Add-ons
Documentation développeur

Ce document a pour but de détailler la structure et le fonctionnement des add-ons par rapport aux développements spécifiques classiques afin qu'un développeur puisse en créer un facilement.

1. Informations

a. Version minimum

Pour pouvoir utiliser ce système d'add-on, il faut que votre projet soit au minimum en version :

- E-majine : 1.21b
- Saytup : 2.06b

b. Exemple

Cette documentation est fournie avec un add-on de démonstration présentant les différents développements spécifiques possibles dans les add-ons. Afin de mieux comprendre ce qui est décrit dans ce document, l'installation de cet add-on de démonstration est recommandée.

c. Arborescence de l'archive de l'add-on

Un add-on, ici "AddOnDemo", devra être structuré de la façon suivante :

- AddOnDemo.zip
  - readme.pdf (fiche descriptive)
  - images/ (répertoire d'images)
  - files/ (répertoire des fichiers de fonctionnement)
    - AddOnDemo/
      - assets/
        - images
          - logo.jpg
          - logo.png
      - modeles
        - addons/
          - AddOnDemo/
            - notification
              - myNotification.html
            - publication
              - myPubMethod.html
      - scripts
        - script.js
      - class/
        - AddOnDemo.class.php
        - actions/
          - manage/
            - MyManageAction.php

        - public/
            - MyPublicAction.php
    - crons/
        - cron_myCron.class.php
    - hooks/
        - core/
            - hook_2C.class.php
    - menus/
        - myMenu/
            - emajine_menu_myMenu.xml
            - menuMyMenu.class.php
    - mxtags/
        - myMxTag/
            - mxtag_myMxTag.php
            - myMxTag.php
    - notification
        - MyNotification.php
    - publication_methods/
        - MyPubMethod/
            - methodManageMyPubMethod.php
            - methodPublicMyPubMethod.php
            - method_MyPubMethod.xml
    - widgets/
        - mywidget/
            - widgetMywidget.class.php
            - widgetManageMywidget.class.php
            - specifbox_mywidget.xml
- database.sql
- data.xml
- readme.pdf

d. En base de données

Vous retrouverez la déclaration des add-ons en base de données dans la table “addons” :

- id : identifiant de l'add-on
- name : nom de l'add-on
- status : état de l'add-on activé (1) / désactivé (0)
- order : ordre d'exécution des add-ons
- date_install : date d'installation de l'add-on

- date_update : date de mise à jour de l'add-on
- date_activation : date d'activation de l'add-on

2. Les éléments obligatoires

Pour que l'addon soit pris en compte par le core d'e-majine, il faut qu'une structure minimale soit en place. L'add-on doit avoir un nom unique sans accents ni espace. Dans notre exemple il sera "AddOnDemo".
Dans un projet e-majine, les différents fichiers de l'add-on doivent être tous regroupés dans un dossier au nom de l'add-on : /html/specifs/addons/AddOnDemo/
On doit obligatoirement y retrouver :

- le fichier "data.xml"
- le fichier "readme.pdf"
- la répertoire "class" contenant une classe au nom de l'add-on : "AddOnDemo.class.php", cette classe doit étendre la classe Addons_Entity :

```php
namespace Addon\AddOnDemo;
class AddOnDemo extends \Addons_Entity
{
    /**
     * Permet de lancer des actions spécifiques après installation de l'add-on
     */
    public function onInstallation() {...}

    /**
     * Permet de lancer des actions spécifiques après désinstallation de l'add-on
     */
    public function onUninstallation() {...}

    /**
     * Permet de lancer des actions spécifiques après l'activation de l'add-on
     */
    public function onEnable() {...}

    /**
     * Permet de lancer des actions spécifiques après la désactivation de
l'add-on
     */
    public function onDisable() {...}

    /**
     * Permet de lancer des actions spécifiques avant l'export de l'add-on
     */
    public function onExport() {...}
```

Toutes les classes php de l'add-on doivent être déclarées au sein du namespace "Addon\[NomAddon]" dans notre cas : "`namespace Addon\AddOnDemo;`".
Par conséquent tous les appels à des éléments du core doivent être précédé d'un "\".
Exemples :

- \em_misc::getUserId()
- \em_db::all()
- \em_mx::text()
- new \Addons_Manager();

- ...

