

# Tutoriel principal Jelix 1.0

par [Contributeurs au wiki de jelix.org](#)

Date de publication : 30 août 2007

Dernière mise à jour : 16 mai 2008

Le but de ce tutoriel va être de développer une petite application de gestion de news.

Ce tutoriel est issu de [jelix.org](#) :  **Tutoriel Principal pour Jelix 1.0**

I - Avant-propos.....	3
II - Création de l'application et d'un module.....	4
II-A - Découverte de Jelix-Scripts.....	4
II-B - Création d'une application.....	5
II-C - Création d'un module.....	5
III - Création d'une première action simple.....	7
III-A - Un peu de théorie.....	7
III-B - Implémentation d'une action.....	8
III-A-1 - Objet réponse.....	9
III-A-2 - Le template.....	9
III-A-3 - Utilisation du template dans l'action.....	9
III-C - Premier affichage.....	10
III-B-1 - Problème d'affichage des caractères accentués.....	10
IV - Configuration de la base de données.....	12
IV-A - Fichier de configuration d'accès.....	12
IV-B - Création de la table.....	13
V - Utilisation d'un DAO.....	14
V-A - Créer un premier DAO.....	14
V-B - Lister les news.....	15
VI - Création d'un formulaire.....	17
VI-A - Le template et les URLs.....	17
VI-B - Affichage du formulaire.....	18
VI-C - Sauvegarde des données.....	18
VII - Création d'un CRUD.....	20
VII-A - Création du formulaire jForms.....	20
VII-B - Création du contrôleur CRUD.....	21
VII-B-1 - Note.....	22
VIII - Fin.....	23

## I - Avant-propos

Le but de ce tutoriel va être de développer une petite application de gestion de news. Il est possible de télécharger de la forge de Jelix le code de ce tutoriel aux formats **zip** ou **gzip**.

## II - Création de l'application et d'un module

Nous allons créer une application à partir de rien. Toute application Jelix a un nom : celui de son répertoire. Nous allons appeler la notre #actu.org#.

On considère que vous avez installé Jelix (**Édition Developer**) comme indiqué sur la page d'**installation** et que vous utilisez la configuration par défaut sans avoir modifié son arborescence. Il faut avoir aussi installé PHP et PHP-cli comme il est précisé dans cette même page, pour utiliser le script *jelix.php*.

### II-A - Découverte de Jelix-Scripts

Jelix (édition *developer*) est fourni avec un script, *jelix.php*, qui facilite la création et la modification des différents fichiers d'une application basée sur Jelix : il est possible de créer les répertoires et les fichiers de base à la main mais *jelix.php* le fait pour vous ! Il faut l'invoquer avec la version ligne de commande de PHP et indiquer en paramètre un nom de commande Jelix, ainsi que d'éventuels paramètres et options :

```
php jelix.php [--NOM_APPLICATION] nom_commande [options] [paramètres]
```

Pour ce faire, ouvrez une console et allez dans le répertoire *lib/jelix-scripts/* où est situé *jelix.php*.

```
cd lib/jelix-scripts/          # sous linux
cd lib\jelix-scripts\         # sous windows
```

Pour avoir l'aide sur toutes les commandes disponibles, tapez :

```
php jelix.php help
```

Vous aurez remarqué qu'il faut toujours indiquer à *jelix.php* (sauf pour la commande *help*), le nom de l'application sur laquelle la commande s'applique. Pour notre exemple, il faudrait taper ceci :

```
php jelix.php --actu.org nom_commande [options] [paramètres]
```

Il est possible d'éviter cela en stockant le nom de l'application dans la variable d'environnement **JELIX\_APP\_NAME**.

Pour notre exemple faites donc ceci :

```
export JELIX_APP_NAME="actu.org"    # sous linux
set JELIX_APP_NAME=actu.org        # sous windows
```

Le lancement de ce script se résumera alors à :

```
php jelix.php nom_commande [options] [paramètres]
```

Notez enfin que vous pouvez lancer ce script depuis n'importe quel répertoire. Si vous savez par exemple que vous n'allez lancer *jelix.php* qu'une seule fois, pas besoin d'aller dans le répertoire du script, ni d'utiliser la variable d'environnement. Il suffit de taper :

```
php chemin_jelix/lib/jelix-scripts/jelix.php --actu.org help # sous linux
```

```
php chemin_jelix\lib\jelix-scripts\jelix.php --actu.org help # sous windows
```

## II-B - Création d'une application

Commençons par créer notre application. Jelix propose une commande pour créer toute l'arborescence d'une application : **createapp**. Tapez :

```
php jelix.php createapp -nodefaultmodule
```

Vous obtenez alors un répertoire *actu.org/*, au même niveau que le répertoire *lib/*. Son contenu est le suivant :

```
actu.org/  
  application.init.php  
  modules/          les modules propres à votre application  
  plugins/          les plugins propres à votre application  
  project.xml  
  responses/  
  var/  
    config/         les fichiers de configuration de votre application  
    log/            les éventuels fichiers journaux  
    themes/         les différents thèmes possibles dans votre application  
    overloads/      contiendra les différents fichiers que vous aurez redéfinis, issus de modules tiers.  
  www/             la racine du site
```

Vérifiez que le répertoire *temp/actu.org* nouvellement créé a les permissions en écriture pour le serveur web.

