce sera l'action du group default. Ainsi, "monAction" équivaut à "default|monAction".
- Si vous ne spécifiez aucun élément, alors Copix retournera l'url de base (sans le nom du script)

Exemples :

```
//appelé depuis http://www.copix.org/chemin/du/site/index.php
CopixUrl::get ();//retourne http://www.copix.org/chemin/du/site/

//Appelé depuis http://www.copix.org/index.php
CopixUrl::get (); //retourne http://www.copix.org/
```

En résumé

- 3 éléments : "module|group|action"
- 2 éléments : "group|action" Copix ira chercher cette action dans le module courant
- 1 élément : "action" Copix ira chercher cette action dans le group "default" du module courant.
- Vide : Url de base de votre site

Cas particuliers

Si vous ne spécifiez pas un élément alors que vous spécifiez le séparateur, Copix considère la valeur de l'élément manquant comme étant "default"

- "||"" équivaut à "default|default|default"
- "|" équivaut à "module courant|default|default"
- "|groupe|action" équivaut à "default|groupe|action"
- "||action" équivaut à "default|default|action"
- "module||action" équivaut à "module|default|action"
- "module|groupe|" équivaut à "module|groupe|default"
- "#" Corresponds à l'url actuelle

```
//Appelé depuis http://www.copix.org/index.php?module=test
CopixUrl::get ('#'); //retourne "http://www.copix.org/index.php?module=test"
```

Webservices_et_PHP

Généralités : Les Web Services et PHP

SOAP, WSDL, WTF?

SOAP veut dire Simple Object Access Protocol, c'est un protocole permettant d'appeler des procédures distantes. Il se base sur XML.

WSDL : Web Services Description Language. Il s'agit d'un ficher XML regroupant la description des élements permettant d'accéder au WebServices.

Plus d'infos sont disponible sur Wikipedia : **Article SOAP**, **Article WSDL**

Les différentes implémentation de SOAP en PHP

Il est possible de mettre en place et d'accéder à des Web Services SOAP avec différentes solutions.

La classe la plus populaire, en PHP4, fut **NuSOAP**. Elle est toujours fonctionnelle et à pour avantage d'effectuer la génération automatique du fichier WSDL, Le développement n'est par contre plus actif.

Autre classe intéressante, Pear::Soap, qui permet de créer et d'utiliser les Web Services SOAP en PHP4 et PHP5. Le développement est toujours actif, mais la classe est toujours en version béta. Pour plus d'infos je vous invite à consulter ce **tutoriel**.

PHP intègre désormais une **extension SOAP** permettant d'écrire des clients et serveur SOAP. C'est cette dernière extension qui a été choisi pour être utilisé dans le module WSServer de **Copix**.

Extension PHP SOAP

Apparu avec PHP5, l'extension SOAP permet d'écrire des serveurs et clients SOAP. Elle supporte une partie des spécifications SOAP 1.1, SOAP 1.2 et WSDL 1.1.

Les classes prédéfinies sont les suivantes : * SoapClient * SoapServeur * SoapFault * SoapHeader * SoapParam * SoapVar

Tous les détails se trouve sur le **manuel de PHP**.

Exemple

Exemple d'un client :

```
Architecture
<?php
$client = new SoapClient("some.wsdl");

$arFunctions = $client->__getFunctions();

foreach ($arFunctions as $function) {
    echo $function.'<br/>'."
";
}

?>
```

Exemple d'un serveur Soap

```php
<?php
$server = new SoapServer ("some.wsdl");

$server->setClass("Foo");

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $server->handle();
} else {
    echo "Ce serveur SOAP peut gérer les fonctions suivantes : ";
    $functions = $server->getFunctions();
    foreach($functions as $func) {
        echo $func . "
";
    }
}

class Foo {
    function Add($x, $y);
}
?>
```

Problèmes

Pour l'instant l'extension Soap ne permet pas de générer un fichier WSDL. Pour celà le module de Copix utilise **WSDL_Gen.php**.

Themes graphiques

NOTE: attention document en cours de rédaction.

Présentation

L'intérêt des thèmes est de pouvoir changer à la volée l'aspect graphique du site sans avoir à modifier une ligne dans son contenu.

Création

Un thème est en général consitué de ressources: les images, les javascripts, les feuilles des styles, ... et d'une partie dédiée à la présentation (le code HTML). Pour concevoir des thèmes, il faut, comme souvent dans l'esprit Copix séparer les problèmes.

Partie ressources

Pour ce qui est des ressources, elles se situent dans le répertoire *www/themes* de votre site. Dans ce répertoire, on trouvera, par exemple, un répertoire *img* pour les images, un répertoire *styles* pour les css, ... tout est libre et optionel.

Vous trouverez toutes les infomations nécessaires dans les pages **CopixResource** et **Personnalisez les ressources**.

Partie affichage et enregistrement

La partie dédiée à l'affichage est aussi une partie qui, comme pour les modules, permet l'enregistrement des thèmes dans Copix. Cette partie se trouve dans le répertoire *project/themes/<nom_du_theme>*.

Dans ce repertoire, il faudra ajouter un fichier *theme.xml* qui est la fiche signalétique du thème, une image du thème ainsi que des repertoires contenant le templace d'affichage.

theme.xml

Ce fichier contient les champs suivants:

- name: le nom du thème (par exemple "mon 1er thème")
- author: l'auteur du thème
- website: le site web de l'auteur du thème
- description: une description du thème par l'auteur

l'image du thème

Copix utilise aussi dans son système de choix de thème une image qui représente le thème. C'est optionnel mais cela permet d'avoir un aperçu rapide du thème. Cette image doit se trouver elle aussi à la racine du répertoire du thème.

les répertoires de templates

Les templates du thèmes permettent de se substituer aux templates d'un module. Copix utilise donc le nommage des templates du thème pour savoir quel template remplacer. Par exemple, considerons un module *exemple* qui contient dans son répertoire *templates* un template nommé *mon_template.php*. Il suffit de créer un nouveau template *mon_template.php* dans le répertoire *project/themes/nom_de_mon_theme/exemple/mon_template.php*.

Pour savoir si un template est surchargé ou pas, Copix va regarder dans l'ordre suivant:

- Le template du thème courant s'il existe
- Le template du thème "default" s'il existe
- En dernier lieu celui livré avec le module

Plus d'informations sur les templates dans la page **template principal**.

Normes_de_developpement

Général

Balises PHP

N'utiliser que les balises PHP complètes, les balises courtes pouvant rentrer en conflit avec d'autres langages.

```
<?php
$phpCode;
?>
```

Indentation

Note : la norme initiale précisait 3 espaces, à cause d'une vieille habitude de Gérald. Elle avait été changée à 4 espaces récemment. Toutefois, à des fins d'harmonisation avec les pratiques courantes, nous recommandons aujourd'hui l'utilisation de tabulations.

L'indentation initiale d'une ligne est d'une tabulation, par exemple (les tabulations sont représentées par "»") :

```
<?php
if ($test){
»       echo $test . ' is ok';
}else{
»       echo $test . ' is KO';
}
?>
```

Cependant, si à des fins de présentation, vous voulez réaliser des alignements, utilisez uniquement des espaces pour l'alignement. Par exemple (les tabulations sont représentées par "»" et les espaces par "°"):

```
<?php
$monTableAssoc = array(
»       "uneClefCourte"°°°°°=> "uneValeur",
»       "uneClefPlusLongue"°=> "uneAutreValeur"
);
?>
```

Ainsi votre alignement sera respecté quelque soit l'indentation, tout en laissant à chacun la possibilité de choisir la largeur d'indentation qu'il préfère (et aussi de "désindenter" avec une simple pression sur Suppr).

Configurer Eclipse pour cette norme

Si vous utilisez l'IDE Eclipse PDT, il est possible de configurer celui ci pour respecter facilement cette norme. 1) Cliquez sur *Windows/Preferences*. 2) Allez dans la section *PHP/Formatter*. Sélectionnez la valeur "Tabs" pour l'option "Tab policy". 2) Allez dans la section *General/Editors/Text Editors*. Assurez-vous que la case "Insert spaces for tabs" est décochée. Vous pouvez saisir la valeur qui vous convient pour "Displayed Tab Width", cela n'affecte que l'affichage.

Notez que le raccourci Ctrl+Shift+F formatera votre code correctement (pour les formats reconnus).

Encodage

Le jeu de caractères à utiliser pour tous les fichiers sources est l'UTF-8.

Astuce : Si vous utilisez l'IDE Eclipse PDT, il est possible de configurer le projet pour utiliser l'UTF-8 par défaut pour tous les fichiers. Cliquez sur *Project/Properties* puis sur l'option "Resource". Dans le cadre "Text file encoding", sélectionner l'option "Other" et utilisez le menu déroulant pour sélectionner "UTF-8".

Règles de nommage

Classes

En un coup d'œil

```php
<?php
/**
* @package   Nom
* @author   Croes Gérald
* @copyright Infos de copyrights
* @link      si lien il y a
* @licence  lien sur la licence choisie
*/

/**
* Objectif de la classe
*
* Une description plus longue de la classe si nécessaire
*/
class MyClass {
    /**
    * Constantes en majuscules
    */
    const UNE_CONSTANTE = 1;

    /**
    * foo sert à ça
    * @var string
    */
    public $foo;

    /**
    * otherFoo sert à ça
    * @var int
    */
    public $otherFoo;
```

```
    /**
    * C'est une variable protégée qui sert à ça
    * @var ressource
    */
    protected $_protectedFoo;

    /**
    * C'est une variable privée qui sert à ça
    * @var ressource
    */
    private $_privateFoo;

    /**
    * Méthode public qui fait quelquechose
    *
    * Alors en fait c'est assez compliqué ce qu'elle fait, voici
    * donc une description longue de son rôle
    */
    public function fooFunction (){
        doSomething ();
    }

    /**
    * Méthode public qui fait quelquechose
    * @param string $pFooParam pFooParam sert à ça
    * @param boolean $pIsFooParam2 indique quelquechose (par défaut true)
    * @return boolean si on peut continuer (true) ou pas (false)
    */
    public function otherFooFunction ($pFooParam, $pIsFooParam2 = true){
        $this->doSomething ();
        return canDoSomethingElse ();
    }

    /**
    * Méthode qui fait quelquechose
    */
    private function _privateFunction (){
        $this->doSomething ();
        doSomethingElse ();
    }
}
?>
```

Nom

Généralités

Tous les mots qui composent le nom d'une classe commencent par une majuscule. Les fichiers portent l'extension ".class.php" et sont en minuscule.

Services

Les classes de services se nomment ServicesRole et sont dans un fichier servicesrole.class.php. On les instancie via la **CopixClassesFactory** avec la syntaxe

```
$service = CopixClassesFactory::create ('module|ServicesRole');
//ou
$service = _class ('module|ServicesRole');
```

Cas particulier des acronymes

Les acronymes sont écrits en majuscules.

Exemple

```
class HTMLHeader {}
class XMLViewer {}
class HTTPResponse {}
```

Méthodes

Le premier mot du nom d'une méthode commence par une minuscule. Les mots suivants commencent par une majuscule.

```
class HTMLViewer {
   function doSomething (){}
   function getSomething (){}
   function getSomethingElseForTheSakeOfTheExample (){}
}
```

Les acronymes, en fonction de leur position dans le nom de la fonction, sont écrits tout en minuscule ou tout en majuscule.

```
class XMLParser {
   function htmlView (){}
   function xmlCheck (){}
   function getXMLNode (){}
   function getHTMLView (){}
   function setValueInXML (){}
}
```

Les méthodes protégées et privées commencent par un underscore "_"

```
class Foo {
   private function _privateMethod (){}
   protected function _protectedMethod (){}
   public function publicMethod (){}
}
```

Les paramètres des méthodes suivent les mêmes règles que les attributs de classe, à l'exception qu'ils commencent par la lettre "p".

```
class MaClass {
   public function maMethode ($pParametre1, $pParametre2, $pParametreFacultatif = true){
   }
}
```

Attributs

Les attributs de classe suivent les mêmes règles que les noms de méthodes (premier mot en minuscule, les mots suivants commencent par une majuscule).

```
class HTMLStuff {
   private $_privateData;
   protected $_xmlValue;
   protected $_formatHTML;
   public $xmlCheck;
   public $isHTML;
}
```

Scripts de bases de données

Scripts MySQL

Lors de la création de tables MySQL ne jamais préciser le moteur à utiliser (option `ENGINE`) afin d'utiliser le moteur par défaut configuré au niveau du serveur/de la base de données.

En revanche, toujours préciser l'encodage à utiliser pour les tables (option `CHARACTER SET`). Sauf besoin exceptionnel, l'encodage à utiliser est l'UTF-8.

Il convient également de mettre tous les noms de table et de champs entre backquotes (`), afin d'éviter les conflits avec des mots réservés.

Exemple:

```
CREATE TABLE `copixconfig` (
  `id_ccfg` varchar(255) NOT NULL default '',
  `module_ccfg` varchar(255) NOT NULL default '',
  `value_ccfg` varchar(255) default NULL,
  PRIMARY KEY  (`id_ccfg`)
) CHARACTER SET utf8;
```

Shortcut

Présentation

Les fonctions raccourcis permettent d'utiliser les fonctions courantes de copix à partir de quelques caractères.

DAO

```
_dao    ($name);    //pour CopixDAOFactory::create ($name);
_ioDAO  ($name);    //pour CopixDAOFactory::getInstanceOf ($name);
_daoInclude ($name); //Pour CopixDAOFactory::fileInclude ($name);
_record ($name);    //pour CopixDAOFactory::createRecord ($name);
_daoSP ($kind);     //pour CopixDAOFactory::createSearchParams ($kind);
```

En savoir plus sur les **DAO**.

CopixDB

```
_doQuery ($query, $params = array (), $ct = null);
// Pour CopixDB::getConnection ($ct)->doQuery ($query, $params);
```

En savoir plus sur **CopixDB**.

Classes

```
_class    ($name); //Pour CopixClassesFactory::create ($name);
_ioClass  ($name); //Pour CopixClassesFactory::getInstanceOf ($name);
_classInclude ($name);//Pour CopixClassesFactory::fileInclude ($name);
```

Evènements

```
_notify ($pEvent, $pParams = array ());
//CopixEventNotifier::notify ($pEvent, $pParams=array ());
```

En savoir plus sur les **évènements**.

Requêtes, URL & ressources

```
_url  ($name, $params = array ());//Pour CopixURL::get ($name, $params = array ());
_resource ($name); //Pour CopixUrl::getResource ();
_request ($pVarName, $pDefaultValue = null, $pDefaultIfEmpty = true);
//pour CopixRequest::get ()
```

En savoir plus sur .CopixUrl.

En savoir plus sur .CopixRequest.

Les logs

```
_log ($pChaine, $pType = "default", $pLevel = CopixLog::INFORMATION, $arExtra =
            array ()); //Alias à CopixLog::log ()
```

L'internationalisation

```
_i18n ($key); //Pour CopixI18N::get ($name);
```

En savoir plus sur .CopixI18N.

Les templates

```
_tag ($name, $params); //pour CopixTpl::tag ($name, $params);
_eTag ($name, $params); //pour echo CopixTpl::tag ($name, $params);
```

En savoir plus sur les .tags.

L'appel de services

```
_service ($name, $params, $transactionContext);
//pour CopixServices::process ($name, $params, $transactionContext)
```

En savoir plus sur les .services.

Retours des actions

```
_arNone ()
_arPPO ($ppo, $template)
_arDirectPPO ($ppo, $template)
_arRedirect ($url)
_arFile ($file, $options)
_arContent ($content, $options)
_arDisplay ($tpl)
```

En savoir plus sur les actions et les Actiongroup.

Utilisateurs

```
_currentUser (); //équivalent à CopixAuth::getCurrentUser ()
```

Sessions

Depuis Copix 3.0.3+

Equivalent à CopixSession::get ou CopixSession::set en fonction du nombre d'arguments.

```
_session ('copixtest|key', 'value');//définition de la valeur de copixtest|key à value
$value = _session ('copixtest|key');//$value vaut 'value'
_session ('copixtest|key', null);//réinitialisation de copixtest|key en session.
```

CopixDAO

Fichier XML

Introduction

Rappels & Introduction du fichier XML

Avec CopixDAO, vous pouvez donc développer un DAO sans écrire une seule ligne de code PHP.