3. Les fichiers généraux

a. readme.pdf

Le fichier readme.pdf est un document descriptif contenant les éléments suivants :

- la version de l'add-on
- les notions de compatibilité
- les informations de contact du développeur
- la liste des fichiers présents dans le zip
- la liste des fonctionnalités de l'add-on
- une notice de configuration
- une notice d'utilisation
- une notice d'installation, notamment en cas de modification des templates déjà existants.
- les annexes (images, fichiers etc) placées dans le répertoire images/

b. data.xml

Le fichier datas.xml permet de stocker les informations liées à l'add-on. Il est structuré de la manière suivante :

```xml
<?xml version="1.0" encoding="UTF-8"?>
<data>
    <title>Add-on de démonstration</title>
    <name>AddOnDemo</name>
    <version><num date="2016-08-15">1.0</num></version>
    <emajine_compatibility><min_version>2.06b</min_version></emajine_compatibility>
    <webo_shop_info>
        <key></key>
        <url>http://webo-shop.com//</url>
        <author>
              <name><!CDATA[Medialibs]]></name>
              <email>medialibs@webo-shop.com</email>
        </author>
    </webo_shop_info>
    <description>
       <!CDATA[Description de l'add-on]]>
    </description>
    <compatibility>
       <!CDATA[Information de compatibilité (modules ou librairies nécessaires, …)]
    </compatibility>
</data>
```

c. database.sql

Ce fichier contient les différentes requêtes SQL nécessaire à l'add-on et qui seront automatiquement exécutées à l'installation de l'add-on. Les requêtes doivent être séparées par des ";" :

```
CREATE TABLE IF NOT EXISTS `spe_addon_demo` (
  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT 'identifiant',
  `label` varchar(255) NOT NULL COMMENT 'libellé',
  PRIMARY KEY (`id`)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8 COMMENT='Table créée à l''installation de l''add
AUTO_INCREMENT=5;
INSERT INTO `spe_addon_demo` (`id`, `label`) VALUES
(1, 'ceci est une liste '),
(2, 'de contenus de démonstration'),
(3, 'pour l''addon de démonstration'),
(4, 'lié à la documentation développeur');
```

4. Les développements spécifiques

a. Action

Fichiers

répertoire : /class/actions
Tout comme dans les développements spécifiques standards ce dossier est composé de deux dossiers :

- public
- manage

A l'intérieur de ces dossiers vous pouvez placer vos actions, à l'identique des actions spécifiques.

Appel

Pour les appeler, il faut ajouter le nom de l'add-on en second paramètre de l'url comme suit :

- public : /action-[NomAddon]-[NomFichier]
- manage : /manage/manageAction-[NomAddon]-[NomFichier]

Exemple :

- public : /action-AddOnDemo-MyPublicAction
- manage : /manage/manageAction-AddOnDemo-MyManageAction

 Ne pas oublier de définir le namespace de l'add-on dans la classe

b. Balise mx

Fichiers

répertoire : /class/mxtags
Dans ce répertoire on trouvera un répertoire par balise mx. Dans chacun de ces répertoires on trouvera le fichier de définition de la balise ainsi que le fichier déterminant ce que la balise mx retournera :

- myMxTag/
    - mxtag_myMxTag.php
    - myMxTag.php

Déclaration de la balise

Dans le fichier de définition de la balise il y a plusieurs éléments à modifier :

- Ajouter un “\” devant le “em_misc::...”
- $objectName doit prendre en compte le namespace comme suit :
    - `$objectName = 'Addon\\[NomAddon]\\[NomObjet]';`
    - soit dans notre exemple : `'Addon\\AddOnDemo\\mxMyMxTag'`
- La déclaration du “$path” peut être supprimée et remplacée dans l’appel de la classe de cette façon :
    - `\em_misc::loadPHP(__DIR__ . '/myMxTag.class.php', $objectName);`

Déclaration du contenu

Le fichier de définition du contenu de la balise fonctionne comme dans les développements spécifiques ordinaires.