## II-C - Création d'un module

Notez que nous avons utilisé l'option *-nodefaultmodule* pour la commande *createapp*. Sans cette option, la commande crée aussi un module par défaut du même nom que l'appli (donc *actu.org/modules/actu.org*). Mais dans le cadre de ce tutoriel, nous voulons créer nous-même le module.

Donc maintenant que nous avons un squelette d'application, il va falloir créer un module, car pour le moment, votre application ne peut rien faire puisqu'il n'y a aucune action définie. En effet, il va falloir déclarer et implémenter des actions. Une action peut être l'affichage d'une page, la sauvegarde d'un formulaire, un appel de service web, etc#

Les actions sont regroupées dans des modules distincts selon le domaine fonctionnel auquel elles sont rattachées. Nous allons créer par exemple un module qui va regrouper les actions pour afficher et gérer des news. Pour ce faire, il existe la commande **createmodule**, qui prend en paramètre le nom du module à créer.

```
php jelix.php createmodule news
```

En tapant cette commande, Jelix vous a créé un module nommé news avec toute son arborescence et quelques fichiers indispensables. Dans *actu.org/modules/news/*, vous trouverez donc :

```
classes/          vos classes métiers et services  
  controllers/    les classes de traitement des actions  
    default.classic.php  un contrôleur  
  daos/          les fichiers de mapping relationnel-objet  
  forms/         fichiers de description des formulaires  
  locales/       fichiers de langues (fichiers "properties")  
    en_EN/  
    fr_FR/  
  module.xml     fichier décrivant l'identité du module
```

templates/ zones/	templates du module objets traitant des zones spécifiques dans une page
----------------------	--

Nous sommes maintenant prêts à définir des actions.

## III - Création d'une première action simple

### III-A - Un peu de théorie

Une action est un élément fondamental du framework. Tout affichage, tout traitement de formulaire, tout appel de service web est une action.

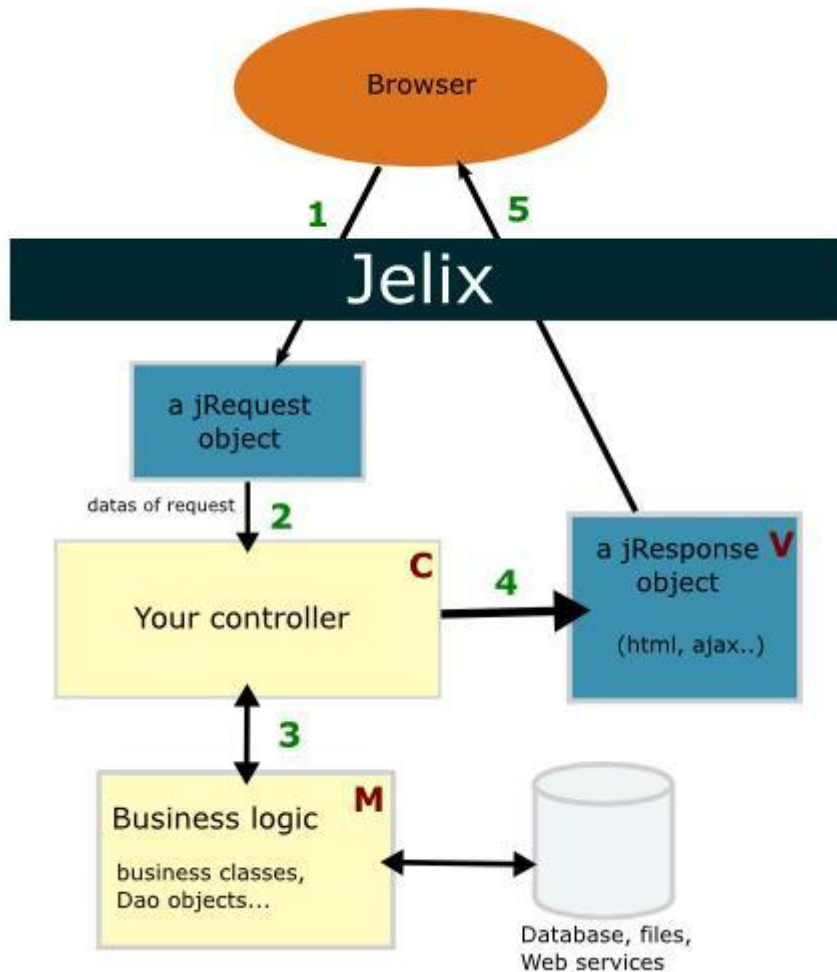
Une action est appelée dans le cadre d'une requête d'un type précis et génère une réponse spécifique dans un format spécifique, qui peut être lié au type de requête en question.