Vous pouvez choisir d'indiquer dans la factory le nom de la table (auquel cas Copix pourra générer automatiquement un DAO pour vous) ou choisir de spécifier un sélecteur de fichier XML, ce que nous allons expliquer ici.

Ce fichier XML doit être situé dans le répertoire des ressources (resources) et porter le nom nom_dao.dao.xml.

Ce fichier XML va vous permettre de déclarer le "mapping" entre l'objet DAO et une ou plusieurs tables en base de données, donc de dire que telle propriété de l'objet correspond à tel champ de telle table.

Un fichier de définition de DAO doit être enregistré dans le répertoire resources d'un module ou du projet. Il doit avoir comme nom : xxxxx.dao.xml, xxxxx étant le nom du DAO, que l'on indiquera en paramètre à certaines fonctions.

Le fichier XML de définition de DAO contient bien sûr une balise racine qui a pour nom daodefinition. Voici un début de ce fichier :

```
<?xml version="1.0" encoding="UTF-8" ?>
<daodefinition>
...
</daodefinition>
```

La balise daodefinition encadre trois sections différentes :

1. La déclaration de la source de données (connection, tables..)
2. La déclaration des propriétés
3. La déclaration (facultative) de méthodes supplémentaires de sélection

Dans quels cas ?

Dans quels cas pouvons-nous souhaiter utiliser un fichier XML plutôt qu'un DAO automatique (à partir du nom de la table) ?

- Déclaration de méthodes supplémentaires (section methods)
- Mapping limité par rapport aux champs
- Déclaration de contraintes supplémentaires sur les champs
- Fournir des informations supplémentaires sur les champs (leur "vrai" nom par exemple)
- Réaliser des jointures

Fichier de définition XML

Section déclaration de la source de donnée

Cette section est délimitée par la balise datasource. Elle contient une balise facultative "connection", indiquant la connection CopixDB à utiliser, et la déclaration des tables, comme ceci :

```
<?xml version="1.0" encoding="UTF-8" ?>
<daodefinition>
   <datasource>
      <connection name="Select" />
      <tables>
         <table name="pers" tablename="personne_per" primary="yes" />
         <table name="ville" tablename="ville_vi" join="left" />
      </tables>
   </datasource>
</daodefinition>
```

Nous avons ici indiqué que le DAO doit utiliser la connexion ayant pour nom Select. Nous pouvons omettre la balise connection, et dans ce cas, le DAO utilisera la connexion déclaré par défaut dans **CopixDB**.

Nous indiquons ensuite que le DAO repose sur 2 tables, personne_per et ville_vi, qui sont liées par une jointure externe (left), la table personne_per étant la table principale (primary="yes")

À chacune des déclarations de table nous donnons un nom arbitraire, pers et ville qui seront utilisés dans la déclaration des propriétés. Ces noms serviront également d'alias dans les requêtes SQL générées.

L'attribut name est obligatoire. L'attribut tablename est facultatif, et vaut par défaut la valeur de l'attribut name. Il doit y avoir une et une seule déclaration de table qui contienne l'attribut primary avec la valeur yes (ou 1, ou true). L'attribut join est facultatif et vaut par défaut 'inner', les autres valeurs possibles étant 'left' et 'right'.

Section déclaration des propriétés

Cette section est délimitée par la balise properties, et contient un ensemble de balise property :

```
<?xml version="1.0"   encoding="UTF-8" ?>
<daodefinition>
   <datasource>...</datasource>
   <properties>
      <property ... />
      <property ... />
      <property ... />
      ...
   </properties>
</daodefinition>
```

Cette section permet de déclarer les propriétés du DAO, ainsi que le mapping avec les tables. Les attributs possibles pour la balise property sont les suivants.

- name : nom de la propriété (obligatoire)
- fieldname : nom du champs correspondant (Si absent, vaut par défaut la valeur de name)
- table : nom (alias) de la table où se trouve le champs. Si cet attribut est omis, la table par défaut est celle déclarée comme primaire.
- required : booléen indiquant si la propriété doit être non null.
- pk : booléen indiquant si le champ correspondant à la propriété est une clé primaire
- type : indique le type de donnée. Les valeurs possibles sont : integer (ou int), autoincrement, double (ou float), numeric (long entier) ou string (valeur par défaut).
- fktable, fkfieldname : si ces attributs sont définis et non vides, ils indiquent que le champ est une clé étrangère. Ces deux attributs contiennent donc respectivement le nom de la table étrangère (valeur de l'attribut name de la balise <table /> correspondante) et celui du champ (le vrai nom du champ) associé dans cette table. Les attributs pk et fktable/fkfieldname sont bien entendu mutuellement exclusif.
- sequence : nom de la séquence à utiliser pour récupérer l'id, dans le cas d'une propriété de type autoincrement (et nécessaire pour certains types de base).

Section déclaration des méthodes

Comme on l'a vu, on peut récupérer des ensembles d'enregistrements soit avec la méthode findAll, soit avec la méthode findBy. Cependant l'inconvénient de la première c'est qu'on récupère tout, sans trie, et que la seconde, elle peut être lente.

On peut déclarer dans le fichier de définition de l'objet DAO, des méthodes supplémentaires pour récupérer des ensembles d'enregistrements. L'avantage est que l'on peut indiquer des critères de sélection (ceux-ci pouvant être dynamiques comme avec findBy), et que c'est aussi rapide que findAll puisque la requête est générée et en dur dans l'objet (Rappelez vous que les classes générées à partir des fichiers de définition sont mis en cache..). On réservera donc l'utilisation de findBy pour des cas où on ne sait pas à l'avance le nombre de critères et/ou leur nature.

Pour déclarer ces nouvelles méthodes, il faut ajouter une section supplémentaire dans le fichier de définition, délimité par la balise <methods>, et contenant un ensemble de balises <method> déclarant chaque méthode.

```
<?xml version="1.0" ?>
<daodefinition version="1">
...
<methods>
   <method ...>....</method>
   <method ...>....</method>
   ...
</methods>
</daodefinition>
```

La balise method possède deux attributs, name et type, qui indiquent respectivement le nom de la méthode et son type. Pour l'instant il y a deux types : select pour renvoyer toute la sélection, ou selectfirst pour ne renvoyer que le premier élément de la sélection.

Ensuite cette balise peut contenir 4 balises différentes, qui indique respectivement les paramètres de la méthode, les conditions de sélection (critères), l'ordre de sélection et les limites :

```
<?xml version="1.0" ?>
<daodefinition version="1">
...
<methods>
   <method name="findByDateAuthor" type="select">

      <parameters>
         <parameter name="" />
         ...
      </parameters>

      <conditions>
         <condition property="" operator="" value="" />
         <condition property="" operator="" value="" />
         ...
      </conditions>

      <order>
         <orderitem property="" way="" />
         ...
      </order>
      <limit offset="" count="" />

   </method>
</methods>
</daodefinition>
```

Chaque section, parameters, conditions, order, et la balise limit sont facultatives (mais en mettre aucune n'a aucun sens bien sûr puisque cela revient à faire l'équivalent de findAll).

La section parameters

Elle définit les paramètres de la méthodes. Il s'agit simplement d'utiliser une balise parameter pour chaque paramètre, avec un attribut name indiquant le nom du paramètre.

La section conditions

Indique les critères de sélection (correspond à la clause WHERE en sql). La balise conditions permet de regrouper un ensemble de critères (balise condition, sans s). Un attribut, logic permet d'indiquer le "lien" entre les critères : OR ou AND (défaut).

On peut bien sûr déclarer des groupes de critères dans des groupes de critères.

La balise condition quant à elle a trois attributs :

- property : le nom de la propriété qui servira de critère.
- value : valeur qui sera comparée avec la propriété. Cela peut être une valeur prédéfinie, ou alors un des paramètres de la méthode. Dans ce cas, il faut mettre le nom du paramètre précédé du caractère $;
- operator : opérateur qui sert de comparaison. Comme on est dans un fichier xml, pour les opérateurs < et >, il faut écrire < et >

La section order

Permet de spécifier dans quel ordre on va récupérer les données. Pour cela, cette section doit contenir une ou plusieurs balises orderitem. Celles-ci ont comme attributs :

- property : le nom de la propriété sur laquelle le tri s'effectuera ;
- way : Le sens du tri (asc ou desc ;

Vous aurez compris bien sûr que cela correspond à la clause ORDER BY en SQL.

La balise limit

Permet d'indiquer de ramener qu'un nombre limité d'enregistrements.

- offset : numéro de l'enregistrement dans la sélection à partir duquel on va récupérer les enregistrements.
- count : le nombre d'enregistrements que l'on va récupérer ;

Les valeurs peuvent être des nombres, ou alors des noms de paramètres définis dans la méthode (précédé d'un $)

Il ne peut y avoir qu'une balise limit par méthode.

Exemple

```
<?xml version="1.0" ?>
<daodefinition>
...
<methods>
   <method name="findByDateAuthor" type="select">

      <parameters>
         <parameter name="paramdate" />
         <parameter name="myoffset" />
      </parameters>

      <conditions>
         <condition property="date" operator="<" value="$paramdate" />
         <condition property="author" operator="=" value="Laurent" />
         <conditions logic="or">
            <condition property="valide" operator="=" value="1" />
            <condition property="categorie" operator="=" value="automobile" />
         </conditions>
      </conditions>

      <order>
         <orderitem property="date" way="desc" />
      </order>
      <limit offset="$myoffset" count="15" />
   </method>
</methods>
</daodefinition>
```

Dans cette exemple, cela va récupérer les enregistrements dont la date est supérieure à la valeur du paramètre $paramdate, ET dont la propriété author est égale à Laurent, ET dont la condition qui suit est respectée : que la propriété valide soit égale à 1 OU que la propriété catégorie soit égale à automobile (ça n'a pas trop de sens mais c'est juste pour l'exemple ;-)).

On pourra récupérer 15 enregistrements, à partir du nième indiqué dans le paramètre myoffset, et ils seront classés selon la propriété date, par ordre décroissant.

Pour appeler cette méthode en php, il suffira de faire :

```
// on récupère un objet DAO basé sur un fichier de définition news.daodefinition.xml
// ce fichier map une simple table de news

$DAOnews = CopixDAOFactory::create ('News');

// Récupération des news à partir du huitième enregistrement.
$liste_news =  $DAOnews->findByDateAuthor ('2004-05-24', 8);
```

Ajouter des méthodes de suppression en xml

À la manière des méthodes de sélection en xml comme on vient de le voir, il est possible de créer des méthodes de suppression : on va pouvoir spécifier les critères de suppression, sans écrire une ligne de code SQL/PHP.

Pour cela, il suffit d'ajouter une balise method, avec dans l'attribut type la valeur delete. Tout comme les méthodes de sélection, vous pouvez définir des paramètres de méthodes, et une section conditions. Par contre, vous ne pouvez pas spécifier une section order et limit, cela n'ayant pas de sens.

```
<?xml version="1.0" ?>
<daodefinition>
...
<methods>
   <method name="deleteByDate" type="delete">

      <parameters>
         <parameter name="paramdate" />
      </parameters>

      <conditions>
         <condition property="date" operator="<" value="$paramdate" />
      </conditions>
   </method>
</methods>
</daodefinition>
</xml>
```

Ici on a défini une méthode deleteByDate, qui accepte un paramètre paramdate. Elle supprimera les enregistrements de la table principale de l'objet, dont la propriété date est inférieure au paramètre donné.

Si vous voulez faire des suppressions supplémentaires, dans d'autres tables pour maintenir une cohérence dans vos données, il faudra faire autrement, en créant vos propres méthodes comme nous le verrons ensuite.

Ajouter des méthodes de mise à jour en xml

Le dernier type de méthode que l'on peut déclarer en XML, ce sont les méthodes de mise à jour, c'est à dire qui vont faire des UPDATE en SQL. La différence avec la méthode update générée automatiquement, c'est qu'avec ce type de méthode, vous pouvez mettre à jour seulement un ou quelques champs, et non pas la totalité.

Il faut donc indiquer comme type de methode, update, et au moins une ou plusieurs balises value dans un élément values, comme ceci :

```
<?xml version="1.0" ?>
<daodefinition>
...
<methods>
   <method name="updateDate" type="update">
      <parameters>
         <parameter name="paramdate" />
      </parameters>

      <values>
         <value property="date" value="$paramdate" />
      </values>
   </method>
</methods>
</daodefinition>
```

Comme vous le voyez, la balise value accepte deux attributs, property, qui indique la propriété concernée par la mise à jour, et value, contenant la valeur à indiquer dans le champs. Cette valeur peut être une valeur en dur, ou comme ici, la valeur d'un paramètre de la méthode.

Comme pour les méthodes de type delete, vous pouvez également ajouter des conditions.

CopixDAOSearchParams

Présentation

L'objet CopixDAOSearchParams est utilisé dans les DAO pour réaliser des requêtes à condition (findBy, countBy, deleteBy).

Exemple :

```
//On récupère les nouvelles de catégorie 2
_ioDao ('maTable')->findBy (_daoSp ()->addCondition ('id_categorie', '=', 2));
```

Cet objet propose 6 méthodes :

- addCondition pour l'ajout d'une condition de sélection
- addSQL pour l'ajout d'une condition de sélection directement en SQL
- orderBy pour le tri des résultats
- startGroup pour ajouter un groupe de conditions
- endGroup pour terminer un groupe de conditions
- setLimit (ou setOffset & setCount) pour limiter les résultats à un certain nombre d'enregistrements

addCondition

La méthode addCondition permet d'ajouter des restrictions à la recherche.

Elle accepte 4 paramètres qui sont :

- le nom du champ sur lequel s'applique la restriction
- la condition de comparaison que doit supporter le champ
- la valeur avec laquelle le champ sera comparé
- Le type de la condition (obligatoire AND ou non OR), par défaut positionnée à AND

Le nom du champ

Le nom du champ doit être celui que vous utilisez dans votre DAO. Si jamais vous avez défini un fichier XML, c'est le nom de la propriété name qu'il faudra utiliser, et non le nom réel du champ en base de données (fieldname).

Si vous utilisez un DAO automatique, le champ et sa représentation physique ayant le même nom, vous n'avez pas de question à vous poser.

La condition

Les conditions peuvent être tout type de comparaison supportée par les moteurs SQL, à savoir en général :

- = égalité
- ! = différence (sera converti en <> en interne par la méthode addCondition)
- <> différence
- > supérieur (le champ doit être supérieur à la valeur)
- > = supérieur ou égal (le champ doit être supérieur ou égal à la valeur)
- < inférieur (le champ doit être inférieur à la valeur)
- < = inférieur ou égal (le champ doit être inférieur ou égal à la valeur)
- LIKE pour la ressemblance

Cas particulier avec les valeurs null

Si vous spécifiez une valeur NULL avec un addCondition et un opérateur d'égalité ou de différence, Copix générera une requête SQL avec "IS" ou "IS NOT". Inutile donc de gérer vous même ces cas particulier de SQL.

exemple

```
//On compte le nombre de nouvelles sans catégories
_dao ('News')->findBy (_daoSp ()->addCondition ('category', '=', null));

//La requête SQL exécutée sera bien
//select .... from News where category IS NULL

//On compte le nombre de nouvelles avec une catégorie
_dao ('News')->findBy (_daoSp ()->addCondition ('category', '!=', null));
//ou _dao ('News')->findBy (_daoSp ()->addCondition ('category', '<>', null));

//La requête SQL exécutée sera bien
//select .... from News where category IS NOT NULL
```

AND / OR

Exemple d'utilisation des AND / OR avec addCondition

```
// On crée l'objet de critères
$criteres = _daoSp ()->addCondition ('author', '=', 'martin')
                     ->addCondition ('author', '=', 'dupont', 'or');

// On récupère le résultat => liste des news dont l'auteur est martin ou dupont
$resultats = _dao ('News')->findBy ($criteres);
```

Utiliser des tableaux en tant que valeur

Vous pouvez donner un tableau en tant que valeur pour les conditions. Si tel est le cas, Copix réagira comme si vous aviez appelé successivement addCondition avec un OR.