La balise mx spécifique

La balise spécifique à mettre en place dans les templates possède un paramètre supplémentaire obligatoire :

- `<mx:specifs id="[NomBaliseSpe]" addon="[NomAddon]"/>`
- Exemple : `<mx:specifs id="myMxTag" addon="AddOnDemo"/>`

Ne pas oublier de définir le namespace de l’add-on dans la classe

c. Hook

Fichiers

répertoire : /class/hooks
L'arborescence de ce répertoire est identique à celle du répertoire "hooks" des développements spécifiques standards.

Ordre d'exécution

Si le même hook est déclaré dans plusieurs addons ou dans des développements spécifiques, les hooks des addons sont appelés suivant l'ordre qui leur a été attribué (par défaut l'ordre d'installation). Cet ordre peut être retrouvé en base de données dans la table "addons" dans la colonne "order".
Les hooks spécifiques sont appelés après les hooks des add-ons.

Paramètres

L'exécution de plusieurs add-ons peut modifier les valeurs des paramètres si celui-ci est :

- un objet
- passé par référence
- défini comme correspondant à la valeur de retour du hook dans la définition du hook

L'appel à la fonction getRawParameters() de la classe du hook permet de récupérer le tableau des paramètres dans son état d'origine, avant toute exécution de hook :

```
$arrayParams = $this->getRawParameters();
$myFirstParam = $arrayParams[0];
```

Arrêt du processus standard

Puisque la classe de hook appartient au namespace de l'add-on pour stopper le processus standard, il faut utiliser la syntaxe suivante :

```
return \Emajine_Hooks::STOP_STANDARD_PROCESS;
```

Exemple

Dans l'add-on exemple vous pourrez retrouver une interface de configuration de l'add-on déclarée grâce à un hook 2C.
Le rendu se trouve dans l'interface d'e-majine dans la rubrique :

- "Mon Site -> Configuration -> Démo - documentation développeur"


Si votre hook ne semble pas être exécuté : allez dans les outils de développement et cliquez sur le bouton "Réinitialiser le cache"


Ne pas oublier de définir le namespace de l'add-on dans les classes de hook

d. Méthode de publication

Fichiers

répertoire : /class/publication_methods
Dans ce répertoire on trouvera un répertoire par méthode de publication. Chacun de ces répertoires contient le fichier XML de définition de la méthode de publication ainsi que les deux fichiers permettant de définir l'interface de configuration de la méthode de publication et le contenu publique :

- publication_methods/
    - MyPubMethod
        - methodManageMyPubMethod.class.php
        - methodPublicMyPubMethod.class.php
        - method_MyPubMethod.xml

Utilisation

Le fonctionnement des fichiers reste identique au fonctionnement des fichiers des développements spécifiques standards.

Exemple

Vous pourrez retrouver La méthode de publication spécifique dans une section qui est propre à l'add-on lors de la sélection du type de publication.
Sur le site de démonstration de l'add-on vous la retrouverez dans la rubrique suivante : http://addondemotest.s21144.m20.atester.fr/publication-add-on/

 Ne pas oublier de définir le namespace de l'add-on dans les classes

e. Notification

Fichiers

répertoire : /class/notification
A l'identique des notifications spécifiques standardes il y a une classe de déclaration par notification.

- notification/
    - MyNotification.php

Pour plus d'informations sur le développement de notifications, se référer à la documentation :
https://docs.google.com/document/d/1OHU4DP1soZVOsLJB-0KsejdGLYPWKVMahiFEy3ebuTE/edit#heading=h.9pbq26bes3ai

Configuration de la notification

La configuration de la notification de l'add-on se fait au même endroit que les autres notifications standardes de votre CMS, puis en fonction de la configuration de la configuration mise en place dans la classe de votre notification.
Dans l'exemple elle se trouve dans les notifications utilisateur, section "Add-on démo"

Envoi de la notification

Pour envoyer la notification il faut spécifier le nom de l'add-on. Dans le cas de notre exemple d'add-on de démonstration :

```
$notif = \Mail_Notification::factory('addon.AddOnDemo.MyNotification');
$notif->send($userId);
```

⚠ Ne pas oublier de définir le namespace de l'add-on dans la classe

f. Section d’e-majine

Fichiers

répertoire : /class/menus
Dans ce répertoire on trouvera un répertoire par section d’e-majine. Chacun de ces répertoires contient le fichier XML de définition du menu ainsi que la classe permettant de définir le contenu affiché dans ce nouveau menu.

