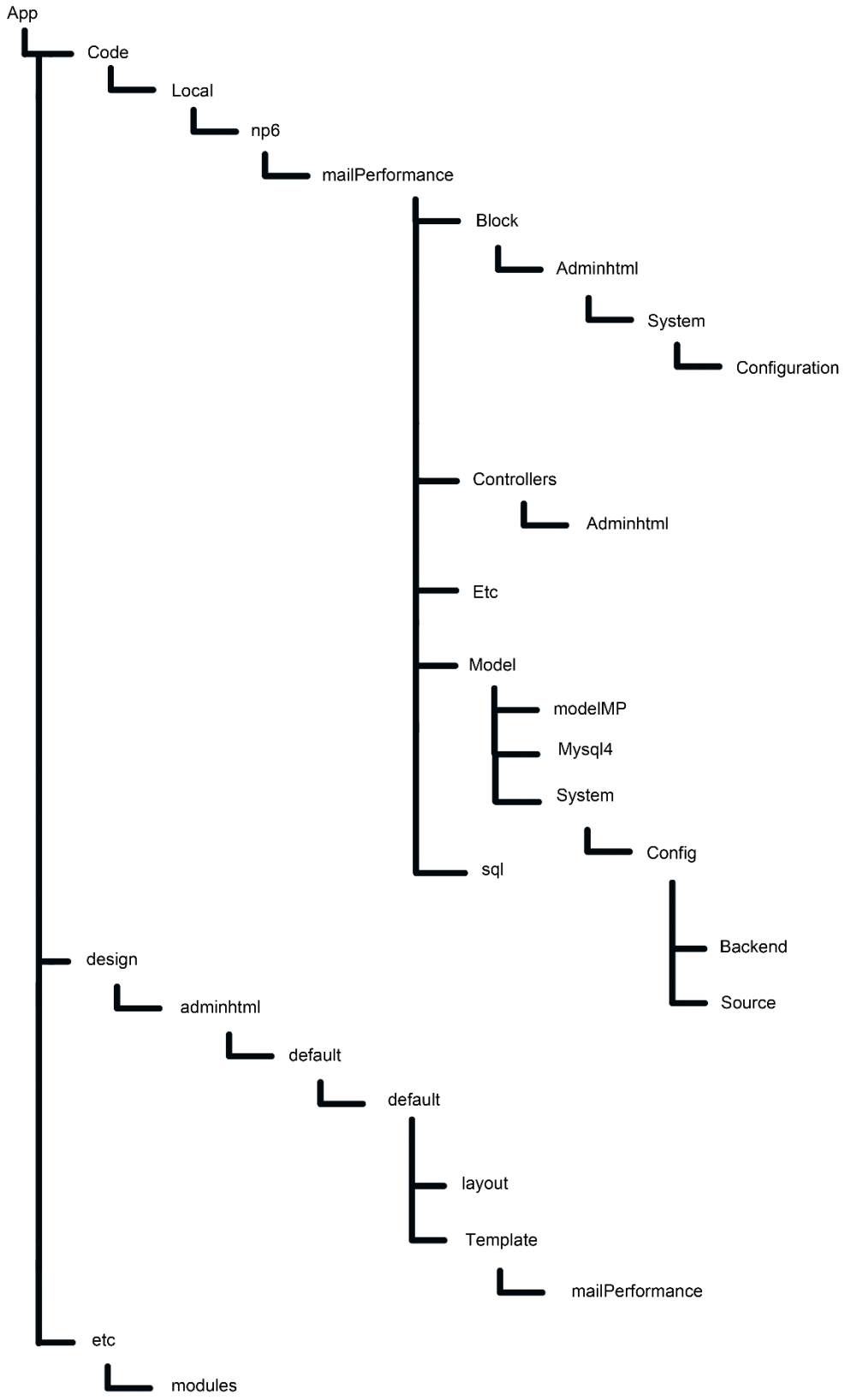
Documentation Technique Module MailPerformance pour Magento

Sommaire :