Exemple :

```
// Cette syntaxe avec un tableau...
_daoSp ()->addCondition ('author', '=', array ('martin', 'dupont', 'durand',
            ✂ 'copix'));
//est équivalente à

_daoSp ()->startGroup ()
         ->addCondition ('author', '=', 'martin')
         ->addCondition ('author', '=', 'dupont', 'or')
         ->addCondition ('author', '=', 'durand', 'or')
         ->addCondition ('author', '=', 'copix', 'or')
         ->endGroup ();
```

addSQL

Parfois, certaines condition de sélection sont impossibles à décrire avec un simple addCondition, et il est également dommage d'avoir recours à une **surcharge de DAO** pour ce faire.

Ainsi, il vous est possible de demander à ajouter des conditions SQL sur vos champs, de la façon suivante :

```
$sp = _daoSP ()->addCondition ('titre_test', '=', 'Titre 3')
              ->addSQL ('not exists (select * from copixtestautodao where titre_test =
           ✂  :titre_test)', array (':titre_test'=>2038));
```

La méthode addSQL accepte 3 paramètres :

- le SQL à ajouter. Si la chaine SQL passée est vide, l'instruction est ignorée.
- la valeur des variables de la chaine SQL, au même titre que si vous faisiez une requête en direct avec **CopixDB**
- le type de condition (AND ou OR). Si vous ne souhaitez pas générer de and ou de or, vous pouvez spécifier une chaine vide.

groupBy

La méthode groupBy vous permet de spécifier quels champs seront utilisés pour grouper les résultats.

La méthode groupBy accepte autant de paramètres que de champs à grouper (dans l'ordre).

```
//grouper sur le champs field2
_daoSp()->groupBy ('field2');
//grouper sur le champs field2 et field3 après recherche sur field1
_daoSp()->addCondition('field1','=','foo')->groupBy ('field2', 'field3');
```

orderBy

La méthode orderBy vous permet de spécifier des ordres de tri pour les résultats.

La méthode orderBy accepte autant de paramètres que de champs utilisés dans le tri.

Spécification d'un tri ascendant

```
//exemple 1
_daoSp ()->orderBy ('champ1');

//exemple 2
_daoSp ()->orderBy ('champ1')
       ->orderBy ('champ2');
//équivalent à
_daoSp ()->orderBy ('champ1', 'champ2');
//équivalent à
_daoSp ()->orderBy (array ('champ1', 'ASC'), array ('champ2', 'ASC'));
```

spécification d'un tri descendant

```
//exemple 1
_daoSp ()->orderBy (array ('champ1', 'DESC'));

//exemple 2
_daoSp ()->orderBy (array ('champ1', 'DESC'))
       ->orderBy (array ('champ2', 'DESC'));
//équivalent à
_daoSp ()->orderBy (array ('champ1', 'DESC'),
                    array ('champ2', 'DESC'));
```

setLimit, setCount, setOffset

Ces méthodes permettent de définir des limites sur les enregistrements à récupérer.

```
//Demande à récupérer 4 enregistrements à partir de l'enregistrement numéro 3
_daoSp ()->setLimit (2, 4);
//ou
_daoSp ()->setOffset (2)
         ->setCount (4);

//Demande à récupérer 5 enregistrements
_daoSp ()->setCount (5);

//Demande à récupérer à partir de l'enregistrement 6 (inclus)
_daoSp ()->setOffset (5);
```

startGroup & endGroup

startGroup et endGroup permettent de définir des groupes de conditions. Vous pouvez imbriquer autant de groupes que vous le souhaitez.

startGroup accepte un paramètre qui est le type de groupe (conditions requises (AND) ou facultatives (OR)) au même titre que le 4ème paramètre de addCondition.

Par défaut, le paramètre est positionné à "AND"

Exemple

```
//sélection de tous les éléments sticky ou de tous les éléments non sticky datant de
2006 ou plus
_daoSp ()->addCondition('sticky', '=', '1')
      ->startGroup ('OR')
      ->addCondition('sticky', '=', '0')
      ->addCondition('date', '>', '2006-01-01')
      ->endGroup ();
```

CopixErrorObject

Présentation

Cette classe permet de stocker un ensemble d'erreurs sous la forme d'un objet simple.

Elle est utilisée en interne par les **DAO** et plus particulièrement la **méthode check**.

Vous pouvez, si vous le souhaitez, vous en servir pour vos propres besoins.

Référence

Constructeur

Le constructeur peut prendre en paramètre un objet ou un tableau associatif.

Si tel est le cas, la méthode addErrors sera appelée avec ce paramètre pour pré-initialiser l'objet avec un contenu.

```
$errors = new CopixErrorObject (array ('code rouge'=>'danger très très grave,
          ✂ attention'));
```

addError

La méthode addError permet de rajouter une erreur à la liste existante. Cette méthode accepte deux paramètres : le code de l'erreur ainsi que son libellé.

Si la méthode est appelée plusieurs fois avec le même code erreur, alors les appels successifs avec le même code erreurs écrasent le libellé précédent.

```
$errors = new CopixErrorObject ();
$errors->addError ('code vert', 'moins grave que le code rouge, mais quand même');
```

addErrors

La méthode addErrors permet d'ajouter un ensemble d'erreurs à l'objet courant. Cette méthode accepte en paramètre un tableau associatif ou un objet (pouvant être de type CopixErrorObject).

```
$errors1 = new CopixErrorObject ();
$errors1->addError ('code vert', 'moins grave que le code rouge, mais quand même');

$errors2 = new CopixErrorObject ();
$errors2->addErrors (array ('code rouge'=>'danger très très grave, attention'));
$errors2->addErrors ($errors1);
```

getError

La méthode getError permet de récupérer le libellé d'une erreur donnée. Si le code erreur n'existe pas, la méthode retourne null.

```
$errors = new CopixErrorObject ();
$errors->addError ('code vert', 'moins grave que le code rouge, mais quand même');

echo $errors->getError ('code vert');
```

errorExists

La méthode errorExists indique s'il existe une erreur de code donné.

```
$errors = new CopixErrorObject ();
$errors->addError ('code vert', 'moins grave que le code rouge, mais quand même');
$errors->errorExists ('code vert');//true
$errors->errorExists ('code rouge');//false
```

isError

La méthode isError indique si une erreur est contenue dans l'objet.

```
$errors = new CopixErrorObject ();
$errors->isError ();//false
$errors->addError ('code vert', 'moins grave que le code rouge, mais quand même');
$errors->isError ();//true
```

countErrors

Cette méthode indique le nombre d'erreur actuellement contenues dans l'objet.

```
$errors = new CopixErrorObject ();
$errors->countErrors ();//0
$errors->addError ('code vert', 'moins grave que le code rouge, mais quand même');
$errors->countErrors ();//1
```

asObject

Cette méthode retourne les erreurs sous la forme d'une représentation objet (StdClass) ou le nom de la propriété est le code erreur et la propriété le libellé.

Note Si les codes erreurs ne sont pas des noms de propriété valide, il n'est pas recommandé d'utiliser cette méthode.

Note Si le code erreur commence par un chiffre, alors la propriété commencera par un underscore. Ainsi, le code "012" sera dans la propriété "_012".

asArray

Cette méthode retourne le tableau d'erreur actuellement représenté par l'objet. Ce tableau est un tableau associatif ou les clefs sont les codes erreurs et les valeurs les libellés d'erreur.

asString

Cette méthode retourne les libellés des erreurs uniquement, séparés par une chaine "colle" (par défaut
).

Surcharge_PHP

Redéfinition de classe DAO

Créer des méthodes PHP

Le système de définition xml et le système de déclaration automatique ont leurs limites. Pour un fonctionnel complexe, les méthodes générées peuvent se révéler insuffisantes. On peut ainsi vouloir ajouter des méthodes ou modifier les méthodes existantes sur nos DAO ou nos Record.

Création d'un fichier PHP

Pour surcharger un DAO, il suffit de définir une classe selon deux conventions :

- Le fichier se nomme xxx.dao.php, dans le répertoire classes du module;
- Les noms des classes sont de la forme DAORecordxxx et DAOxxx ;

xxx est le nom du DAO.

Contenu des classes définies

Dans ces classes, vous pouvez rajouter toutes les méthodes et propriétés que vous voulez. Les DAO / record générés par Copix hériteront automatiquement de vos classes.

Note : Si vous développez dans votre classe une méthode PHP qui a le même nom qu'une méthode que Copix est censé générer (findAll par exemple) alors Copix générera cette dernière avec le nom _compiled_xxx (_compiled_findAll par exemple). Ainsi dans votre propre méthode, vous pouvez faire appel à la méthode originale. Par exemple définissons un fichier news.dao.php :

```
class DAONews {
 public function update ($obj){
   // j'update l'objet normalement
   $this->_compiled_update($obj);

   // je fais des traitements en plus
   return CopixDB::getConnection ($this->_connectionName)->doQuery ('...une requête');
 }
}
```

Ainsi vous avez redéfini ici la méthode update. On a décidé qu'elle ferait le traitement standard de l'update, avec en plus d'autres requêtes.

Vous noterez que dans notre objet DAO, on dispose de propriétés qu'il va falloir utiliser si on veut que notre classe reste cohérente par rapport au fichier de définition.

- $this->_table, contenant le nom de la table principale;
- $this->_connectionName, contenant le nom de la connection à la base de donnée;
- $this->_selectQuery, contenant la clause SELECT de base avec tous les champs.

Exemples d'ajout de méthodes

Dans l'exemple suivant, nous avons rajoutés à notre DAO et au Record deux méthodes :

```
class DAOTest {
   function fonctionEnPlus (){
      return 'fonction en plus';
   }
}

class DAORecordTest {
   function autreFonctionEnPlus (){
      return 'autre fonction en plus';
   }
}
```

L'utilisation de ces méthodes est transparente, par exemple :

```
//La méthode du DAO
_ioDAO ('test')->fonctionEnPlus ();

//et la méthode du record :
$test = _record ('test');
$test->autreFonctionEnPlus ();
```

methode_get

Utilisation de la méthode get des DAO

La méthode get des DAO permet de récupérer un enregistrement à partir de sa clef primaire

Exemple :

```
// On récupère l'actualité numéro 10
$news = _dao ('News')->get (10);
```

Le retour de get est un objet de type "**Record**" si l'enregistrement a été trouvé. S'il n'a pas été trouvé, get retourne false.

Avertissement Si vous surchargez la méthode get, veillez à bien retourner un **Record**.

Dans le cas d'une clef primaire composée de plusieurs champs ?

Si la clef primaire de la table "News" est composée de plusieurs champs, il suffit de spécifier tous les champs qui composent la clef primaire à la méthode get, par exemple :

```
// On récupère l'actualité numéro 10
$news = _dao ('News')->get ($clef1, $clef2);
```

methode_insert

Utilisation de la méthode insert des DAO

La méthode get des DAO permet d'inserer un enregistrement

Exemple :

```
// On crée un nouvel enregistrement
$news = _record ('News');
// On affecte les valeurs aux champs
$news->titre = 'Nouvelle actualité';

// On insert l'enregistrement
_dao ('News')->insert ($news);
```

Cas des autoincrement

Si la clef primaire est une séquence ou qu'elle est du type autoincrement, alors la clef primaire est mise à jour lors de l'enregistrement.

par exemple, si le champ id_news est de type autoincrement, on peut en récupérer la valeur dès la fin de l'insert

```
$news = _record ('News');
$news->titre = 'Nouvelle actualité';
_dao ('News')->insert ($news);

$message = 'Nouvelle enregistrée avec identifiant '.$news->id_news;
```

Forcer l'utilisation d'un identifiant (3.0.1+)

Depuis Copix 3.0.1 il est possible de demander à Copix d'utiliser une valeur donnée dans l'identifiant plutôt que de systématiquement utiliser la valeur de l'autoincrement.

Pour ce faire, vous devez spécifier à la méthode insert la valeur true au deuxième paramètre.

Par exemple

```
$record = _record ('copixtestautodao');
$record->id_test = 100;//on veut utiliser 100 comme valeur et non utiliser
            ✂ l'autoincrement
$record->titre_test = 'Titre XXXX';
_dao ('copixtestautodao')->insert ($record, true);//true demande d'utiliser la valeur
            ✂ fournie dans id_test
```

Echec d'insertion

Si jamais l'insertion échoue à cause d'un problème de contrainte (modèle de la base), la méthode insert lance une Exception de type CopixDAOCheckException. Si la requête elle même échoue, alors l'exception générée par CopixDB (CopixDBException[2])sera transmise.

methode_update

Utilisation de la méthode update des DAO

La méthode update des DAO permet de modifier un enregistrement

Exemple :

```
// On récupère l'actualité numéro 10
$news = _ioDao ('News')->get (10);
// On affecte les valeurs aux champs
$news->titre = 'Mon nouveau titre';
// On modifie l'enregistrement
_ioDAO ('News')->update($news);
```

Echec de mise à jour

En cas d'échec de mise à jour, une exception de type CopixDAOCheckException est levée.

Utilisation des versions

Si vous avez spécifié un champ "version" pour gérer automatiquement les concurrences d'accès à vos données et que l'enregistrement que vous tentez de mettre à jour à été modifié entre temps, une exception de type CopixDAOVersionException est levée.

[2] Voir la note de bas de page du chapite CopixDB

methode_delete

Utilisation de la méthode delete des DAO

La méthode delete des DAO permet de supprimer un enregistrement à partir de sa clef primaire

Exemple :

```
// On supprime l'actualité numéro 10
_ioDAO ('News')->delete(10);
```

La fonction retourne 1/0 selon si l'enregistrement à bien été supprimé ou non.

Cas ou la clef primaire est composée

Si la clef primaire de la table "News" est composée de plusieurs champs, il suffit de spécifier tous les champs qui composent la clef primaire à la méthode delete, par exemple :

```
// On supprime l'actualité numéro 10
_ioDAO ('News')->delete ($clef1, $clef2);
```

Valeur retournée

delete renvoie false en cas d'echec, et le nombre d'enregistrements supprimés en cas de réussite.

Voir aussi

- La méthode deleteBy

methode_findAll

Utilisation de la méthode findAll des DAO

La méthode findAll des DAO permet de récupérer tous les enregistrements de la DAO.

Exemple :

```
_ioDAO ('News')->findAll ();
```

Voir aussi

- Méthode **findBy**.

methode_findBy

Utilisation de la méthode findBy des DAO et des conditions (OR)

La méthode findBy des DAO permet de créer des critères de selection en utilisant l'object **CopixDAOSearchParams** :

```
// On crée l'objet de critères
$criteres = _daoSp ()->addCondition('author', '=', 'dupont', 'or')
                     ->orderBy ('date');

// On récupère le résultat => liste des news triée par date ayant pour auteur dupont
$resultats = _dao ('News')->findBy($criteres);
```

Cette méthode retourne un tableau d'objets ou chaque objet est le représentation d'un enregistrement correspondant aux critères de recherches.

Faire des requêtes limitées

En spécifiant le tout avec setLimit

```
// On crée l'objet de critères en demandant à récupérer les 4 premiers enregistrements
            ✂ uniquement
$criteres = _daoSp ()->addCondition('author', '=', 'dupont', 'or')
                     ->setLimit (0, 4)
                     ->orderBy ('date');


// On récupère le résultat => 4 premières news ayant pour auteur dupont
$resultats = _dao ('News')->findBy($criteres);
```

En spécifiant les éléments séparément

```
// On crée l'objet de critères en demandant à récupérer les enregistrements 2 à 12
$criteres = _daoSp ()->addCondition('author', '=', 'dupont', 'or')
                     ->setOffset (1)
                     ->setCount (10)
                     ->orderBy ('date');


// On récupère le résultat => news (2 à 12) ayant pour auteur dupont
$resultats = _dao ('News')->findBy($criteres);
```

methode_countBy

Présentation

La méthode countBy permet de compter le nombre d'enregistrements qui correspondent à des critères de recherches spécifiés via CopixDAOSearchParams.

exemple

```
// On crée l'objet de critères
$criteres = _daoSp ()->addCondition ('author', '=', 'dupont');
$nb = _dao ('News')->countBy ($criteres);
```

La méthode retourne un entier.