Il existe plusieurs types de requêtes, notamment le type que l'on nomme dans Jelix, `#classic#`, pour lequel une action peut fournir une réponse dans un format quelconque : HTML, XML, etc. C'est pour ce type de requête que vous définirez le plus souvent des actions. En général, ce type de requête fournit ses paramètres dans l'url ou dans le corps de la requête HTTP (méthode POST).

Vous avez aussi des types de requêtes plus spécifiques, comme par exemple le type `xmlrpc` (utilisée dans le cadre d'un service web). En XML-RPC, les données en entrée ne sont pas des paramètres dans une url, mais sont stockées dans un contenu XML. Comme le veut le protocole XML-RPC, une action définie pour ce type de requête doit fournir obligatoirement une réponse au format XML-RPC.

En connaissant le type de requête traitée et l'action, Jelix connaît ainsi le type de la réponse à générer, et donc contrôle plus ou moins la génération de la réponse. Ainsi, même en cas d'erreur (une exception ou autre) survenant pendant le traitement de l'action, le format de sortie sera toujours celui attendu. Un client qui appelle un service web en XML-RPC, aura donc quoi qu'il arrive une réponse au format XML-RPC. Cela apporte une certaine robustesse à l'application.

Voici un schéma simplifié du déroulement d'une action :



- 1 Jelix reçoit une requête HTTP. Il instancie un objet jRequest qui contient les données de la requête, et instancie le contrôleur qui correspond à l'action.
- 2 La méthode du contrôleur correspondante à l'action est exécutée. La méthode récupère les paramètres de requête pour déterminer les traitements à suivre.
- 3 Le contrôleur exécute les traitements métiers et récupère éventuellement des résultats qui seront utilisés pour l'affichage
- 4 Le contrôleur instancie un objet de type jResponse auquel il assignera les données à afficher, initialisera les templates etc.
- 5 Jelix récupère cet objet jResponse, invoque la génération du document en sortie et envoi ce dernier au navigateur.

### III-B - Implémentation d'une action

Les actions sont implémentées dans ce qu'on appelle des contrôleurs. Les contrôleurs sont des classes contenant des méthodes pour chaque action. Les contrôleurs sont stockés dans des fichiers `controllers/nom_controleur.type_requete.php`.

En général il y a une méthode `index()` pour l'action par défaut.

Modifions donc cette action par défaut. Pour cela ouvrons le fichier `controllers/default.classic.php`. Vous devez avoir ce contenu :

```
class defaultCtrl extends jController {
    function index () {
```

```
$rep = $this->getResponse('html');  
  
return $rep;  
}  
}
```

Vous remarquerez qu'il y a certaines conventions de nommage. Les classes contrôleurs ont un nom suffixé par #Ctrl#. Ce qui précède le suffixe, #default#, est le nom du contrôleur, que vous indiquerez dans le paramètre action, et c'est aussi le préfixe du nom du fichier \*.classic.php.

### III-A-1 - Objet réponse

Dans la méthode #index()# du contrôleur, on récupère dans la variable \$rep une réponse de type #html#. Vous obtenez en fait un objet de classe jResponseHtml, dérivant de la classe jResponse.

Vous verrez plus tard qu'il existe d'autres types de réponses et que vous pouvez développer vos propres objets réponse.

L'objet jResponseHtml s'occupe de générer une réponse en HTML (donc une page en HTML). Il génère automatiquement la partie <head> du HTML, à partir de certaines de ses propriétés. Par exemple spécifions le titre de la page :

```
$rep->title = 'Dernières actualités';
```

Et le navigateur recevra :

```
<head>  
  <title>Dernières actualités</title>  
</head>
```

Tout le corps de la page, c'est à dire le contenu de la balise html <body>, doit être généré par vous même, via éventuellement le moteur de template de Jelix : **JTpl**. jResponseHtml instancie en standard un moteur de template placé dans la propriété body. Le nom du fichier template est à placer dans la propriété bodyTpl. Avant de voir le code voyons d'abord le contenu du template.

### III-A-2 - Le template

Créez un fichier listenews.tpl dans le répertoire *templates* du module. Et mettez y ce contenu :

```
<h2>Dernières actualités</h2>  
<p>Ouverture prochaine de cette rubrique.</p>
```

Comme il a été dit auparavant, le contenu du template sera le contenu de la balise <body>. C'est pourquoi vous n'avez pas à mettre les balises standard <html>, <head>, etc# Juste le contenu de la page.