I. Architecture du projet
II. Détail du code
    1. La base du module
        a. Déclarer son module (part 1)
        b. Déclarer son module (part 2)
    2. Utilisation de l'api
    3. Création des fields
        a. Field statique
        b. Field Dynamique
    4. Gérer les évènements Magento
III. Ajouter le module à Magento
IV. Fonctionnalités actuellement disponible
V. Fonctionnalités manquante

I. Architecture du projet



L'architecture d'un module Magento étant plutôt complexe (schéma ci-dessus), les noms des dossiers, l'utilisation des majuscules ou encore le nom des classes doivent respecter les pratiques Magento pour que le projet fonctionne.

Nous allons d'abord expliquer l'architecture de dossier du projet :

- App : dossier de Magento qui contiendra tous ce qui sera code, frontend, backend, déclaration du module.

- App/Code : explicite, contiendra tout notre code
- Code/local : contiendra les différents namespace
- Local/np6 : np6 est ici le nom du namespace de notre projet
- Np6/mailPerformance : mailPerformance est le nom de notre module

- mailPerformance/block : va contenir tous les blocks de notre module
- block/Adminhtml/system/configuration : va contenir les blocks utilisés pour réaliser l'affichage des différents fields personnalisés dans les onglets de configuration

- mailPerformance/controllers : contient les controllers de notre module
- controllers/adminhtml : contient les controllers de notre module pour la partie admin

- mailPerformance/etc : contient deux fichiers très important :

    - config.xml : on va y déclarer la structure de notre code
    - system.xml : permet de déclarer des options de configuration supplémentaire et personnalisé dans l'admin panel.

- mailPerformance/model : contiendra toutes les classes pour communiquer avec l'api et autre classes instanciable
    - model/mysql : classe nécessaire à l'accès en BDD
    - model/modelMP : classe mailPerformance

    - model/System/config : Contient les fichiers de backend et éléments source pour les fields de configuration.

- mailPerformance/sql : contient les modifications à apporter à la BDD lors de l'installation (pour l'instant le modules ne s'installant pas à proprement parler vue qu'on réalise un simple copier-coller, la BDD en local est à modifier manuellement)

- App/design : contient les layouts/templates dans ses sous dossier, (on ne verra ici que le dossier adminhtml les autres ne nous concernant pas)
- design/adminhtml/default/default : contient les dossiers layout et template.
- default/layout : contient les layouts. Le layout indique quel block et quel Template utiliser.
- Default/template : contient les templates classé par module
- Template/mailPerfomance : contient les templates de notre module

- App/etc/modules : contient la définition de chaque modules dans des fichiers xml.

II. Détail du code

1. La base du module

A. Déclarer son module (part 1)

Le fichier « Np6_All.xml » qui se trouve dans app/etc/modules, pour que Magento puisse comprendre qu'on lui a rajouté nos modules.

Sensible à la case, le nommage se fait de cette manière :
« Namespace_All.xml »

Il contient la déclaration de notre namespace et de nos modules, leurs noms, s'ils sont activés, dans quel pool ils se trouvent. (Le pool correspond aux dossiers dans App/code) Ici notre module se trouve dans le pool « local ».

B. Déclarer son module (part2)

Le fichier « config.xml » du dossier
« App/code/local/np6/mailPerformance/etc/ »

Il déclare la structure de notre code. Le code étant commenté je me contenterai de préciser que le nommage est sensible à la casse ainsi que certains noms séparé par des '_' sont en fait des chemins.

Exemple :

```
<models>
    <mailPerformance>
      <class>np6_mailPerformance_Model</class>
```

```
            <resourceModel>mailPerformance_mysql4</resourceModel>
        </mailPerformance>
    </models>
```

Les noms du type “np6_mailPerformance_Model » sont donc des chemins, ici du type : « namespace_module_dossierModel »

Il faut donc faire attention dans certain cas, savoir de quel dossier on part peut s’avérer ambigu.