- menus/
    - myMenu/
        - emajine_menu_myMenu.xml
        - menuMyMenu.class.php

Déclaration du menu

Dans le ficher XML de définition de la section il faut penser à modifier le chemin jusqu’à la classe définissant le contenu du menu.

Exemple :

```
<script>addons/AddOnDemo/class/menus/myMenu/menuMyMenu.class.php</script>
```

Utilisation

La classe de définition du contenu fonctionne à l’identique de la classe des développements spécifiques standards.

 Ne pas oublier de définir le namespace de l’add-on dans la classe

g. Tâche planifiée

Fichiers

répertoire : /class/crons
Les fichiers des tâches planifiées sont à la racine de ce dossier “crons” :

- crons/
    - cron_myCron.class.php

Leur fonctionnement est identique à celui des tâches planifiées des développements spécifiques standards.

Lancement manuel

La tâche planifiée ne peut être lancée manuellement depuis les outils de développements, si cette opération est nécessaire, il est conseillé de faire une interface spécifique pour cela.
Dans l’add-on de démonstration, l’interface d’activation de l’add-on permet également de lister les tâches planifiées et de les exécuter.

Ne pas oublier de définir le namespace de l’add-on dans la classe

h. Widget

Fichiers

répertoire : /class/widgets
Dans ce répertoire on trouvera un répertoire par widget. Chacun de ces répertoires contient le fichier XML de définition du widget ainsi que les classes permettant de définir l'interface d'administration ainsi que l'apparence publique du widget.

- widgets/
    - mywidget/
        - widgetMywidget.class.php
        - widgetManageMywidget.class.php
        - specifbox_mywidget.xml

Déclaration du widget

Dans le fichier XML de définition du widget, il faut penser à modifier le chemin jusqu'aux classes définissant l'interface d'administration et le contenu du widget.

Exemple :

```
<scriptmanage>addons/AddOnDemo/class/widgets/mywidget/widgetManageMywidget.class.php</script
manage>
<scriptpublic>addons/AddOnDemo/class/widgets/mywidget/widgetMywidget.class.php</scriptpublic
>
```

Configuration

Lorsque vous mettez en place un widget au sein d'un add-on, dans l'interface d'administration des widgets de votre CMS apparaît une nouvelle catégorie au nom de l'add-on où tous les widgets de l'add-on sont présents.

Utilisation

Le fonctionnement des fichiers reste identique au fonctionnement des fichiers des développements spécifiques standards.

Ne pas oublier de définir le namespace de l'add-on dans les classes

5. Les ressources

a. Introduction

Afin d'éviter les installations compliquées il est conseillé de mettre les templates, images et scripts dans un dossier "assets" à la racine de l'add-on et de déplacer les fichiers au bon endroit à l'activation de l'add-on les différentes ressources doivent être déplacées dans leur répertoire de destination.
Dans l'exemple de l'add-on de démonstration ce travail est réalisé à l'activation de l'add-on dans les classes du dossier "/specifs/addons/AddOnDemo/class/tools/interface" et plus particulièrement dans la classe "InterfaceInstallation".
Dans le cas où l'add-on est mis à jour, les fichiers ne sont pas écrasés, il faut les mettre à jour manuellement ou en réactivant la procédure de déplacement des fichiers (désactivation puis activation si l'action de déplacement des fichiers a été liée à l'activation de l'addo-on comme dans l'exemple fourni).

```
/**
* Fonction de copie d'une arborescence à une autre
* @param  string $source chemin vers l'arborescence source
* @param  string $dest   chemin vers l'arborescence destination
* @return boolean | null
*/
protected function copyTree($source, $dest)
{
        if (!file_exists($source)) {
                return;
        }
        if (is_link($source)) {
                return symlink(readlink($source), $dest);
        }
        if (is_file($source)) {
                return copy($source, $dest);
        }
        if (!is_dir($dest)) {
                $this->makeDir($dest);
        }
        $dir = dir($source);
        while (false !== $entry = $dir->read()) {
                if ($entry == '.' || $entry == '..') {
                        continue;
                }
                $this->copyTree("$source/$entry", "$dest/$entry");
        }
        $dir->close();
        return true;
}
```

```php
/**
* création d'un répertoire
* @param  string $dirToCreate répertoire à créer
* @return null
*/
protected function makeDir( $dirToCreate )
{
        if (!is_dir($dirToCreate)) {
                $dirs = explode('/', substr($dirToCreate, 1));
                $path = '';
                foreach ($dirs as $dir) {
                        $path .= '/' . $dir;
                        if (!is_dir($path)) {
                                mkdir($path);
                        }
                }
        }
}
```