Voir aussi

- CopixDAOSearchParams

methode_deleteBy

Présentation

La méthode deleteBy permet de supprimer des enregistrements en spécifiant des critères de sélection avec CopixDAOSearchParams.

exemple

```
// On crée l'objet de critères
$criteres = _daoSp ()->addCondition ('author', '=', 'dupont');
// On supprime tous les messages de dupont
$nbDeleted = _dao ('News')->deleteBy($criteres);
```

La méthode retourne le nombre d'enregistrements supprimés.

orderBy

La clause orderBy des CopixDAOSearchParams n'est pas utilisable avec la méthode deleteBy, pour des raisons évidentes.

Voir aussi

- CopixDAOSearchParams

methode_check

Utilisation de la méthode check des DAO

La méthode check des DAO permet de vérifier la validité des champs de l'enregistrement avant une insertion par exemple. Il est ainsi possible de vérifier les champs obligatoires, les types numériques etc ...

Un exemple d'utilisation :

```
// On crée une DAO
$DAOnews = _dao ('News');
// On crée un nouvel enregistrement
$news = _record ('News');

// On affecte les valeurs aux champs
$news->titre = 'Nouvelle actualité';

// Si la vérification retourne une ou plusieurs erreurs
if ($news->check() !== true) {
  //redirection vers une page d'erreur
}else{
  //mise à jour
}
```

La méthode check retourne true si l'enregistrement est valide, un tableau d'erreurs sinon.

Le tableau d'erreur, si vous n'avez pas surchargé la méthode, est de la forme array (champ=>message en clair).

Voir aussi

- La méthode check générée utilise en interne CopixErrorObject.

Record

Présentation

Les objets "record" sont une composante du système de DAO de Copix. Les "records" sont la représentation des enregistrements de la base.

La méthode CopixDAOFactory::createRecord ('nomDAO'); permet de récupérer la représentation d'un enregistrement "vide" dont le schéma corresponds à la table à laquelle accède "nomDAO".

Exemple

```
$record = _record ('nomDAO');

//Si votre table "nomDAO" dispose des champs id_demande et titre_demande
//alors
var_dump (_record('nomDAO'));

//donne
// object(CompiledDAORecorddemande)
//  public 'id_demande' => null
//  public 'titre_demande' => null
```

CheckException

Présentation

L'Exception CopixDAOCheckException est lancée en cas d'échec de mise à jour (update) ou d'insertion (insert) dans un **DAO**.

Méthodes pour analyser l'erreur

Exemple simple avant de commencer

```
try {
   $dao->insert ($record);
}catch (CopixDAOCheckException $e){
   echo "echec d'insertion avec l'erreur ", $e->getErrorMessage ();
```

getErrorMessage ()

La méthode getErrorMessage () retourne le message d'erreur sous la forme d'une chaine de caractère.

getErrors ()

La méthode getErrors () retourne les messages d'erreurs sous la forme d'un tableau.

Si vous n'apportez aucune surcharge à la méthode check, ce tableau sera de la forme

```
$error['id_champ'] = 'message';
```

getRecord ()

La méthode getRecord () retourne l'enregistrement qui a provoqué l'erreur.

VersionException

Présentation

Lorsque vous utilisez un champ de type "version" dans vos DAO, Copix vérifie lors des mises à jour si vous tentez bien de modifier la dernière version de l'enregistrement (pour éviter la perte de données).

Si ce n'est pas le cas, une exception de type CopixDAOVersionException est levée.

Cette exception dispose d'une méthode getRecord () qui vous retourne le record que vous avez tenté de modifier.

exemple

```
try {
  $dao->update ($record2);//record2 n'est pas à jour
}catch (CopixDAOVersionException $e){
  $record = $e->getRecord ();//corresponds au record qui n'est pas à jour.
}
```

Voir aussi

- Le tutoriel sur la Concurrence d'acces aux données

Documentation des modules

GenericTools

Description

Présentation

Le module generictools contient un certain nombres d'éléments génériques qui vont vous permettre de gagner quelques précieuses minutes lorsque vous développez sous Copix.

Ces éléments sont souvent des petits gadgets, telles des classes utilitaires ou des **actions** déjà réalisées.

Actions réutilisables du module

- **getError** pour l'affichage d'un message d'erreur
- **getConfirm** pour l'affichage d'une demande de confirmation
- **getInformation** pour l'affichage d'un message à l'utilisateur

Classes utilitaires

- **RTFTemplate** pour la génération de documents RTF en mode template.

getError

Présentation

L'action getError du module generictools vous permet de réaliser des écrans avec un simple message d'erreur rapidement.

Exemple d'utilisation

```
public function processQQchose (){
   //... des conditions nous amène à rencontrer un problème
   if (! $toutVaBien){
   return CopixActionGroup::process ('generictools|Messages::getError',
                                     array ('message'=>'Aïe, il y a eu un gros problème',
                                            'back'=>_url ('module|refaire')));
   }
}
```

Paramètres de l'action

message

Le paramètre message est le texte qui sera affiché à l'écran.

back

Le paramètre back corresponds à l'url qui sera appelée lorsque l'utilisateur cliquera sur retour.

TITLE_PAGE

Par défaut, ce paramètre est initialisé grâce à la clef **i18n** messages.titlePage.error (Erreur).

template

Le paramètre template vous permet de spécifier un template à utiliser plutot que d'utiliser celui par défaut (generictools|messages.error.tpl)

Si vous choisissez de spécifier un template personnel, vous devrez prendre en charge les variables suivantes :

- message
- back

getConfirm

Présentation

L'action getConfirm du module genericTools vous permet de réaliser des écrans de confirmation standards.

Exemple d'utilisation

```
public function processAConfirmer (){
   //... des conditions nous amène à demander confirmation d'un message
   return CopixActionGroup::process ('genericTools|Messages::getConfirm',
                                     array ('message'=>'Êtes vous sûr de cela ?',
                                            'confirm'=>_url ('module|actionOui'),
                                            'cancel'=>_url ('module|actionNon')));
}
```

Paramètres de l'action

message

Le paramètre message est le texte qui sera affiché à l'écran.

confirm

Le paramètre confirm corresponds à l'url qui sera appelée si l'utilisateur clique sur oui (s'il confirme l'action)

cancel

Le paramètre cancel corresponds à l'url qui sera appelée si l'utilisateur renonce à l'action qu'il a demandé (s'il clique sur non)

title

Vous pouvez spécifier un titre (de niveau 2)) votre page. Si vous ne spécifiez aucun titre, aucun titre n'est affiché (pas de valeur par défaut).

TITLE_PAGE

Par défaut, ce paramètre est initialisé grâce à la clef **i18n** messages.titlePage.confirm (Confirmation).

template

Le paramètre template vous permet de spécifier un template à utiliser plutot que d'utiliser celui par défaut (genericTools|confirm.tpl)

Si vous choisissez de spécifier un template personnel, vous devrez prendre en charge les variables suivantes :

- title
- message
- confirm
- cancel

Exemple d'utilisation concret

Cette page, issue du module d'authentification standard de Copix, vous permet de supprimer un utilisateur. Avant de supprimer un utilisateur, ce module demande confirmation à l'internaute.

Pour cette confirmation, c'est l'action getConfirm de genericTools qui est utilisée.

```
public function processDelete (){
 if (CopixRequest::getInt ('confirm') == 1){
  _ioDAO ('dbuser')->delete (CopixRequest::getInt ('id'));
  return _arRedirect (_url ('auth|users|'));
 }else{
  if (! ($user = _ioDAO ('dbuser')->get (CopixRequest::getInt ('id')))){
   throw new Exception ('Utilisateur introuvable');
  }
  return CopixActionGroup::process ('genericTools|Messages::getConfirm',
                        array ('message'=>'Supprimer '.$user->login_dbuser. '?',
                               'confirm'=>_url ('auth|users|delete', array
('id'=>$user->id_dbuser, 'confirm'=>1)),
                               'cancel'=>_url ('auth|users|')));
 }
}
```

getInformation

Présentation

L'action getInformation du module generictools vous permet de réaliser des écrans avec un simple message.

Exemple d'utilisation

```
public function processQQchose (){
   //... des conditions nous amène à afficher un message à l'écran de l'utilisateur
   return CopixActionGroup::process ('generictools|Messages::getInformation',
                                     array ('message'=>'Tout va bien !',
                                            'continue'=>_url ('module|next')));
}
```

Paramètres de l'action

message

Le paramètre message est le texte qui sera affiché à l'écran.

continue

Le paramètre continue corresponds à l'url qui sera appelée lorsque l'utilisateur cliquera sur suivant.

TITLE_PAGE

Par défaut, ce paramètre est initialisé grâce à la clef i18n messages.titlePage.information (Information).

template

Le paramètre template vous permet de spécifier un template à utiliser plutôt que d'utiliser celui par défaut (genericTools|messages.information.tpl)

Si vous choisissez de spécifier un template personnel, vous devrez prendre en charge les variables suivantes :

- message
- continue

RTFTemplate

Présentation

La classe RTFTemplate, dont l'interface est très proche de **CopixTpl**, vous permet de générer rapidement et sans efforts des documents RTF.

Exemple d'utilisation

Supposez le document RTF suivant, nommé document.rtf dans le répertoires templates de votre module.

```
Bonjour {$Monsieur},

Merci de votre confiance, .....

votre numéro de commande : {$numeroCommande}.
```

Vous développerez dans votre **action** les éléments suivants pour pouvoir le faire télécharger à votre internaute via **_arContent** :

```
publlic function processDownloadRTF (){
   $tpl = _class ('generictools|RTFTemplate');
   $tpl->assign ('Monsieur', 'Utilisateur de Copix');
   $tpl->assign ('numeroCommande', '123456789');

   return _arContent ($tpl->fetch ('document.rtf'), array
('filename'=>'votre_commande.rtf'));
}
```

Socket

Il est fortement utile de pouvoir ouvrir une connexion sur un hôte à un port donné. De plus, vous serez parfois mené à devoir récupèrer le contenu d'une page WEB mais votre installation PHP ne vous permet pas d'utiliser "fopen" sur une url...

C'est donc à la class Socket de s'en charger.

Comment l'utiliser

La classe Socket fait partie de "generictools". L'instanciation est simple:

```php
<?php
//...
 $sock = _ioClass('generictools|socket');
//...
?>
```

Il existe 5 méthodes:

1. open
2. close
3. write
4. read
5. getHttpContent

Ouvir, écrire, lire...

Voici un exemple simple qui va vous permettre d'ouvrir un socket et de l'utiliser:

```php
<?php
//...
    $s = _ioClass('generictools|socket')->open("www.google.com",80);
    $s->write("GET HTTP/1.0\n\n");
    $response = $s->read();
    $s->close();
//...
?>
```

Avoir le contenu HTTP

Voici la méthode qui va vous permettre de vous passer de fopen pour lire le contenu HTTP. La méthode est `getHttpContent` et prend en paramètre l'url à lire:

```php
<?php
//...
    $url = "http://www.copix/org/wiki/Socket";
    $content = _ioClass('generictools|socket')->getHttpContent($url);
//...
?>
```

`$content` contient le contenu HTML de la page "http://www.copix/org/wiki/Socket"

description

Présentation

Le module comments de Copix (utilisé sur ce même site en bas de page), situé dans stable/tools/comments, vous permet de laisser la possibilité à vos internautes de laisser des commentaires sur n'importe quelle page de votre site, simplement en appelant la zone "comment" depuis vos templates.

Fonctionnalités

- Système de captcha activable
- Possibilité d'indiquer les droits nécessaires à la création de commentaire.
- Système de liens direct vers les commentaire
- Modes plié / déplié
- Administration des commentaires
- Possibilité de vérouiller les commentaires.

Paramètres du module

- captcha, valeur par défaut 0
  - Indique si les captcha sont actifs (1) ou non (0). Vous pouvez administrer les captchas en créant vos propres questions.
- aminitemsperpage, valeur par défaut 10
  - Indique le nombre de commentaire que vous voyez simultanéement dans l'écran d'administration des commentaires.

Zone du module

La zone comment est la zone que vous utiliserez dans vos templates lorsque vous souhaiterez laisser la possibilité à vos internautes de commenter les pages de votre site.

Exemple d'appel de la zone dans Smarty

Cette exemple demande à la zone de permettre des commentaires pour la page courante. L'identifiant du commentaire sera construit à partir des informations module, group, action et title de la **requête**.

required=false permet d'indiquer au **tag copixzone** de ne pas générer de contenu si le module comments n'est pas installé.

```
{copixzone process="comments|comment" id="module;group;action;title" required=false}
```

Paramètre id

Le paramètre id indique les paramètres issus de la requête qui doivent représenter l'identifiant du commentaire.

Vous spécifiez cette liste de paramètre sous la forme d'une chaine séparée par des points virugules ";".

Par exemple

```
{* utilisera les paramètres module group action et id comme identifiant de commentaire*}
{copixzone process="comments|comment" id="module;group;action;id" required=false}

{* utilisera les paramètres module id et langue comme identifiant de commentaire *}
{copixzone process="comments|comment" id="module;id;langue" required=false}
```

Si vous le souhaitez, vous pouvez assigner des valeurs par défaut à ces paramètres, simplement avec l'opérateur d'affectation (=) pour le paramètre en question.

Par exemple

```
{* utilisera les paramètres module group action et id comme identifiant de commentaire
module aura pour valeur test, action aura pour valeur show. Les autres paramètres
utiliseront la requête pour déterminer leur valeur. *}
{copixzone process="comments|comment" id="module=test;group;action=show;id"
required=false}

{* utilisera les paramètres module id et langue comme identifiant de commentaire
langue prendra pour valeur la variable $langue. Les autres paramètres utiliseront
la requête pour déterminer leur valeur.  *}
{copixzone process="comments|comment" id="module;id;langue=$langue" required=false}
```

Paramètre mode

Ce paramètre peut prendre 3 valeurs :

- list pour afficher la liste des commentaires ainsi que le formulaire d'ajout (si les droits sont suffisants)
- summary pour afficher le nombre de commentaires sur l'élément
- request pour utiliser la valeur du paramètre "comments" de la requête pour déterminer le mode.

Par défaut, le mode est défini à "request", puis, si rien n'est spécifié dans la requête, le mode par défaut est summary.

Paramètre moreUrl

Ce paramètre indique l'url ou doit se rendre l'internaute lorsqu'il clique sur le nombre de commentaires (mode summary uniquement).

Paramètre credentialRead

Indique les droits nécessaires pour pouvoir lire les commentaires.

Dans l'exemple suivant, on indique que seuls les internautes connectés peuvent lire les commentaires.

```
{copixzone process="comments|comment" id="module;group;action;id"
credentialRead="basic:registered" required=false}
```

Paramètre credentialWrite

Indique les droits nécessaires pour pouvoir écrire un commentaire.

Dans l'exemple suivant, on indique que seuls les internautes connectés peuvent écrire des commentaires.

```
{copixzone process="comments|comment" id="module;group;action;id"
credentialWrite="basic:registered" required=false}
```

Addons

Mootools

Le renouveau de Javascript

Après un long moment de dénigrement des développeurs WEB quant à Javascript, l'avénement de AJAX à annoncé un retour en force de ce langage. Au delà d'une utilisation "gadget", Javascript est aujourd'hui utilisé au travers de frameworks très évolués pour des opérations complexes, annoncant des interfaces "riches" à l'aube du Web 2.0.

JQuery, script.aculo.us..., et aujourd'hui Mootools sont de la partie. Ces frameworks ont leurs avantages comme leur points faibles. Copix 3 intègre désormais Mootools pour sa souplesse et sa rapidité. Les plugins et fonctions ajoutées sont simplement celles que nous avons préféré.

Voyons ce que nous apporte Mootools et comment utiliser ce framework javascript.

Mootools - framework javascript

Il allait de soit que Copix commence à se pencher de plus près à la partie "vue" et interface pour permettre à l'utilisateur de créer des thèmes plus animés, mais aussi plus riches.