### III-A-3 - Utilisation du template dans l'action

Voyons maintenant comment cela se concrétise dans le contrôleur :

```
class defaultCtrl extends jController {
```

```
function index () {  
    $rep = $this->getResponse('html');  
    $rep->title = 'Dernières actualités';  
    $rep->bodyTpl = 'listenews';  
    return $rep;  
}
```

On a donc rajouté une instruction pour indiquer à l'objet réponse que l'on utilise le template **listenews.tpl**.

Il n'est pas besoin de mettre le suffixe `#.tpl#` du nom du fichier, car il s'agit en fait d'un sélecteur Jelix. Un sélecteur est une chaîne, permettant de désigner facilement une ressource du projet, indépendamment de son emplacement physique.

Un sélecteur comporte un nom de module et un nom de ressource séparés par le caractère `#~#`, comme ceci : **nom\_module~nom\_resource**. La partie **#nom\_module~#** est facultative quand il s'agit du module courant. Le nom de la ressource n'est pas forcément un nom de fichier, même si la plupart du temps elle désigne un fichier. L'objet qui utilise le sélecteur (ici `jTpl`) sait comment récupérer le fichier correspondant au sélecteur. Vous verrez que les sélecteurs sont abondamment utilisés, et permettent une certaine souplesse et une indépendance vis à vis des chemins de fichiers physiques.

### III-C - Premier affichage

Nous sommes maintenant prêt à afficher la première version de notre action. Pour cela, tapez l'url suivante dans votre navigateur :

```
http://localhost/jelix/actu.org/www/index.php?module=news&action=default:index
```

Vous verrez alors s'afficher le contenu du template que l'on vient de créer.

Le paramètre **module** indique le nom du module. Le paramètre **action** est le nom de l'action à exécuter dans ce module. Ce nom est constitué de deux parties, séparées par deux points `#:#` (pour les utilisateurs de jelix 1.0beta3.1 ou précédent, il faut utiliser le caractère `##_#`). La première partie est le nom du contrôleur, la deuxième le nom de la méthode à exécuter. Si il n'y a pas de `#:#`, jelix considère qu'il s'agit du nom de la méthode dans le contrôleur de nom `#default#`.

On peut indiquer que cette action sera l'action par défaut de l'application. Pour cela, ouvrez le fichier de configuration `actu.org/var/config/index/config.ini.php` et indiquez-la :

```
startModule = "news"  
startAction = "default:index"
```

Pour afficher notre première page, vous pouvez alors utiliser simplement l'url :

```
http://localhost/jelix/actu.org/www/index.php
```

### III-B-1 - Problème d'affichage des caractères accentués

Si les caractères accentués s'affichent mal dans votre navigateur, c'est que l'édition de vos fichiers ne s'est pas faite avec le même encodage que celui indiqué dans la configuration de Jelix (qui est par défaut UTF-8). Donc :

**Donc soit :**

- Vous modifiez la configuration de votre éditeur préféré pour qu'il édite en UTF-8, et il faut alors convertir vos scripts existants pour UTF-8 (ou les réécrire) ;
- Vous modifiez le fichier `var/config/defaultconfig.ini.php` en changeant la propriété `charset` (en mettant ISO-8859-1 par exemple).

## IV - Configuration de la base de données

Avant de continuer plus loin avec du code, il nous faut configurer Jelix pour pouvoir accéder à une base de données et alimenter cette base un minimum. Nous allons en effet stocker nos news dans une table.

Jelix prend en charge MySQL, PostgreSQL, SQLite et PDO. Par son système de driver, il est possible d'ajouter d'autres types de bases de données (pour le cas où on ne veuille pas utiliser PDO).

### IV-A - Fichier de configuration d'accès

L'accès à une base de données est configuré dans le fichier **actu.org/var/config/dbprofils.ini.php**. Dans ce fichier, on peut spécifier un ou plusieurs profils de connexion, donc une ou plusieurs connexions.