Lorsque nous évoqueront les évènements nous reviendrons brièvement sur ce fichier config.xml

2. L’utilisation de l’api

Pour utiliser l’api, nous allons utiliser les fichiers présent dans le dossier et sous dossier de /app/code/local/np6/mailPerformance/Model

Dans le dossier model on va trouver :

- Le dossier ModelMP : contient toutes les class mailPerformance
- Le dossier Mysql4 : contient les fichiers nécessaires à l’utilisation de la BDD Magento
- Le dossier System pour stocker des éléments de backend et les valeurs pour les fields de configuration
- Les class « connector » des class présente de modelMP
- La class np6_mailPerformance_Model_MPAPI, classe pour effectuer les requêtes.
- La class np6_mailPerformance_Model_Api, qui servira à se lier à mailPerformance
- La class Observer, qui permet de gérer les évènements.
- La class mailPerformance, liée aux éléments du dossier Mysql4 pour l’accès en BDD.

De manière simple np6_mailPerformance_Model_Api va stocker notre connexion, les infos récupérées de l'api et toutes ses fonctions seront accessibles depuis le code à l'aide de Magento de la manière suivante :

```
Mage::getSingleton('mailPerformance/api')->getFieldType("email");
```

Mage::getSingleton, va permettre d'instancier une fois la classe api puis elle sera stocké en mémoire. De cette manière on a accès à toutes ses fonctions de manière simple, partout dans notre code.

A noter que c'est pour cela que l'on utilise la convention de nommage pour la classe api :« namespace_nomModule_ChemindossierModel_NomdeFichier » sans cela on ne pourrait pas y accéder depuis
Mage ::getSingleton('nomdumodule/NomdeFichier')

3. La création de fields

Pour créer des fields dans System>config, il va falloir utiliser le fichier System.xml de app/code/local/np6/mailPerformance/etc

a. Field statique

Dans ce fichier on va pouvoir déclarer un nouveau Tabs (Onglet principale visible sur le côté gauche)

On pourra y créer des sections (sous onglet)

Puis des groupes et enfin des fields

Il suffit de répéter la même syntaxe que dans le fichier xml du projet pour ajouter des Field/group/sections.

La seule chose importante à voir ici est la possibilité de préciser pour les fields :

- Le source_model, pour définir quel seront les valeurs contenue par le field
- Le frontend_model / frontend_type, fronted_type permet d'utiliser un frontend de base (voir doc Magento) alors que frontend_model, permet de définir son propre model
- Backend_model, permet d'implémenter une action en backend

Le source_model :

Défini de la manière suivante :

<source_model>mailPerformance/system_config_source_account</source_model>

Nommage : NomDuModule/{chemin à partir de l'intérieur du dossier model}

La classe ensuite créé doit implémenter « public function toOptionArray() »

Qui retourne un tableau de tableau du type :

```
Array (
    Array('value' => 0 , 'label' => 'monlabel'),
    Array('value' => 1 , 'label' => 'monlabel1'),
) ;
```

L'utilisation qui en est faite dans notre projet est par exemple :

Au constructeur à l'aide de notre class api on va récupérer les infos qui nous intéressent.

```
$this->_account_details = Mage::getSingleton('mailPerformance/api')->getContact();
```

Et dans la fonction « toOptionArray » on va renvoyer un tableau avec les infos récupéré plus haut.

Le frontend_model :

Définit de la manière suivante :

<frontend_model>mailPerformance/adminhtml_system_config_account</frontend_model>

Nommage : NomDuModule/{chemin à partir de l'intérieur du dossier block}

La classe ensuite créé doit extends
« Mage_Adminhtml_Block_System_Config_Form_Field »

Et implementer « protected function _getElementHtml(Varien_Data_Form_Element_Abstract $element)"

Cette fonction retourne le html de l'élément affiché

Attention dans certain cas pour que la sauvegarde prenne effet, ne pas oublier de préciser id et Name à notre objet html (voir les fichier ComboBinding, ComboDV, RadioButton)

Le Backend_model :

Défini de la manière suivante :