Mootools fait parti de ces frameworks. La Copix Team a décidé d'utiliser celui-ci plutôt que les autres pour différentes raisons:

- mootools est léger
- il est libre, en constante évolution, et l'équipe de création est active
- la communauté participe activement à la création de plugins, d'effets et autres addons
- mootools est très bien implémenté, utilise des concepts avancées de DOM et travaille sur les prototypes basiques pour les réimplémenter (nous le verrons plus tard)
- il travaille essentiellement sur les éléments DOM et CSS ce qui permet de rester standard XHTML.
- il est possible de créer son propre "build" de mootools n'intégrant que nos besoins.

Nous allons nous pencher sur ce que permet Mootools, d'abord en expliquant les bases même du framework, puis sur les effets, les aptitudes de travailler sur l'arborescence DOM, puis Ajax. Index des articles

Principe_de_base

Avant Mootools

Pour rappel, en javascript, tout est objet. A tel point que même les fonctions sont des objets... D'ailleurs, pour nous embrouiller l'esprit, créer une classe (à l'ancienne mode) revenait à écrire une fonction en attribuant des propriétés... Heureusement, Mootools apporte une méthodologie différente qui rendra le code plus clair.

Pour l'heure, un petit rappel est important. En javascript, nous avons une série de types de base: int, float, char, string, array qui représentent respectivement un entier, un nombre décimal, un caractère, une chaine de caractère et un tableau.

Tout ces éléments sont des objets qui ont de méthodes et des propriétés.

Javascript permet non seulement de scripter avec ces types, mais aussi (et surtout !) de pouvoir manipuler la page affichée. C'est en l'occurrence sont but propre et précis. Javascript permet donc de récupérer une descente d'objet à partir de l'objet "window". Le document en lui même est "window.document" et le corps est "window.document.body"...

"window" est appelé implicitement, vous pouvez directement appeler "document" dans appel à "window".

Etant donné que la page web est une série d'éléments avec des attributs, javascript permet de les récupérer de différentes manières. Par exemple par leur identifiant unique ou leur classe. Voici par exemple un élément HTML:

```
<div id="foo">Test de contenu</div>
```

Anciennement (car nous allons utiliser Mootools pour nous simplifier la vie), nous utilisions `document.getElementById('foo')` pour récupérer une référence de l'objet qui a un identifiant nommé "foo". Cet élément peut être un div, une images, un lien... bref tout élément de la page.

Puis nous faisions des manipulations sur l'objet, comme par exemple changer le style, changer le contenu, etc...

Par exemple, changer la bordure revenait à faire: `document.getElementById('foo').style.border="1px solid blue"`. La ligne parait longue, et elle l'est... mais ça va changer.

Avec Mootools

Mootools va surcharger l'ensemble des objets de la page pour leur ajouter des méthodes. De plus, des "accesseurs" ont été créé, réduisant le code que nous devrons taper pour prendre un objet.

Mieux encore, nous allons pouvoir animer et modifier littéralement l'ensemble de la page avec une facilité déconcertante. Enfin, nous allons avoir des classe Ajax permettant de ne pas avoir à recoder la récupération du HTTPRequest-Object.

Attention !

Il est important de comprendre que nous ne pourrons utiliser Mootools et l'arbre dom que si, et seulement si, l'arbre est accessible. De ce fait, tous les exemples que vous verrez devrons être appelés de cette manière:

```
window.addEvent('domready',function (){
        //ici le code
});
```

Si vous ne faites pas cela, vous risquez fortement de voir une erreur javascript comme quoi des méthodes n'existent pas, ou que les éléments n'ont pas de propriétés...

Justement, parlons de DOM

DOM

DOM

DOM est l'acronyme de "Data Object Model" ou "Model Objet de Données"

Dom est la structure objet hiérarchisée des éléments de la page. Chaque éléments peut être enfant ou parent. Par exemple, voici un extrait de page:

```
	<!-- ... -->
	<body>
		<div id="header">L'entête de page</div>
		<div id="contenu">
			<h1>Le titre</h1>
			<p>
				Un paragraphe, <img src="../images/foo.jpg" alt="Foo"
/> et une image.
			</p>
			<p class="bar">
				Un autre paragraphe...
			</p>
		</div>
		<div id="footer">Le pied de page</div>
	</body>
	<!-- ... -->
```

Nous voyons que "contenu" est enfant de "body" tout comme "header" et "footer". "contenu" a 4 enfants (h1, p, p) et le premier paragraphe à encore des enfants (texte, et image)...

En imaginant un peu, nous pouvons représenter l'arbre DOM:

- body
  - header
    - texte
  - contenu
    - h1
  - p
    - texte
    - img
  - p
    - texte
  - footer
    - texte

Chaque élément est typé (div, image, texte, paragraphe...) et a des attributs (id, class, name...). Javascript permet déjà de pouvoir récupérer les éléments via des méthodes plus ou moins simples. Mais avec Mootools, nous allons gagner en vitesse.

Fini le "document.getElementById..." qui se voit aujourd'hui remplacé par la fonction "$"... par exemple:

```
	var a = $('foo')
```

C'est que l'on appelle un "accesseur".

Cela permet de manipuler l'élément ayant pour id: 'foo'. La fonction "$" nous donne presque l'impression de coder en PHP, et en fait nous avons bien plus l'impression de manipuler une variable qu'un élément HTML... C'est le but !

Il est possible de prendre une collection d'éléments en utilisant la fonction "$$". Celle-ci attend un attribut de syntaxe CSS. Par exemple, pour récuperer les éléments de classe "bar", il faut faire:

```
	var b = $$('.bar')
```

Et mieux encore:

```
	var b = $$('.bar'); //ensemble d'élements de classe 'bar'
	var b = $$('.bar a'); //ensemble de lien contenu dans l'élément de classe
'bar'
	var b = $$('a.bar'); //ensemble de lien ayant la classe bar
	var b = $$('input[type=button]'); //ensemble de bouton de la page
	var b = $$('div#foo .baz'); //ensemble des élément de classe 'baz' contenu
dans un div d'id "foo"
	//...
```

La collection est un tableau itératif. Mootools ajoute la méthode qui n'était accessible qu'avec Gecko: "each". De ce fait, pour parcourir la collection d'éléments:

```
	$$('.bar').each(Foo)
```

Où "Foo" est la méthode appelée pour chaque éléments trouvé. "Foo" prend en argument l'élément et l'itération:

```
	$$('.bar').each(Foo);
	function Foo(el,i){
		//...
	}
```

En manipulant "el", vous manipulez en fait les éléments ayant pour class "bar". Le plus simple reste tout de même l'utilisation de "fonction anonyme":

```
	$$('.bar').each(function (el,i){
		//...
	});
```

De cette manière, on ne déclare par de fonction, on en crée une en mémoire le temps de manipuler la collection. C'est la méthode **recommandée**.

Un aspect intéressant de l'arborescence DOM est de pouvoir déplacer un élément dans un autre, avant un autre, derrière un autre, ou de remplacer un élément par un autre. Vous pouvez aussi créer un nouvel élément, et le placer dans l'arbre DOM.

Imaginons :

```
<div id="foo">
        <div id="bar"> Contenu de BAR </div>
        <div id="baz"> Contenu de BAZ </div>
</div>
```

Notre arborescence DOM (sommes toute très simplifié je vous l'accorde) peut se modifier de la sorte:

```
$('baz').injectBefore('bar'); //déplace le div "baz" au dessus de "bar", baz
    ✂ est toujours dans le div "foo"
$('baz').injectAfter('foo'); //déplace le div "baz" après le div "foo", il
    ✂ n'est donc plus dans ce div
$('baz').replace('bar'); //remplace "bar" par "baz", le div "bar" disparait et
    ✂ n'est plus récupérable !
$('baz').remove(); //supprime le div "baz"

//création d'un élément "div" et le placer dans "foo"
var d = new Element('div');
d.injectInside('foo');

//chaque fonction retourne l'élément, on peut donc chainer les appels:
new Element('div').injectInside('foo');

//Allons plus loin:
new
Element('div').injectInside('foo').setStyle('color','#AAEEAA').setHTML('Salut le
    ✂ monde!');
```

Il est donc possible de manipuler complètement la page et de changer la structure complète. Cela peut-être très utile pour des thèmes, des modules qui veulent déplacer des éléments, ou pour ajouter des fonctionnalité seulement si javascript est activé.

Il est temps de passer à la **méthodologie mootools**

Methodologie

La méthodologie Mootools

Je reviens donc à quelque chose de plus technique, mais qui doit être lut si vous voulez travailler correctement avec Mootools. Ce framework a une méthodologie qui demande une petite étude. Tentez de **toujours** la respecter afin d'être standard à Mootools et ainsi pouvoir faire profiter de vos scripts à la communauté. De plus, la maintenance de votre code sera bien plus aisée.

Les éléments

L'un des buts de Mootools est de modifier et animer les éléments. De ce fait, la classe de base que nous manipulons pour ce cas précis est "Element".

Tous les éléments HTML sont alors surchargés pour implémenter la classe "Element". Vous pourrez alors modifier ces objets et implémenter de nouvelles méthodes.

Mootools est livré avec un lots de plugins. Ces plugins surchargent eux-même la classe Element dans certain cas. Notez par exemple makeDraggable qui n'existe que dans le cas où le plugin "drag" est installé (Copix 3 installe tous les plugins de base).

Les arguments en objet

Les fonctions qui peuvent recevoir un grand nombre d'options doit reçevoir un objet. C'est d'ailleurs ce que fait mootools dans la totalité des cas. Par exemple:

```
$('div1').setStyles({
        'font-size': "1em",
        'color': "#EEFFEE"
});
```

De votre coté, voici comment procéder:

```
function Bar (el,options){

}
```

Faites des classes

L'intérêt de créer des classes est de pouvoir réutiliser vos scripts sans soucis. De plus, vous pourrez surcharger les classes mootools pour ajouter des méthodes au framework (ce sera alors un plugin).

```
var MaClass = new Class({
        initialize: function(){
                //constructor
        }
})
```

Pour respecter le système d'options en objets:

```
	var MaClass = new Class({
		options: {
			'color': '#AAAAAA',
			'message': 'hello !'
		},
		initialize: function(options){
			//constructor
			//merge les options par défaut et celles données en paramètre
			this.options = $merge(options,this.options);
		}
	})
```

Dans ce cas, si nous appelons:

```
	var m = new MaClass({message: 'My new message'});
```

Les options seront:

```
	{
		color: '#AAAAAA',
		message: 'My new message'
	}
```

Surcharge de Mootools

Mootools ajoute des méthodes aux éléments, par exemple injectInside, ou setStyles... vous pouvez en ajouter vous même sans avoir à modifier le code du framework. Le principe est simple, implémentons une méthode "mySimpleHello":

```
	Element.implement({
		mySimpleHello: function(){
			this.setHTML('Hello world');
			return this; //un conseil, retournez toujours l'objet pour
pouvoir chainer les appels
		}
	})
```

De ce fait:

```
	<div id='test'>Un message</div>
	<script type="text/javascript">
		window.addEvent('domready',function(){
			$('test').mySimpleHello();
		})
	</script>
```

Le contenu du div est immédiatment remplacé par "Hello world". Vous verrez que cela est extrêmement utile quand vous commencerez à maitriser Mootools. Il est toujours plus simple d'appeller une méthode d'un élément HTML plutôt que d'appeler une fonction en envoyant en paramètre l'élément.

les évènements

Il faut éviter d'utiliser les attributs de balise "onClick", ou "onMouseOver"... et utiliser plutôt Mootools pour les définir. Voici des exemples:

```
$('test').addEvent('mouseenter',function(){}); // la fonction a appelé si la
    ✂ souris entre sur l'élément
$('test').addEvent('mouseleave',function(){}); // la fonction a appelé si la
    ✂ souris sort de l'élément
$('test').addEvent('click',function(){}); //  click de la souris sur l'élément
$('test').addEvent('keypress',function(){}); //  une touche du clavier est
    ✂ pressée
$('test').addEvent('keyup',function(){}); //  une touche du clavier est pressée
    ✂ et relachée
```

La fonction attend un évènement, il faut le transtyper en évènement mootools et jouer avec le contenu:

```
$('test').addEvent('keyup',function(e){
        e = new Event(e); //transtypage
        console.debug(e); //valable si vous utiliser Firebug sur Firefox
        if(e.key=='esc'){
                //par exemple, si la touche pressée est Echap...
        }
});
```

Maintenant, amusons nous un peu avec les effets.

Effets

Les effets

Avec mootools vient les classes dérivées de "Fx". Plusieurs plugins à "Fx" sont déjà préinstallés comme: Style et Styles, slide, scroll,... Fx comprend des méthodes de base: start, stop et set.

`start` lance l'animation, `stop` l'arrête. Quant à `set` il permet d'aller directement à la fin de l'animation.

Chaque animation peut prendre au moins un jeux d'options:

- transition: c'est une équation de transition que vous pouvez voir dans la classe Fx.Transitions par defaut Fx.Transitions.Sine.easeInOut
- duration: la durée de l'animation en milisecondes
- unit: l'unité de mesure, par défaut le pixel (px) est utilisé.
- wait: true ou false pour définir si il faut attendre la fin de l'animation pour en commencer une nouvelle sur la même instance d'effet
- fps: le nombre de rafraichissement par seconde, par défaut 50;

Ensuite, l'animation est simple à mettre en place, regardez:

```
<div id="test">Mon texte</div>
<script type="text/javascript">
        window.addEvent('domready',function(){
                //faison un "fade-in":
                //d'abord, on passe l'opacité à 0:
                $('test').setStyle('opacity', 0);

                //créons l'objet d'animation
                var f = new Fx.Style ($('test'),'opacity', {duration: 900,
    ✂ wait: false});
                //et on le lance
                f.start(1);
        })
</script>
```

L'objet passe son opacité de 0 à 1 pendant 900ms.

Tutoriels

Exemples de base

Hello_You__

Objectifs

L'objectif de ce tutoriel est de vous faire développer une première page avec Copix, de façon à introduire les différents concepts de l'outil.

Nous allons ici répondre à l'url index.php/default/hello/you (qui peut également revêtir la forme index.php?group=hello&action=you).

Note: pour en savoir plus sur les URL Copix, consultez la page "**comprendre la forme des URL**".

Introduction

Dans Copix, une "URL/page HTML" correspond à une action. Ces actions sont implémentées dans des *ActionGroup*. Ces ActionGroup dérivent tous de *CopixActionGroup*.

Dans Copix, vous allez toujours développer dans des *modules*. Pour ce tutoriel, nous allons créer des éléments dans le module *default* (situé dans *project/module/public/stable/standard/default*)

Notre première page

Créez le fichier *project/module/public/stable/standard/default/actiongroups/hello.actiongroup.php* avec le contenu suivant :

```
<?php
   /**
    * Page d'accueil et fonctionnalité standard.
    * Cet objet peut gérer en standard les urls de la forme
index.php/moduleConcerné/hello/xxxx
    */
    class ActionGroupHello extends CopixActionGroup {
        /**
         * Notre premier exemple
         * Cette dernière implémente en standard l'url
index.php/moduleConcerné/hello/you
         */
        function processYou () {
        }
    }
?>
```

Implémentation de l'action

Nous souhaitons ainsi afficher une page avec dedans "hello you".

```
        function processYou () {
            $ppo = new CopixPPO ();//création de l'objet de données
            $ppo->name = 'you';  //assignation d'une donnée
            return _arPPO ($ppo, 'hello.tpl');//demande d'affichage des données
        }
```

Copix utilise un **modèle MVC** (Modèle Vue Controller).

Ici, nous allons donc renseigner les variables du modèle dans la variable "$ppo" (PPO comme Plain PHP Object) qui est un simple objet permettant de véhiculer les données.

Nous attribuons à la variable $ppo->name la valeur "you".