Chaque profil a un nom et est spécifié comme suit :

```
[NomProfil]
driver="mysql" ; nom du driver à utiliser
database="foo" ; nom de la base à utiliser
host= "localhost" ; nom de la machine du serveur de base de données
user= "john" ; utilisateur de connexion
password="doo" ; mot de passe
persistent= on ; indique si la connexion est persistante
; à activer si vous avez des problèmes de caractères bizarres avec vos données :
; force_encoding = on
```

Mis à part le paramètre driver qui est obligatoire, le reste des paramètres dépend du driver utilisé. En général, vous aurez toutefois les paramètres *database*, *host*, *user* et *password*. Changez le contenu de ce fichier en fonction de votre configuration. Nous prendrons comme nom de profil #actu.org# par exemple, et nous allons définir ce profil par défaut en l'indiquant avec le paramètre #default#.

```
default = actu.org

[actu.org]
driver="mysql"
database="actu"
host= "localhost"
user= "actu"
password="actu"
persistent= on
; à activer si vous avez des problèmes de caractères bizarres avec vos données :
; force_encoding = on
```

Note : si dans votre application, vous voulez créer plusieurs points d'entrée qui utilisent des fichiers de profils de base de données différents, vous pouvez, dans les fichiers de configuration des points d'entrée, indiquer les fichiers de profils au niveau de l'option *dbProfils*. Par exemple dans *actu.org/var/config/index/config.ini.php*, vous pouvez indiquer :

```
dbProfils = dbprofils.ini.php
```

Mais si vous n'avez qu'un fichier de profils, pas besoin de renseigner cette option.

## IV-B - Création de la table

Dans la base #actu#, nous allons maintenant créer une table news. Exécutez ce script SQL (adaptez-le si vous utilisez une autre base que MySQL) dans phpmyadmin par exemple :

```
CREATE TABLE `news` (  
  `id_news` INT NOT NULL AUTO_INCREMENT ,  
  `sujet` VARCHAR( 255 ) NOT NULL ,  
  `texte` TEXT NOT NULL ,  
  `news_date` DATE NOT NULL ,  
  PRIMARY KEY ( `id_news` )  
);
```

Et alimentons cette table avec deux news (nous ferons un formulaire plus tard).

```
INSERT INTO `news` VALUES (1, 'première news',  
'Ceci est une première news. In commodo, neque sit amet laoreet accumsan,  
neque velit rutrum augue, a fringilla nibh lorem nec est. Cras eleifend eros.  
Sed vehicula. Donec vel enim at nunc tincidunt pellentesque. Donec malesuada.  
Praesent volutpat orci ut leo. Donec dictum tortor quis odio. Aliquam pulvinar  
justo eu eros.',  
'2006-01-15');  
  
INSERT INTO `news` VALUES (2, 'Lorem Ipsum',  
'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse enim  
elit, luctus in, rhoncus quis, facilisis in, nulla. Nam eu dolor vel erat semper  
porta. Phasellus pellentesque nulla a urna. Phasellus nonummy diam id risus.  
Donec faucibus mi sed nisi. Sed et lectus at ligula scelerisque tempus. Proin  
justo nibh, consectetur porta, accumsan ac, consectetur id, dui. Morbi at  
mi auctor urna elementum convallis. Etiam et massa porta risus imperdiet  
ullamcorper. Aenean a metus at tortor ultrices accumsan. Mauris luctus.',  
'0000-00-00');
```

Nous sommes maintenant prêts à utiliser ces données dans notre application.

## V - Utilisation d'un DAO

Jelix propose un système de mapping relationnel objet, jDao, basé sur le pattern DAO.

Le pattern DAO se base sur deux types d'objets : un objet #record#, contenant des données (représentant un enregistrement), et un objet #factory#, qui permet de récupérer des listes de records, ou de créer, sauver, effacer des records.

Concrètement avec jDao, un fichier XML DAO vous permet de définir un record et une factory, qui agiront sur une ou plusieurs tables en même temps. Vous y définissez donc le mapping : quel champs de la table ira dans quelle propriété du record, ainsi que le type de donnée, les clés, sur quelles tables s'effectue le mapping, selon quelles jointures, etc#

À partir de ce fichier, jDao génère à la volée deux classes basées respectivement sur jDaoRecordBase et jDaoFactoryBase (qui sont stockées dans un fichier PHP dans le cache de Jelix), dans lesquelles sont fournies toutes les méthodes et requêtes SQL principales **en dur**. En effet, contrairement à d'autres systèmes de mapping, les requêtes SQL sont donc générées une seule fois, et non pas dynamiquement à chaque appel de pages. Cela permet de meilleures performances.

Dans le fichier XML de DAO, vous pouvez aussi définir vos propres méthodes d'accès aux données, et jDao générera les méthodes et requêtes correspondantes dans la factory du DAO.

### V-A - Créer un premier DAO

Vous disposez d'une commande pour créer un fichier DAO, basé sur une table existante. Elle a la syntaxe suivante :

```
createdao //nom_module// //nom_dao// //nom_table//
```

Nous avons précédemment créé une table news, et nous allons créer un DAO appelé #news#, dans le module #news#. Tapez donc :

```
php jelix.php createdao news news news
```

Vous obtenez alors un fichier **actu.org/modules/news/daos/news.dao.xml**. (vous pouvez bien sûr aussi le créer à la main).