b. Images

répertoire : /assets/images

Si l'on se base sur l'exemple fourni, les images présentes dans ce répertoire seront déposées dans le dossier /html/images/addons/[NomAddon] avec la sous-arborescence définie dans votre répertoire.

```php
copyTree('html/specifs/addons/AddOnDemo/assets/images',
'html/images/addons/AddOnDemo');
```

c. Templates et css

répertoire : /assets/modeles

Tout comme les images, les templates et css doivent idéalement être déplacés dans le dossier "/html/modeles". Les sous-arborescences doivent être déplacées dans chaque langue et chaque modèle de chaque langue. Les répertoires existants ne devront pas être écrasés et les autres créés en fonction de l'arborescence présente dans le dossier "modeles" de l'add-on.

```php
foreach ('html/modeles/*') as $lang) {
   if (is_dir($lang)){
        foreach (glob($lang . '/*') as $modele){
       if (is_dir($modele)) {
            $this->copyTree(html/specifs/addons/AddOnDemo/assets/modeles',
$modele);
        }
         }
    }
}
```

d. Scripts

répertoire : /assets/scripts

Tout comme les images, les scripts (javascript ou autres) doivent être déplacés dans "html/scripts/addons/[NomAddon]" afin d'être différenciés des autres scripts du site.

Les appels aux scripts se feront au mieux dans un traitement php via la méthode \em_misc::loadScript() afin d'éviter les éléments à modifier dans les templates.

```
copyTree('html/specifs/addons/AddOnDemo/assets/scripts',
'html/scripts/addons/AddOnDemo');
```

Il faudra prêter attention au nommage des fonctions des fichiers javascript des add-ons en les préfixant du nom de l'add-on par exemple afin d'éviter les conflits avec d'éventuels autres add-ons ou développements spécifiques.

6. Installation et utilisation

L'installation de l'add-on et automatique une fois que le répertoire au nom de l'add-on est créé dans le dossier "specifs/addons/" et que la structure est correcte (cf les éléments obligatoires).

Par défaut à l'installation l'add-on n'est pas activé. Tant que l'add-on n'est pas activé, les développements de l'add-on ne fonctionneront pas.

Si vous souhaitez l'activer par défaut à l'installation, dans la classe au nom de l'add-on il y a la fonction "onInstallation" qui peut être utilisée :

```
$addon = \Addons_Manager::getInstance()->getAddon('AddOnDemo', true);
\Addons_Manager::getInstance()->enable($addon);
```

Pour le désactiver, il suffit d'utiliser la méthode suivante :

```
$addon = \Addons_Manager::getInstance()->getAddon('AddOnDemo', true);
\Addons_Manager::getInstance()->disable($addon);
```

Vous pouvez également laisser la main à l'utilisateur concernant l'activation et créer une interface spécifique pour.
Dans l'add-on de démonstration lié à cette documentation, il y a une interface à votre disposition afin d'activer, désactiver l'add-on et lancer les crons de l'add-on manuellement.
Elle est accessible par une action du manage : /manage/manageAction-AddOnDemo
Vous trouverez son code dans le dossier /specifs/addons/AddOnDemo/class/tools/interface et le template lié dans /specifs/addons/AddOnDemo/class/manageTpls .