Ensuite, nous indiquons que nous souhaitons afficher les données de la variable *$ppo* dans le template *hello.tpl*

Création du template

Créez le fichier *project/modules/public/stable/standard/default/templates/hello.tpl* avec le contenu suivant :

```
<p>Hello {$ppo->name}</p>
```

En standard, Copix utilise Smarty comme moteur de templates. Si vous le souhaitez, vous pouvez écrire vos template directement en PHP. Dans ce cas, il faudra utiliser l'extension ".php" pour ces derniers, et le code aurait été de la forme :

```
<p>Hello <?php echo $ppo->name; ?></p>
```

Hello_You_Strikes_back__

Introduction

Si vous avez suivi le tutoriel **Hello you !**, alors vous êtes prêts pour ce nouveau défi.

Le but ici est de proposer un formulaire permettant de saisir le nom de l'internaute afin de lui souhaiter la bienvenue par la suite.

Commençons par les templates

Le formulaire

Vous pouvez copier ce code dans un fichier formulaire.tpl, à mettre dans le répertoire templates du module :

```
<form action="{copixurl dest="default|hello|someone"}" method="POST">
<input type="text" name="someone" value="" /><br />
<input type="submit" value="Envoyer" />
</form>
```

L'élément nouveau dans ce formulaire est l'utilisation d'une *balise Smarty* {*copixurl*}. Les *balises Smarty* permettent de réaliser dans les *templates* des tâches courantes (affichage d'un calendrier, mise en majuscule, liste déroulante, ..., ...).

Ici, {copixurl} permet de générer l'URL permettant d'aller à la page du module *default*, dans l'*ActionGroup* hello pour y chercher l'*action* someone.

Une chose interressante est que nous utilisons systématiquement "POST" pour les formulaires dans Copix, pour des raisons de compatibilité avec certains navigateurs.

Hello !

Le template Hello est le même que nous avons utilisé précédemment, pour rappel :

```
<p>Hello {$ppo->name}</p>
```

Implémentation des actions

Reprenons notre *ActionGroupHello* et ajoutons les actions *formulaire* et *someone*

```
/**
* Affichage du formulaire pour demander le nom de l'utilisateur
*/
function processFormulaire (){
   return _arPpo (new CopixPPO (), 'formulaire.tpl');
}

/**
* Affichage du formulaire pour demander le nom de l'utilisateur
*/
function processSomeone (){
    //création de l'objet de données
    $ppo = new CopixPPO ();
    //On récupère la valeur du champ someone du fomulaire. Ici, on utilise getAlphaNum
            ✂ pour s'assurer de ne récupérer que les caractères AlphaNumériques.
    $ppo->name = CopixRequest::getAlphaNum ('someone');
    //demande d'affichage des données
    return _arPpo ($ppo, 'hello.tpl');
}
```

Avec une base de données

EUGB__Encore_un_gestionnaire_de_breves_

Objectifs

Si Copix est **correctement installé** avec une base de données configurée par défaut, il est temps de découvrir les DAO :-)

Note: DAO est l'acronyme de "Data Access Object".

L'objectif de ce tutoriel est de créer une page affichant une liste de nouvelles issues de la base de données.

Schéma de la table NEWS pour MySQL

Dans la base par défaut, créez la table NEWS.

```
CREATE TABLE news (
  news_id int(11) NOT NULL AUTO_INCREMENT,
  news_date varchar (8) NOT NULL DEFAULT '00000000',
  news_sujet varchar(255) NOT NULL DEFAULT '',
  news_texte text NOT NULL,
  PRIMARY KEY (news_id)
) CHARACTER SET utf8;
```

Implémentation de l'action

Nous allons développer dans le module "default" (*project/modules/public/stable/standard/default*) et créer un *ActionGroup News* dans le fichier *project/modules/public/stable/standard/default/actiongroups/news.actiongroup.php*.

Nous allons également implémenter la méthode *processList* pour répondre à l'url *index.php/default/news/list*.

```
<?php
/** Gestion de nouvelles
*/
class ActionGroupNews extends CopixActionGroup {
  /** Pour afficher la liste des news à l'écran
  * index.php/default/news/list
  */
  function processList (){
     $ppo = new CopixPPO ();//création de l'objet pour transporter les données
     $ppo->TITLE_PAGE = 'Liste des nouvelles';//On donne un titre à la page avec la
           ✂ variable spéciale TITLE_PAGE
     $ppo->arNews = _dao('news')->findAll ();//Copix est capable de dialoguer
automatiquement avec une base MySQL (voir plus en détail les DAO dans la documentation)
     return _arPPO ($ppo, 'news.list.tpl');//On demande l'affichage des données dans le
           ✂ template news.list.tpl
  }
}
?>
```

Note: _dao est une fonction raccourcis pour CopixDAOFactory::create. N'hésitez pas à consulter la documentation de ces fonctions pour économiser vos claviers !

Création du template

Voici le code du template Smarty utilisé pour afficher le tableau de news (à mettre dans *project/modules/public/stable/standard/default/templates/news.list.tpl*) :

```
{**
* Template d'affichage d'une liste de news.
* @param arNews - tableau d'objets News.
*}
<table border="1">
  {foreach from=$ppo->arNews item=objNews}
    <tr>
      <td>{$objNews->news_date}</td>
      <td>{$objNews->news_sujet}</td>
      <td>{$objNews->news_texte}</td>
    </tr>
  {/foreach}
</table>
```

CopixDB pour faire soi-même les requêtes

Objectifs

Le tutoriel du système de DAO ne vous a pas plus ou vous voulez simplement faire vos requêtes complexes à la main ? Pas de problème, cette section est faite pour vous !

Requêtes de sélection

```
$arResults = CopixDB::getConnection ()->doQuery ('select champ1, champ2 from
tableTest');
```

En résumé, on demande à CopixDB de nous fournir une connexion à la base de données par défaut, puis d'exécuter une requête de sélection.

Les résultats de la requête de sélection seront stockés dans un tableau ($arResults) d'objets ayant une propriété par champ. On peut ainsi afficher les valeurs de cette façon dans un template :

```
foreach ($arResults as $record){
   echo $record->champ1, '-', $record->champ2, '<br />';
}
```

Requêtes de sélection paramétrées

Si vos requêtes disposent de paramètres, plutôt que de concaténer une chaine de caractères SQL, nous vous conseillons vivement d'utiliser le système de paramètres (évitant tout risque de SQLInjecton).

```
$arResults = CopixDB::getConnection ()->doQuery ('select champ1, champ2 from tableTest
           ✂ where champCritere=:valeurCritere', array (':valeurCritere'=>
           ✂ CopixRequest::get ('filtreUtilisateur')));

//exemple avec un LIKE
$arResults = _doQuery ('SELECT idEleve AS id, nom, prenom1 AS prenom FROM eleve WHERE
           ✂ nom LIKE :vnom', array (':vnom'=>'debut%'));
```

Note _doQuery est une fonction raccourci de CopixDB::getConnection ()->doQuery ().

Pour indiquer un paramètre, comme vous le voyez, il suffit de nommer une variable :qqChose, **sans** l'entourer de guillemets (même pour une chaine de caractères). Ensuite, pour définir sa valeur à partir d'une variable PHP, il suffit de passer un tableau associatif avec

- clef = identifiant variable,
- valeur = valeur à affecter au paramètre.

Ici, comme nous utilisons un paramètre, il n'existe aucun risque de SQLInjection dans la requête, et vous pouvez ainsi utiliser la valeur donnée par l'internaute tranquillement.

Autres requêtes ?

Les requêtes de type update, delete, insert retournent le nombre d'enregistrements affectés par l'opération et non un tableau (à l'évidence).

Si la requête échoue ?

Si la requête est un échec, alors une exception de type CopixDBException est levée.

Gerer_les_transactions

Objectifs

Lorsque vous devez lancer tout un tas de requêtes sans être sur que ces dernières arrivent à terme, il est souvent laborieux de gérer les transactions à la main, d'autant plus si vous utilisez plusieurs objets métiers, et encore plus si vous utilisez plusieurs connexions ou bases de données.

Copix propose ainsi une API unique et simple pour gérer dans une seule et même "transaction Copix" une multitude de requêtes et connexions, une multitude de bases, ou que se fassent les appels.

La solution en quelques lignes

```
CopixDB::begin ();
try {
   CopixDB::getConnection ()->doQuery ('....');
   CopixDB::getConnection ('autreConnexion')->doQuery ('....');
   $unObjet->faitDesChosesAvecDesBases ();
   CopixDB::commit ();
}catch (Exception $e){
   CopixDB::rollback ();
}
```

En gros, CopixDB::begin indique à l'API de base de données de mettre en place un contexte transactionnel à toutes les nouvelles demandes de connexions qui seront réalisées.

Les rollback et commit seront appliqués à toutes ces sources une fois l'opération terminée.

Copix est également capable de gérer des transactions imbriquées si votre base de données le permet.

Gestion transactionnelle automatique

Introduction

CopixServices est la classe de base pour l'implémentation des services sous Copix.

Lorsque vous utilisez un service, vous êtes assurés que tout s'exécute dans un contexte transactionnel (et donc que votre base n'est modifiée que si les opérations sont réussies).

Exemple d'utilisation

```
try {
   //demande de publication de la nouvelle d'identifiant 4
   $results = CopixServices::process ('module|News::publish', array ('id_news'=>4));
}catch (exception $e){
   //CopixServices aura fait un rollback pour nous car l'opération a échouée.
   //Il ne nous reste plus qu'a indiquer à l'utilisateur que ce n'est pas possible.
}
```

Exemple d'implémentation d'un service

Dans le fichier *cheminModuleNews/news/classes/news.services.php*

```
class ServicesNews extends CopixServices {
   private function publish (){
      //On trace dans la table d'historique les modifications
      $recordHistorique = CopixDAOFactory::createRecord ('NewsHistorique');
      $daoHistorique    = CopixDAOFactory::create ('DaoHistorique');
      $recordHistorique->date_hist = date ('Ymd');
      $daoHistorique->insert ($recordHistorique);

      $daoNews = CopixDAOFactory::create ('News');
      if (($newsRecord = $daoNews->get ()) !== null){
         $newsRecord->status_news = NEWS::Published ();
         $daoNews->update ($newsRecord);
      }else{
          throw new ApplicationException ("La nouvelle à publier n'existe pas");
      }
   }
}
```

Dans cet exemple, si nous arrivons dans le cas ou la nouvelle n'existe pas (on lance l'exception), Copix annule pour vous les insertions qui ont déjà été réalisées via le $daoHistorique.

En bref, si jamais une exception est levée par votre service, toutes les modifications en cours sur les bases sont annulées automatiquement !

Paramétrage de la gestion transactionnelle

Au moment de l'appel, vous pouvez spécifier la façon dont vous voulez que CopixServices gère la transaction

- CopixServices::NEW_TRANSACTION - Réalise un begin avant de démarrer le service, rollback en cas de problème sinon commit. (C'est le mode par défaut si vous ne spécifiez rien)
- CopixServices::CURRENT_TRANSACTION - Intègre les requêtes dans la transaction courante sans faire appel a begin / commit. Si aucune transaction n'est active, alors le mode est "auto commit" à chaque requête effectuée dans la base.

Exemple

```
   CopixServices::process ('SomeService::someMethod', array ('param'=>$param),
CopixServices::NEW_TRANSACTION);
```

Il est temps de créer un module

Création d'un module

Objectifs

Dans les précédents tutoriels, nous avons directement modifié les éléments du module *default*.

Ce module *default* correspond au module exécuté par défaut lorsqu'aucune information n'est indiquée à Copix.

Il est temps pour vous d'être initié à la création de module, modules qui vous permettront de distribuer vos applications sur les plateformes réalisées avec Copix.

Choisir un emplacement

Par défaut, Copix est livré pré-configuré avec deux emplacements de modules par défaut (une sorte de include_path). Ces emplacements sont *project/modules/public/stable/standard* et *project/modules/publis/devel/webtools*.

Lorsque vous parcourez la page d'administration dans la section "ajouter / supprimer des modules", c'est ce que Copix vous rappelle en bas de page.

Vous pouvez créer votre module dans l'un de ces emplacements ou **ajouter votre propre emplacement de module**.

Etape 1 - Création du répertoire

Pour ce tutoriel, créons notre module dans *project/module/public/devel/webtools/monmodule*.

Créer le répertoire n'est pas suffisant pour qu'il soit considéré comme un module, il va falloir également créer un fichier **module.xml**.

Etape 2 - Création du fichier module.xml

Créez le fichier *project/module/public/devel/webtools/monmodule/module.xml* avec le contenu suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<moduledefinition>
 <general>
  <default name="monmodule" description="Mon premier module"
        longdescription="Mon premier module de test pour jouer avec Copix" />
 </general>
</moduledefinition>
```

A partir de maintenant, ça y est, votre module va apparaitre dans la liste des modules dans l'interface d'ajout/suppression de modules !

Après l'avoir installé, vous ne devriez maintenant pas avoir de mal à reproduire les étapes du tutoriel **Hello You !** pour réaliser votre première page dans votre module.

Plus loin avec les modules

Plus tard, vous verrez que Copix est capable de gérer les dépendances entre les modules, que vos modules peuvent être multilingues, vous verrez également qu'il est possible de rendre vos modules paramétrables, qu'il est possible lors de leur installation / suppression de lancer un ensemble de scripts SQL, ...

Création de scripts d'installation / désinstallation pour un module

Objectifs

Maintenant que vous savez créer un module, vous avez surement envie de le partager avec d'autres et de fournir un package prêt à l'emplois !

Pour se faire, Copix intègre une nomenclature vous permettant de distribuer votre module avec ses scripts de base de données.

Répertoire nomDuModule/install/

Si vous placez un fichier nommé *nomDuModule/install/scripts/install.xxx.sql*, alors Copix exécutera ce dernier au moment de l'installation du module. Simple et efficace :-)

"xxx" corresponds au nom du driver que vous aurez configuré lors de l'installation de Copix. Par exemple, avec MySQL, le driver se nomme "pdo_mysql" et votre fichier sera donc nommé install.pdo_mysql.sql.

Note: Pour placer des scripts de suppression pour votre module, il vous suffit de placer un fichier nommé delete.xxx.sql dans ce même répertoire.

Vous pouvez egalement rajouter un script PHP. Pour cela, créer un fichier *nomDuModule/install/scripts/nomDuModule.class.php*

Celui ci doit contenir du code selon ce schéma :