Son contenu est le suivant :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<dao xmlns="http://jelix.org/ns/dao/1.0">
  <datasources>
    <primarytable name="news" realname="news" primarykey="id_news" />
  </datasources>
  <record>
    <property name="id_news" fieldname="id_news" datatype="autoincrement"/>
    <property name="sujet" fieldname="sujet" datatype="string"/>
    <property name="texte" fieldname="texte" datatype="string"/>
    <property name="news_date" fieldname="news_date" datatype="date"/>
  </record>
</dao>
```

C'est un contenu très simple, et bien sûr il existe d'autres attributs et balises pour l'enrichir et le personnaliser. Pour l'instant, nous allons nous en tenir là.

## V-B - Lister les news

Nous allons maintenant utiliser ce DAO pour récupérer la liste des news. Nous allons donc demander à jDao la factory de ce DAO, et appeler sa méthode **findAll** prédéfinie. Pour ce faire, on fait appel à jDao::get() :

```
$fact = jDao::get('news~news');
$liste = $fact->findAll();
```

En paramètre de jDao::get(), on donne le sélecteur du DAO (qui a pour nom #news#, et dans le module news) : #news~news#. Elle nous renvoie la factory de ce DAO. et en appelant la méthode *findAll*, on récupère toute la liste des records.

En fait, il ne s'agit pas vraiment d'une liste, mais d'un objet `jdbcResultSet`, qui est un itérateur sur les résultats de la requête correspondante.

Intégrons ça dans notre contrôleur :

```
function index() {
    $rep = $this->getResponse('html');
    $rep->title = 'Dernières actualités';
    $rep->bodyTpl = 'listenews';

    $fact = jDao::get('news~news');
    $liste = $fact->findAll();

    $rep->body->assign('liste', $liste);

    return $rep;
}
```

Comme il a été dit précédemment, la propriété *body* de `jResponseHtml` est un objet `jTpl` (moteur de template). On lui donne la liste dans une variable de template #liste#, pour pouvoir l'utiliser dans le template. Celui-ci devient donc :

```
<h2>Dernières actualités</h2>

<table>
{foreach $liste as $news}
<tr>
    <td>{$news->sujet}</td><td>{$news->news_date}</td>
</tr>
{/foreach}
</table>
```

Vous découvrez ici le tag de template {foreach}, qui fonctionne exactement comme le foreach php. Pour afficher des valeurs, il suffit de mettre le nom de la variable de template précédé par un \$, et encadré par des accolades.

La liste renvoyée par *findAll* est une liste d'objets record, tels qu'ils sont définis dans le fichier XML. Ils ont donc les propriétés *id\_news*, *sujet*, *texte* et *news\_date*.

Affichez à nouveau la page avec :

```
http://localhost/jelix/actu.org/www/index.php?module=news&action=default:index
```

Vous devriez voir la liste des news.

Remarque : si vous avez des problèmes d'affichage des accents, essayez de mettre `force_encoding=on` dans le fichier `dbprofils.ini.php`.

## VI - Création d'un formulaire

Il y a deux manières de faire des formulaires : soit de manière classique, #à la main#, soit en utilisant le nouveau système de formulaire jForms. jForms s'occupe à la fois des traitements côté serveur des données reçues d'un formulaire (HTML, XUL ou service web) : création, sauvegarde, vérification des saisies, etc, et à la fois de la génération des formulaires HTML dans les templates.

Dans ce chapitre, on va utiliser la méthode #traditionnelle#, histoire de se familiariser avec les enchaînements d'actions et autres concepts de base de Jelix. Plus loin dans le tutoriel, nous vous montrerons comment utiliser jForms.

Le but de ce chapitre, va être de créer un simple formulaire afin d'enregistrer de nouvelles news.

### VI-A - Le template et les URLs

En premier lieu, nous allons faire le template, très simplifié, que l'on stocke dans le fichier newsform.tpl :