```
class CopixModuleInstallerNomDuModule implements ICopixModuleInstaller {

        public function processInstall () {
        }

        public function processDelete () {
        }

        public function processUpdate () {
        }
}
```

La méthode processInstall() sera executée a l'installation du module La méthode processDelete() sera executée a la désinstallation du module TODO : update

Evénements liés à l'installation / désinstallation (Copix 3.0.2 minimum)

Plusieurs événements sont disponibles lors de l'installation / désinstallation de modules.

Voir aussi : Gestion d'événements, Récupérer un événement

beforeInstallModule

Cet événement est envoyé avant toute opération d'installation d'un module.

Paramètre passé :

- moduleName : le nom du module

Retour du listener :

- install : boolean, si indéfini ou true le module s'install, si false, le module ne s'installe pas et une CopixException avec le texte du paramètre message est générée.
- message : message d'erreur en cas de retour avec install = false.

afterInstallModule

Cet événement est envoyé après l'installation complète d'un module, et après que le cache concernant les modules soit régénéré.

Paramètre passé :

- moduleName : le nom du module

Retour du listener :

- install : boolean, si indéfini ou true le module sera installé, si false, le module est désinstallé et une CopixException avec le texte du paramètre message est générée.
- message : message d'erreur en cas de retour avec install = false.

beforeUninstallModule

Cet événement est envoyé avant toute opération de désinstallation d'un module.

Paramètre passé :

- moduleName : le nom du module

Retour du listener :

- uninstall : boolean, si indéfini ou true le module se désinstalle, si false, le module ne se désinstalle pas et une CopixException avec le texte du paramètre message est générée.
- message : message d'erreur en cas de retour avec uninstall = false.

afterUninstallModule

Cet événement est envoyé après qu'un module soit désinstallé.

Paramètre passé :

- moduleName : le nom du module

Mieux comprendre l'arborescence de Copix

Arborescence principale de Copix

Lorsque vous décompressez Copix, vous trouvez les répertoires suivants :

- *var*/ par convention, tous les fichiers non temporaires manipulés par Copix sont stockés ici. Ce chemin est représenté par la constante COPIX_VAR_PATH.
- *temp*/ tous les fichiers de cache, les fichiers générés par Copix (dao, ressources, paramètres, ...). Ce chemin est représenté par la constante COPIX_TEMP_PATH.
- *project*/ Les éléments spécifiques au projet. Ce chemin correspond à la constante COPIX_PROJECT_PATH.
  - *config*/ Emplacement par défaut du fichier de configuration (**copix.conf.php**)
  - *modules*/ Là où vous allez travailler ! En général (ce n'est pas obligatoire), c'est ici que l'on positionne les modules.
  - **project.inc.php**
- *utils*/ Les fichiers nécessaires au fonctionnement de Copix (le framework lui même)
- *www*/ Les fichiers qui seront publiés sur le serveur web dans le "DocumentRoot"
  - **index.php** Le seul fichier php qui doit être accessible aux utilisateurs pour lancer Copix

Arborescence d'un module

Dans Copix, nous développons souvent des modules pour répondre aux demandes de l'utilisateur. Ces modules disposent de cette arborescence :

- nom_du_module (doit être en minuscules)
  - actiongroups les groupes d'actions, ce sont les contrôleurs de page
  - classes c'est ici que vous placerez vos objets métiers
  - resources c'est ici que vous mettrez vos fichiers i18n et la déclaration de vos DAO
  - templates tous les modèles d'affichage de vos modules sont ici
  - zones les zones sont des objets capables de prendre en charge un élément graphique indépendant.
  - **module.xml** le fichier de déclaration de votre module **fichier requis**

Ajouter votre propre emplacement de module

Objectif

Lorsque vous avez téléchargé Copix, ce dernier était configuré avec deux emplacements de modules.

Pour des raisons pratiques (modules partagés, différents gestionnaires de sources, ...) vous pouvez être amené à en rajouter de nouveau.

Et bien comme d'habitude, c'est très simple.

dans le fichier project/config/copix.conf.php

Trouvez la ligne

```
$config->arModulesPath = array (
 COPIX_PROJECT_PATH.'modules/public/stable/standard/',
 COPIX_PROJECT_PATH.'modules/public/devel/webtools/'
);
```

Cette ligne assigne dans un tableau la liste de tous les emplacement de module. Ici, on utilise une **constante** COPIX_PROJECT_PATH qui corresponds à l'emplacement de votre répertoire project.

Pour rajouter votre emplacement, vous pouvez donc faire :

```
$config->arModulesPath = array (
 COPIX_PROJECT_PATH.'modules/public/stable/standard/',
 COPIX_PROJECT_PATH.'modules/public/devel/webtools/',
 COPIX_PROJECT_PATH.'modules/private/mes_modules/'
);
```

Dorénavant, Copix ira chercher les modules dans ce nouvel emplacement (*COPIX_PROJECT_PATH.'modules/private/mes_modules/*)

Quelques plugin

Qu'est-ce qu'un plugin ?

Objectifs

Cette page a pour but d'expliquer ce qu'est un plugin dans Copix.

Rôles & Possibilités

Un plugin est une petite classe qui respecte une interface donnée, et que Copix appelle systématiquement à chaque demande de page.

Un plugin est donc utilisé lorsque vous voulez étendre les fonctionnalités du framework avec des actions systématiques.

On peut supposer un plugin pour des tâches telles que :

- Renforcement de la protection d'un ensemble de pages
- Anti-aspirateurs de sites
- Changement de thèmes graphiques
- Statistiques
- Logs des pages appelées
- Modification des pages avant affichage
- Inclusion de bibliothèques de classes
- ...

L'interface d'un plugin

```
abstract class CopixPlugin {
    public function beforeSessionStart ();
    public function beforeProcess (& $action);
    public function afterProcess ($actionreturn);
    public function beforeDisplay (& $display);
}
```

beforeSessionStart

Cette méthode sera appelée avant le démarrage de la session (session_start).

beforeProcess (& $action)

Cette méthode est appelée avant que Copix n'instancie l'**ActionGroup** choisie. Vous avez donc ici une dernière chance pour changer l'action retenue (qui est passée en paramètre).

afterProcess ($actionreturn)

Cette méthode est appelée alors que Copix a terminé l'appel à l'Action.

Vous disposez ainsi d'une opportunité pour modifier le retour (par exemple rediriger les demandes de téléchargements lourds sur un autre serveur, compresser les fichiers à télécharger, convertir des formats, ...)

beforeDisplay (& $display)

Cette méthode est appelée alors que Copix a complètement terminé une demande d'affichage.

Vous trouverez dans $display le contenu HTML que Copix s'apprête à afficher. Vous pouvez ainsi appeler des bibliothèques de traitement HTML, faire une fonctionnalité de glossaire sur certains mots de la page, ajouter une bannière publicitaire, ...

Activer un plugin

Objectifs

Maintenant que vous savez ce qu'est un plugin, voyons comment faire pour les activer.

Une simple tour dans l'interface d'administration

Pour activer un plugin, il suffit d'aller dans la partie activation/désactivation des plugins de l'interface d'administration et d'ajouter son plugin.

Développer un plugin

Objectifs

Nous allons ici apprendre à créer un premier plugin qui aura pour rôle de mettre dans le log l'ensemble des URL appelées.

Nous allons créer ce plugin dans le module "default".

Création du répertoire pour notre plugin

Les plugins sont créés dans le répertoire "plugins" des modules concernés.

Ainsi, pour créer un plugin "mon_plugin" dans le module "default", nous allons créer le répertoire *project/modules/public/stable/standard/default/plugins/mon_plugin*.

Comme vous pouvez le constater, il existe déja plusieurs plugins dans le module default. Nous vous invitons à les lire pour vous familiariser avec les plugins.

Création du fichier de plugin

Créez le fichier *project/modules/public/stable/standard/default/plugins/mon_plugin/mon_plugin.plugin.php*

Dans ce dernier, créez le contenu suivant :

```
<?php
class PluginMon_Plugin extends CopixPlugin {
   public function beforeProcess (){
      CopixLog::log (CopixUrl::getCurrentUrl (), $this->config->getLogType ());
   }
}
?>
```

Voilà, notre plugin est capable de loguer les URL appelées par nos internautes.

Création du fichier de configuration

Tous les plugins peuvent disposer d'un fichier de configuration.

Dans notre cas, nous allons permettre de configurer dans quel log seront ajoutées les URL.

project/modules/public/stable/standard/default/plugins/mon_plugin/mon_plugin.plugin.conf.php

```
<?php
class PluginConfigMon_Module {
   private $_logType = 'default';
   public function getLogType (){
      return $this->_logType;
   }
}
?>
```

Il ne vous reste maintenant plus qu'à **activer votre plugin**.

Encore plus de bases de données

Déclarer un DAO

Objectifs

Vous savez qu'il est est possible d'**obtenir des DAO automatiquement** depuis la base de données. Toutefois, il existe des cas où vous pouvez préférer déclarer vous même votre DAO pour préciser par exemple les clefs étrangères ou des méthodes supplémentaires.

Supposons la table suivante

```
CREATE TABLE NEWS (
  id_news int(11) NOT NULL AUTO_INCREMENT,
  date_news varchar(8) NULL,
  sujet_news varchar(255) NOT NULL DEFAULT '',
  texte_news text NOT NULL,
  PRIMARY KEY (news_id)
);
```

Et le fichier de DAO correspondant

(à compléter)

Concurrence_d_acces

Exposé du problème

Il arrive souvent dans le web que deux personnes souhaitent modifier une même donnée au même moment, ou bien qu'une personne travaille sur une version qui n'est plus à jour (car modifiée entre temps).

Dans Copix, cette problématique est gérée pour vous de façon transparente et automatique !

Uniquement pour les DAO "déclarées"

Comme vous le savez, Copix est capable de vous fournir des **DAO automatiques** en inspectant la base de données.

Toutefois, la fonctionnalité de concurrence d'accès ne peut pas réellement être détectée, c'est pourquoi vous devrez créer un **fichier de définition** pour votre DAO.

Rassurez vous, rien de compliqué !

Le fichier en question

Pour notre exemple, nous allons simplement inspecter une DAO fournie avec Copix dans le **module copixtest**, en particulier le fichier *project/modules/public/standard/copixtest/resources/copixtestmain.dao.xml*

Ce fichier fait partie du jeu de test que nous lançons régulièrement avant chaque livraison de Copix pour nous assurer que nous vous livrons un produit avec le moins de bugs possible.

Mais allons à la partie qui nous intéresse :

```
<property name="version_test"
        caption="version"
        type="version"
        />
```

Grâce à cet élément, nous indiquons à Copix de se servir du champ "version_test" pour vérifier que nous travaillons bien sur le dernier enregistrement disponible en cas d'update.

Un petit script pour tester la concurrence d'accès

```
$dao = _ioDAO ('copixtest|copixtestmain');
$record = $dao->get ($this->_firstID);
$record2 = clone ($record);//on fait une copie de l'objet

$record->titre_test = 'Titre modifié';
$dao->update ($record);//record est modifié, record2 ne corresponds donc plus à la
          ✂ dernière version de l'enregistrement.

//Nous allons essayer de modifier une nouvelle fois l'élément à partir de cette
          ✂ version non à jour de l'enregistrement, et cela va échouer :-)

try {
  $dao->update ($record2);//record2 n'est pas à jour
}catch (CopixDAOVersionException $e){
  echo "Enregistrement déja modifié, Copix voit tout !";
}
```

Authentification & Droits

Protection de votre page, gestion des droits

Objectifs

Si vous avez suivi l'**exemple précédent**[3], nous allons ici simplement demander à Copix de n'afficher notre page que pour les utilisateurs connectés.

Attention, ce n'est pas facile !

Si vous vous souvenez de notre méthode *processList* pour afficher les nouvelles, et bien nous allons lui rajouter une ligne terrible :

```
//...
function processList (){
   CopixAuth::getCurrentUser ()->assertCredential ('basic:registered');
   //..suite de la fonction
```

Cette ligne demande à **CopixAuth** l'utilisateur courant (**CopixUser**) et lui demande de s'assurer (assertCredential) qu'il dispose bien du droit 'basic:registered' (qui vérifie simplement si l'utilisateur est connecté).

S'il ne dispose pas de ces droits, Copix redirigera l'internaute sur la page d'authentification.

Aller plus loin

Bien évidemment, l'API de **CopixAuth** est plus complète et permet de faire énormément de choses, tout en vous permettant de définir vos propres modèles d'authentification.

Ce tutoriel n'est là que pour vous présenter la facilité d'utilisation du système.

[3] Voir Tutoriel / Avec une base de données / EUGB Encore un gestionnaire de breves

Votre premier gestionnaire d'authentification

Objectifs

L'objectif de ce tutoriel est de vous permettre de créer rapidement un gestionnaire simple d'authentification.

Introduction

Avec la version de 3.0 il est désormais possible de créer vos propres gestionnaires d'authentification et de pouvoir les utiliser pour authentifier vos utilisateurs de diverses manières, de façon uniques ou combinées.

Pour ce tutoriel, nous allons ajouter des éléments au module *auth* (situé dans *project/module/public/standard/auth*)

Si vous avez suivi le tutoriel **Creation d un module** vous pouvez créer votre propre module d'authentification, et les handlers associés.

Création de nos classes

Dans le répertoire classe du module auth, créez un fichier simpleuserhandler.class.php. A l'intérieur de ce fichier déclarez la classe suivante :

```
class SimpleUserHandler implements ICopixUserHandler {
}
```

Cette classe implémente l'interface ICopixHandler qui définit quatre fonctions à coder.

Ajoutons ces quatre fonctions :

```
class SimpleUserHandler implements ICopixUserHandler {

    // Fonction de connexion
    public function login ($pParams){
    }

    // Fonction de déconnexion
    public function logout ($pParams){
    }

    // Chercher un utilisateur
    public function find ($pParams){
    }

    // Récupérer les informations d'un utilisateur
    public function getInformations ($pUserId){
    }
}
```

Maintenant on peut coder les fonctions de login et de logout. Par exemple nous authentifions les utilisateurs dont le login est toto et le mot de passe titi.

```
public function login ($pParams){
 if (isset($pParams['login']) && $pParams['login'] == "toto" && $pParams['pass'] ==
          ✂ "titi") {
  // On retourne vrai et on ajoute les informations sur le handler, l'id et le login
  return new CopixUserLogResponse (true, 'auth|simpleuserhandler', 0, "toto");
 } else {
  return new CopixUserLogResponse (false, null, null, null);
 }
}

public function logout ($pParams){
 return new CopixUserLogResponse (true, null, null, null);
}
```

Modification de la configuration

Enfin il suffit de modifier la configuration (dans /project/config/copix.conf.php) pour prendre en compte notre nouvel handler. Le paramètre indique si la réponse de ce handler est requise ou non pour procéder à l'authentification.

```
$config->copixauth_registerUserHandler (array ('name'=>'auth|simplehandler'
                                               'required'=>false));
```

Votre premier gestionnaire de droits

Objectifs

L'objectif de ce tutoriel est de vous permettre de créer rapidement un gestionnaire de droits simple. Nous vous invitons, si ce n'est déjà fait, de suivre le tutoriel **Votre premier gestionnaire d'authentification**.

Introduction

Pour ce tutoriel, nous allons ajouter des éléments au module *auth* (situé dans *project/module/public/standard/auth*)

Création de nos classes

Dans le répertoire classe du module auth, créez un fichier simplecredentialhandler.class.php. A l'intérieur de ce fichier on crée la classe suivante :

```
class SimpleCredentialHandler implements ICopixCredentialHandler {

}
```

Cette classe implémente l'interface ICopixCredentialHandler qui définit la fonction de vérification des droits, assert.

Ajoutons à notre classe cette fonction

```
class SimpleCredentialHandler implements ICopixCredentialHandler {

  public function assert ($pStringType, $pString, $pUser){
  }

}
```

Pour information : la fonction assert prend trois arguments : - pStringType : Le type de droit (basic ou autre) - pString : La chaine de droit à tester - pUser : L'utilisateur dont on teste les droits.

assert renvoie vrai si la chaine de droit est vérifiée, faux sinon.

Vous pouvez désormais coder la fonction de vérification. Par exemple, pour tous les types de vérifications, on renvoie vrai si le login de l'utilisateur est égal à toto.

```
  public function assert ($pStringType, $pString, $pUser){
    if ($pUser->getLogin() == 'toto') {
      return true;
    } else {
      return false;
    }
  }
```

Modification de la configuration

Enfin il suffit de modifier la configuration pour prendre en compte votre nouveau gestionnaire de droits.

```
$config->copixauth_registerCredentialHandler (array
('name'=>'admin|simplecredentialhandler', 'stopOnSuccess'=>true,
                                          'stopOnFailure'=>false,
                                          'handle'=>'all'));
```

Pour aller plus loin

A partir de cet exemple simple, vous pouvez aller plus loin en changeant le référentiel (par exemple, un annuaire LDAP). N'hésitez pas à nous faire part de vos essais et Happy Copix!

Le cache

Utiliser_le_cache

Introduction

Le système de cache vous permet de conserver les résultats de traitements couteux en temps processeur, et donc de pouvoir les réutiliser en un temps record.

L'objet qui pose problème

Supposez l'objet suivant, avec une méthode complexe qui met un temps fou à calculer son résultat :

```
class ObjetComplexe {
   public function operationComplexe ($num1, $num2){
      sleep (2);
      return $num1+$num2;
   }
}
```

Sans le cache

Vous développez un site traitant des mathématiques, vous avez donc besoin de faire souvent appel à cet objet pour présenter les résultats de vos recherches, par exemple comme suit :

```
class ActionGroupOperationsMathematiques extends CopixActionGroup {
   function processDefault (){
      $ppo = new CopixPPO ();
      $ppo->resultatOperation = _ioClass ('monmodule|ObjetComplexe')->
            operationComplexe (2, 2);
      return _arPpo ($ppo, 'resultatoperation.tpl');
   }
}
```

Faire attendre chaque internaute 2 secondes n'est pas envisageable, d'autant plus si chaque demande de page essouffle le serveur.

C'est là que CopixCache intervient !

Avec le cache

Vous pouvez choisir de faire intervenir CopixCache où bon vous semble. Pour l'exemple, nous allons le placer dans l'action.

```
class ActionGroupOperationsMathematiques extends CopixActionGroup {
   function processDefault (){
      $ppo = new CopixPPO ();
      if (!CopixCache::exists ('operationComplexe')) {     //On regarde si le cache a
         ✂ déjà été stocké
         $ppo->resultatOperation = _ioClass ('monmodule|ObjetComplexe')->
         ✂ operationComplexe (2, 2);
         CopixCache::write ('operationComplexe', $ppo->resultatOperation);   // On
         ✂ stocke le résultat du traitement en cache
      } else {
         $ppo->resultatOperation = CopixCache::read ('operationComplexe');  // Le
         ✂ traitement a été stocké, on récupère le résultat
      }
      return _arPpo ($ppo, 'resultatoperation.tpl');
   }
}
```

La première demande de page mettra ainsi 2 bonnes secondes à s'afficher, alors que la deuxième demande sera quasi instantanée !

Remise à zéro du cache

De temps à autres, vous aurez besoin de remettre le cache à zéro (modification de l'algorithme de calcul, modification effectuée dans les données sources, ...).

Pour ce faire, CopixCache permet la remise à zéro du cache en question avec une simple commande.

```
CopixCache::clear('operationComplexe');
```

Thèmes graphiques

Personnalisez les templates

Introduction

Cet article a pour but de vous présenter très rapidement le système de thèmes graphiques de Copix.

Si vous vous souvenez de l'**arborescence des modules**, vous vous souvenez également qu'il existe un répertoire *templates* dans lequel sont situés tous les modèles d'affichage par défaut pour votre module.

Si vous ne modifiez pas ces modèles dans le thème graphique appliqué sur votre site, alors Copix utilisera ces derniers.

Toutefois, afin que l'application corresponde à vos besoins graphiques, vous avez la possibilité de remplacer ces templates en en **créant de nouveaux**. (pas question de modifier votre module de départ, après tout il est surement parfait ;-))

Exemple pour modifier le template principal dans le thème par défaut

Le nom du template principal livré avec la distribution de Copix est le fichier *main.ptpl*.

Par défaut, comme vous n'utilisez aucun système de thème, alors Copix va utiliser le fichier à l'emplacement *modules/public/standard/default/templates/main.ptpl*.

Toutefois, si vous créez un fichier template dans *project/themes/default/default/main.ptpl*, l'utilisation de ce dernier sera préféré au modèle par défaut.

Quelques explications

project/themes correspond au chemin où sont stockés tous les modèles pour les thèmes graphiques.

- Le premier "default" correspond au thème par défaut appliqué pour une application.
- Le second "default" correspond au nom du module pour qui nous souhaitons modifier le jeu de template.
- Enfin main.ptpl correspond au nom du modèle que nous voulons remplacer.

Ainsi, si vous voulez modifier le template "a_modifier.tpl" du module "news", vous créerez le fichier *project/themes/default/news/a_modifier.tpl*.

Voir aussi

- Création d'un thème graphique
- Personnalisez les ressources

Création d'un thème graphique

Thèmes graphiques ?

Copix vous permet de créer des thèmes graphiques que vous pouvez sélectionner à loisir pour votre site. Ces thèmes sont composés de modèles d'affichages (les templates) et de ressources (images, fichiers javascript, ...) dont nous reparlerons dans un **autre tutoriel**.

Création du thème, theme.xml

Pour créer un thème, il vous suffit de créer un répertoire dans le répertoire de themes (*project/themes*).

Par exemple pour créer un thème test, vous devez créer un répertoire *test* dans le répertoire *project/themes*.

Ensuite, pour indiquer à Copix que ce répertoire contient effectivement un thème, il vous faut créer un fichier donnant quelques détails sur votre oeuvre en créant le fichier "theme.xml" dans *project/themes/test/theme.xml*.

Voici un exemple du code qui doit composer ce fichier :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<themedefinition>
        <name>test</name>
        <author>Un artiste !</author>
        <description>Thème éblouissant</description>
        <image>screenshot.png</image>
</themedefinition>
```

- La balise "name" contient le nom de votre thème. Elle est la seule balise obligatoire.
- La balise "author" vous permet de préciser l'auteur de ce thème.
- La balise "description" donne une description plus précise du thème.
- La balise "image" permet de préciser une image d'aperçu de votre. Cette image devra se trouver à la racine de votre répertoire de thème.

Quand tout cela est fait, il ne vous reste plus qu'à créer vos répertoires correspondants aux modules que vous voulez surcharger, et d'y mettre les templates surchargés.

Personnaliser_les_ressources

Introduction

Copix, possède un système vous permettant de gérer plus facilement vos ressources (css, image, js ...). Ce système est simple, quand vous voulez afficher une image, linker un fichier css, etc... vous ne mettez jamais son chemin en dur mais vous utilisez Copix qui va générer le chemin voulu en fonction de vos thèmes.

Un exemple

Pour intégrer une image dans votre template principal, utilisez le code suivant :

en php :

```
<img src="<?php echo CopixUrl::getResource ("img/mon_image.jpg"); ?>" />
```

en smarty :

```
<img src="{copixresource path="img/mon_image.jpg"}" />
```

L'appel aux ressources de cette manière aura l'effet suivant :

- Si vous avez activé le thème par défaut, les fichiers utilisés seront ceux situés dans www/themes/default, ainsi, dans notre exemple, nous utiliserons www/themes/default/img/mon_image.jpg
- Si vous avez sélectionné un thème "votre_theme", Copix cherchera en premier lieu les ressources dans www/themes/votre_theme. Dans notre exemple, il chercherait l'image dans www/themes/votre_theme/img/mon_image.jpg
- Enfin, si Copix ne trouve pas la ressource demandée ni dans le thème courant, ni dans le thème default, alors le chemin utilisé sera la racine de votre serveur web. Ainsi, dans notre exemple www/img/mon_image.jpg.

Conclusion

Ce système de ressources vous permet d'utiliser des ressources css, js, images et autres qui s'adapteront à vos envies créatrices, et ce sans nécessairement changer de template.

Les classiques

Comment faire du CRUD (Create, Read, Update, Delete) avec Copix ?

Présentation

Cette page vous présente comment, avec Copix, faire un module de type CRUD (Create Read Update Delete).

Cette documentation est absolument exhaustive : aucune opération ou ligne de code supplémentaire n'est nécessaire pour arriver à bout du module.

Si vous avez téléchargé une version complète de Copix, ce module se situe dans le répertoire project/modules/public/stable/tutorials/crud/*.*

C'est parti !

Table utilisée

Voici la table que nous allons utiliser. Nous considérons que vous utiliserez MySQL pour ce tutoriel.

```
CREATE TABLE `tutorial_crud` (
`id_crud` INT NOT NULL AUTO_INCREMENT PRIMARY KEY ,
`caption_crud` VARCHAR( 20 ) NOT NULL ,
`description_crud` TEXT NOT NULL
);
```

Première étape, Module.xml

Comme toujours, vous devez créer un fichier **module.xml** pour déclarer votre module dans Copix.

```
<moduledefinition>
 <general>
  <default name="crud" longdescription="Tutorial CRUD (Create Read Update Delete) avec
            ✂ Copix." />
 </general>
</moduledefinition>
```

Une fois ce fichier crée, installez votre module grâce au module admin.

Deuxième étape, page de liste des éléments

Créez un **ActionGroup** par défaut pour votre module, actiongroup/default.actiongroup.php. L'**action** par défaut aura pour objectif d'afficher la liste des éléments de la table.

```
class ActionGroupDefault extends CopixActionGroup {
 public function processDefault (){
  $ppo = new CopixPpo ();

  $ppo->TITLE_PAGE = 'Liste des éléments';
  $ppo->arData = _ioDAO ('tutorial_crud')->findAll ();//Le DAO est généré
            ✂ automatiquement

  return _arPpo ($ppo, 'crud.list.tpl');
}
```

Cette action utilise le système de **DAO** de Copix et profite de la possibilité qu'offre Copix de générer automatiquement les DAO à partir du nom des tables.

Cette action demande un affichage via le code retour **PPO** dans le template crud.list.tpl.

Créez donc le fichier templates/crud.list.tpl

```
<table class="CopixTable">
 <thead>
  <tr>
   <th>Libellé</th>
   <th>Actions</th>
  </tr>
 </thead>
 <tbody>
 {foreach from=$ppo->arData item=element}
  <tr {cycle values=',class="alternate"'}>
   <td>{$element->caption_crud}</td>
   <td>
     <a href="{copixurl dest="delete" id_crud=$element->id_crud}"><img
            ✂ src="{copixresource path="img/tools/delete.png"}" /></a>
     <a href="{copixurl dest="edit" id_crud=$element->id_crud}"><img
            ✂ src="{copixresource path="img/tools/update.png"}" /></a>
   <td>
  </tr>
 {/foreach}
 </tbody>
</table>

<a href="{copixurl dest="edit" new=1}"><img src="{copixresource
            ✂ path="img/tools/new.png"}" />Nouveau</a>
```

- Ce template profite du tag Smarty alternate afin d'alterner la couleur des lignes pour plus de lisibilité.
- On utilise également le système d'**url** afin d'assurer à notre application une portabilité complète et pouvoir profiter plus tard de la possibilité de **réécriture d'URL**.
- On utilise le système de **ressources** afin de pouvoir profiter des **thèmes graphiques** Copix.

Troisième étape, page de modification

Dans votre ActionGroup, ajoutez une action de modification. Toutes les lignes importantes du code suivant sont documentées.

```
public function processEdit (){
 //Si un identifiant d'élément à modifier est donné et que ce dernier existe, on le
          ✂ place
 //dans la session pour qu'il soit défini comme élément à modifier
 if ($crud_id = CopixRequest::getInt ('id_crud')){
  if ($toEdit = _ioDAO ('tutorial_crud')->get ($crud_id)){
   CopixSession::set ('crud|edit', $toEdit);
  }
 }
 //On regarde s'il existe un élément en cours de modification, si ce n'est pas le cas
          ✂ on
 //passe en mode création. Création forcée si demandé dans l'url.
 if ((($toEdit = CopixSession::get ('crud|edit')) === null) || (_request ('new')
          ✂ == 1)){
  CopixSession::set ('crud|edit', $toEdit = _record ('tutorial_crud'));
 }

 //Préparation de la page
 $ppo = new CopixPpo ();
 $ppo->TITLE_PAGE = $toEdit->id_crud ? "Modification de l'élément" : "Création d'un
          ✂ élément";
 $ppo->toEdit = $toEdit;

 //si on demande à afficher les messages d'erreurs
 $ppo->arErrors = _request ('errors') ? _ioDAO ('tutorial_crud')->check ($toEdit) :
          ✂ array ();
 return _arPpo ($ppo, 'crud.form.tpl');
}
```

- Cette page regarde donc si l'utilisateur a demandé à éditer un élément (en spécifiant son identifiant dans la **requête**).
- Si c'est le cas, on utilise le système de DAO pour le récupérer et le mettre en **session**.
- Si l'on a demandé à afficher les erreurs sur la page, on utilise la **méthode check** pour récupérer un tableau d'erreurs concernant l'enregistrement en cours de modification.
- On demande ensuite à afficher les éléments dans le template crud.form.tpl.

```
{if count ($ppo->arErrors)}
<div class="errorMessage">
<h1>Erreurs</h1>
{ulli values=$ppo->arErrors}
</div>
{/if}

<form action="{copixurl dest="valid"}" method="POST">
<table class="CopixVerticalTable">
 <tr>
  <th>Libellé</th>
  <td><input type="text" name="caption_crud" value="{$ppo->toEdit->
            ✂ caption_crud|escape}" /></td>
 </tr>

 <tr>
  <th>Description</th>
  <td><textarea name="description_crud">{$ppo->toEdit->description_crud|escape}
</textarea></td>
 </tr>

</table>

<input type="submit" value="Valider" />
<a href="{copixurl dest="|" cancel=1}"><input type="button" value="Annuler" /></a>
</form>
```

- Ce template utilise le tag ulli pour générer une liste d'erreurs
- Les données de formulaire sont toujours transmises par la méthode POST
- On utilise le modificateur escape pour échapper les caractères spéciaux HTML que l'utilisateur pourrait saisir.

Quatrième étape, enregistrer les changements

Dans votre ActionGroup, il est temps de développer la méthode de validation, ici commentée :

```
public function processValid (){
 //On vérifie que l'on est bien en train de modifier un élément, sinon on retourne
          ✂ à la liste.
 if (($toEdit = CopixSession::get ('crud|edit')) === null){
  return _arRedirect (_url ('|'));
 }

 //validation des modifications depuis le formulaire
 $toEdit->caption_crud = _request ('caption_crud');
 $toEdit->description_crud = _request ('description_crud');
 CopixSession::set ('crud|edit', $toEdit);

 //Si tout va bien, on sauvegarde
 if (_ioDAO ('tutorial_crud')->check ($toEdit) === true){
  if ($toEdit->id_crud){
   _ioDAO ('tutorial_crud')->update ($toEdit);
  }else{
   _ioDAO ('tutorial_crud')->insert ($toEdit);
  }

  //on vide la session
  CopixSession::set ('crud|edit', null);
  return _arRedirect (_url ('|'));
 }else{
  //un problème ? on retourne sur la page de modification en indiquant qu'il
          ✂ existe un problème
  return _arRedirect (_url ('edit', array ('errors'=>1)));
 }
}
```

- Cette méthode récupère depuis la session l'élément en cours de modification.
- En utilisant les données transmises dans la requête, on met à jour l'enregistrement.
- Si la méthode check indique que tout est correct, on utilise les méthodes ou **insert** (pour ajouter l'enregistrement) ou **update** (s'il s'agit d'une mise à jour).
- S'il y a un problème, l'utilisateur est **redirigé** vers la page de modification avec les messages d'erreurs, sinon on redirige l'utilisateur dans la page de liste.

Cinquième étape, permettre la suppression

Pour permettre la suppression, on rajoute l'action suivante dans l'ActionGroup :

```
 public function processDelete (){
  //On s'assure qu'un id_crud a bien été donné
  CopixRequest::assert ('id_crud');

  //On vérifie si l'enregistrement existe, si tel n'est pas le cas,
  //on ne s'embête pas et on retourne en liste.
  if (! ($record = _ioDAO ('tutorial_crud')->get (CopixRequest::getInt ('id_crud')))){
   return _arRedirect (_url ('|'));//'|' signifie action par défaut dans l'actiongroup
par défaut du module courant, on aurait pu ici écrire 'crud|default|default' ou
'crud||'
  }

  if (! _request ('confirm', false, true)){
   return CopixActionGroup::process ('generictools|Messages::getConfirm',
   array ('message'=>'Etes vous sûr de vouloir supprimer cet élément '.$record-
>caption_crud.' ?',
    'confirm'=>_url ('delete', array ('id_crud'=>$record->id_crud, 'confirm'=>1)),
    'cancel'=>_url ('|')));
  }else{
   _ioDAO ('tutorial_crud')->delete ($record->id_crud);
  }
  return _arRedirect (_url ('|'));
}
```

Dans cette action, on s'assure que l'utilisateur est bien passé par une page de confirmation réalisée avec le module GenericTools avant d'effectivement supprimer la donnée avec la méthode delete.

C'est terminé

Dans ce tutoriel, vous pouvez apprécier le nombre de choses que Copix vous a proposé pour aller plus vite dans vos développements, à savoir :

- Les DAO automatiques pour lire, supprimer, modifier, insérer et contrôler vos données.
- Un module pour générer automatiquement les page de confirmation.
- Un système de Session pour y placer des objets facilement (sans se soucier de leur inclusion préalable).
- Des tags pour réaliser les opérations communes en une ligne dans vos templates.
- Un système de génération d'URL pour faire évoluer la forme de vos URL sans impact sur le code existant
- Un système de thèmes graphiques qui vous permettra de changer l'apparence de votre site sans modifier le code du module.