```
<h1>Création d'une news</h1>

<form action="{formurl 'news~default:createsave'}" method="POST">
  {formurlparam 'news~default:createsave'}
  <table>
  <tr>
    <th><label for="sujet">Sujet</label></th>
    <td><input type="text" id="sujet" name="sujet" /></td>
  </tr>
  <tr>
    <th><label for="texte">Texte</label></th>
    <td><textarea id="texte" name="texte"></textarea></td>
  </tr>
  <tr>
    <th><label for="date">Date</label> (AAAA-MM-JJ)</th>
    <td><input type="text" id="date" name="date" /></td>
  </tr>
  </table>
  <div><input type="submit" value="Enregistrer"/>
  <a href="{jurl 'news~default:index'}">Annuler</a>
  </div>
</form>
```

Rien de très complexe, au niveau HTML. Toutefois vous remarquerez l'usage de nouveaux tags : {formurl}, {formurlparam} et {jurl}. Quelques explications#

Dans Jelix, vous éviterez de mettre des URLs en dur dans les templates ou les actions, pour des raisons de facilité de maintenance et d'évolution. Le système d'URL de Jelix vous permet de centraliser toutes les URLs dans la config de Jelix ou dans un fichier urls.xml (tout dépend du moteur d'URL utilisé). Ainsi l'objet jUrl et le tag {jurl} vous permettent d'obtenir une URL en donnant uniquement le nom de l'action et des éventuels paramètres.

Ainsi, pour le lien #Annuler#, l'URL générée sera #/jelix/actu.org/www/index.php?module=news&action=default:index#. Si vous changez l'emplacement de index.php, ou si vous activez le moteur d'URLs significatives (en mappant news/list sur l'action news~default:index), vous n'aurez pas à modifier vos templates.

On aurait pu utiliser le tag {jurl} aussi sur la balise <form>, mais si l'URL contient des paramètres (comme c'est le cas ici avec module et action), il est préférable de les mettre dans des champs cachés. Cependant, si on changeait la configuration sur le mapping URL, ce n'est pas forcé qu'il y ait toujours ces paramètres. Aussi on utilise conjointement les plugins {formurl} et {formurlparam} qui se chargeront eux même de décider ce qu'il faut mettre dans l'attribut #action#, et ce qu'il faut mettre dans des champs cachés.

Note : par défaut, {jurl}, {formurl} et {formurlparam} échappe correctement dans l'URL les caractères réservés en HTML/XML.

## VI-B - Affichage du formulaire

On va créer une première action, 'createForm' pour afficher un formulaire de saisie d'une nouvelle news.

```
function createform(){
    $rep = $this->getResponse('html');
    $rep->title = 'Saisie d\' une actualité';
    $rep->bodyTpl = 'newsform';
    return $rep;
}
```

Pour accéder à cette page, on va rajouter un lien en bas dans le template listenews.tpl

Ajouter une news

Là encore, on utilise le tag {jurl}.

Maintenant rafraichissez la page de liste des news dans le navigateur ( <http://localhost/jelix/actu.org/www/index.php> ). Vous devriez avoir le lien #Ajouter une news#. Si vous cliquez dessus, le formulaire devrait s'afficher.

## VI-C - Sauvegarde des données

Comme on l'a indiqué dans le formulaire, il nous faut maintenant créer une action 'default:createsave' de sauvegarde des données.

En premier lieu, on récupère un record, que l'on remplit ensuite avec les données envoyées par le formulaire. Les paramètres d'URL (\$\_GET) ou postés (\$\_POST), sont accessibles via la méthode param() des contrôleurs.

```
$news = jDao::createRecord('news~news');
$news-> sujet = $this->param('sujet');
$news->texte = $this->param('texte');
$news->news_date = $this->param('date');
```

On récupère ensuite une factory de dao pour enregistrer le record.

```
$dao = jDao::get('news~news');
$dao->insert($news);
```

Enfin, on va rediriger vers la liste de news.

```
$rep = $this->getResponse('redirect');
$rep->action = 'news~default:index';
return $rep;
```

Cela donne au final :

```
function createsave(){
    $news = jDao::createRecord('news~news');
```

```
$news->sujet = $this->param('sujet');
$news->texte = $this->param('texte');
$news->news_date = $this->param('date');

$dao = jDao::get('news~news');
$dao->insert($news);

$rep = $this->getResponse('redirect');
$rep->action = 'news~default:index';
return $rep;
}
```

Vous pouvez maintenant utiliser le formulaire pour créer de nouvelles news.

## VII - Création d'un CRUD

Nous avons vu dans un chapitre précédent, comment créer un formulaire simple. Cependant nous n'avons pas réalisé tout ce qu'une interface d'administration de donnée exige : vérification de la saisie, affichage des erreurs, affichage d'une liste de news avec des liens pour modifier, suppression, modification etc#

Ce sont souvent des développements rébarbatifs, aussi Jelix propose une solution : un système de formulaire jForms, et un contrôleur générique pour faire du CRUD (Create/Read/Update/Delete).

### VII-A - Création du formulaire jForms

On va d'abord créer un formulaire jForms. Il s'agit d'un fichier XML dans lequel on liste tout les champs de saisie du formulaire, ainsi que leur libellé, les éventuelles contraintes sur leur contenu etc..

Il existe une commande jelix-scripts qui permet de créer un formulaire jForms à partir d'un DAO, et ça tombe bien, nous avons déjà un DAO pour les news. Allez dans le répertoire jelix-scripts et tapez la commande suivante :

```
php jelix.php createform news newsform news
```

#### **createform prend trois arguments :**

- 1 Le nom du module ;
- 2 Le nom du futur formulaire ;
- 3 Le nom du DAO.

Nous obtenons alors dans le fichier **actu.org/modules/news/forms/newsform.form.xml** :

```
<?xml version="1.0" encoding="utf-8"?>
<forms xmlns="http://jelix.org/ns/forms/1.0">

  <input ref="sujet" type="string">
    <label>sujet</label>
  </input>

  <input ref="texte" type="string">
    <label>texte</label>
  </input>

  <input ref="news_date" type="date">
    <label>news_date</label>
  </input>

  <submit ref="submit">
    <label>Enregistrer</label>
  </submit>

</forms>
```

La commande génère un formulaire très basique qu'il faut la plupart du temps adapter car le script ne peut pas tout deviner. Ici par exemple, il va falloir indiquer des libellés plus parlant. Dans d'autre cas, il vous faudra certainement changer le type de champs de saisie,etc.

Changeons les libellés, indiquons que tous les champs sont obligatoires et ajoutons une petite aide pour la date :

```
<?xml version="1.0" encoding="utf-8"?>
<forms xmlns="http://jelix.org/ns/forms/1.0">

  <input ref="sujet" type="string" required="true">
```

```

        <label>Sujet de la news</label>
    </input>

    <input ref="texte" type="string" required="true">
        <label>Contenu</label>
    </input>

    <input ref="news_date" type="date" required="true">
        <label>Date</label>
        <hint>Le format de la date est aaaa-mm-jj</hint>
    </input>

    <submit ref="submit">
        <label>Enregistrer</label>
    </submit>

</forms>
    
```

Notre formulaire est prêt à être traité. On devrait utiliser l'API de jForms pour le manipuler, mais dans notre tutoriel, le contrôleur générique va le faire pour nous.

## VII-B - Création du contrôleur CRUD

Créons le fichier modules/news/controllers/admin.classic.php, et commençons à y écrire un contrôleur pour faire une gestion de news :

```

<?php

class adminCtrl extends jControllerDaoCrud {

}

?>
    
```

Le contrôleur n'hérite pas de jController, mais de jControllerDaoCrud. C'est un contrôleur qui contient un ensemble d'actions prédéfinies. Il suffit alors de compléter certaines informations dans ses propriétés pour que la gestion des news soit opérationnelle.

En principe, dans un premier temps, indiquer le dao et le formulaire à utiliser suffit :

```

<?php

class adminCtrl extends jControllerDaoCrud {

    protected $dao = 'news~news';

    protected $form = 'news~newsform';

}

?>
    
```

Lancez maintenant dans votre navigateur : <http://localhost/jelix/actu.org/www/index.php?module=news&action=admin:index>

Vous devriez obtenir# une page blanche !

En effet, le contrôleur CRUD s'attend à ce que la réponse HTML qu'il récupère est un objet déjà préparé par l'application, et disponible pour toutes les actions (comme il est indiqué sur la page de la personnalisation des

réponses communes). Mais nous n'avons pas vu ça encore dans le tutoriel, il faut donc redéfinir une méthode du contrôleur pour préparer la réponse HTML.

Il s'agit de la méthode `_getResponse`, spécifique à `jControllerDaoCrud`. On va donc préparer un objet `jResponseHTML` en indiquant un titre de page, un template pour le body#

```
<?php
class adminCtrl extends jControllerDaoCrud {
    protected $dao = 'news-news';
    protected $form = 'news-newsform';

    protected function _getResponse(){
        $rep = $this->getResponse('html');
        $rep->title = "gestion des news";
        $rep->bodyTpl = "admin_news";
        return $rep;
    }
}
?>
```

Il ne faut pas oublier de créer le template que l'on vient d'indiquer, `modules/news/templates/admin_news.tpl` :

```
Gestion des news

{$MAIN}

Retour à l'accueil
```

Maintenant vous pouvez relancer <http://localhost/jelix/actu.org/www/index.php?module=news&action=admin:index>, et vous devriez voir la liste des news, avec des liens pour les modifier, les effacer etc#

Vous pouvez personnaliser cette affichage, en fournissant les templates propres à l'édition, au listage, en indiquant le nombre d'items par page dans la liste des news etc#

## VII-B-1 - Note

Pour le cas où il n'y a pas de DAO ou de formulaire préexistant, il est possible en une seule ligne de commande de créer le dao, le formulaire et le contrôleur :

```
php jelix.php createdaocrud le_module le_nom_de_la_table
```

Et ensuite, on peut directement appeler la page correspondante `site.com/index.php?module=le_module&action=default:le_nom_de_la_table` pour voir le résultat.

## VIII - Fin

Pour le moment, c'est fini. Le tutoriel n'est pas encore terminé. En attendant les futures pages du tutoriel, vous pouvez aller lire le manuel et posez vos questions dans le [forum](#).