<backend_model>mailPerformance/system_config_backend_dataBinding</backend_model>

Nommage : NomDuModule/{chemin à partir de l'intérieur du dossier model }

Il extends Mage_Core_Model_Config_Data

Et implémente la fonction de l'évènement souhaité (voir doc Magento)

par exemple dans notre projet nous implémentons la fonction _aftersave()

On peut du coup y effectuer notre code puis, ne pas oublier de retourner la fonction parent à la fin, sans quoi il y a un risque que cela ne marche pas correctement

return parent::_afterSave();

b. Field dynamique

Pour Gérer l'apparition de champs en fonction d'un autre, ce que j'appelle ici Field dynamique. On va créer un observer sur la fonction sauvegarde de system>configuration.

Et en fonction de ce qui est sauvegardé (ici notre clé d'authentification) on va ajouter dynamiquement les autres onglets.

Pour créer un observer on va d'abord devoir déclarer l'événement dans config.xml entre les balises « events»

On obtient :

<{Nom de l'event}>

<observers>

<{nom aux choix généralement namespace_nomDel'event}>

<class>{namespace}_{NomDuModule}_model_observer</class>

<method>{nom de la method}</method>

</{nom aux choix généralement namespace_nomDel'event}>

</observers>

</{Nom de l'event}>

La balise <class> indique le chemin de notre fichier ici:

np6_mailPerformance_model_observer

On créé donc un fichier Observer.php dans /model

À l'intérieur de la class on implémente la méthode indiqué dans le config.xml avec en paramètre « Varien_Event_Observer $observer »

On va ensuite pouvoir extraire la configuration et y apporter les modifications nécessaires.

Le code parle de lui-même pas besoin de plus d'explication.

4. Gérer les évènement Magento

De la même manière que pour l'observer réalisé en partie 3.

Il faut d'abord chercher l'évènement qui nous intéresse dans la doc :

http://www.magentocommerce.com/wiki/5_-_modules_and_development/reference/magento_events

Ou

https://www.nicksays.co.uk/magento-events-cheat-sheet-1-7/

Évènement intéressant restant à implémenter :

- **sales_order_item_cancel** (à annulation d'une commande)
- **sales_order_place_after** (à la création d'une commande durant le processus de validation)
- **sales_order_payment_pay**(quand la commande est payé)
- pour le retour produit je n'ai pas trouvé d'évènement correspondant

Ensuite le rajouter dans le xml entre les balises <event> :

<{Nom de l'event}>

<observers>

<{nom aux choix généralement namespace_nomDel'event}>

<class>{namespace}_{NomDuModule}_model_observer</class>

<method>{nom de la method}</method>

</{nom aux choix généralement namespace_nomDel'event}>

</observers>

</{Nom de l'event}>

Puis implémenter notre classe « observer ».

III. Ajouter le module à Magento

Pour cela il suffit de copier-coller le dossier /app du projet sur le /app du Magento, les dossiers/fichiers se placeront au bon endroit.

Le module demande une table dans la bdd :

Un script présent dans le dossier sql est sensé la créer automatiquement, mais le fait de copier/coller le dossier app ne lance pas l'installation du module il faut donc créer la table manuellement.

app\code\local\np6\mailPerformance\sql\instal-1.0.0.php

Donc une table : « np6_mailperformance »

avec « id_magento » int(11) primary_key,

"id_mailperf" varchar(8)

Pour y accéder, une fois installé il suffit de se connecter sur l'admin panel et d'aller dans l'onglet « System > Configuration ».

IV. Fonctionnalité actuellement disponible

Dans System>configuration :

- Onglet d'Authentification
- Si Authentification Succès :

- Affichage des informations de compte
- Apparition de l'onglet configuration :
    - Groupe Data binding, liaisons des champs Magento aux champs MailPerformance
    - Groupe User Inscription, choisir comment ajouter ou non l'utilisateur à l'inscription.
    - Groupe User inscription, choix du segment dans le quel ajouter la cible
    - Un groupe par évènement :
        - Le premier est l'évènement sur création/modification de panier (event : checkout_cart_save_before)
- Apparition onglet Action :
    - Export csv
    - Création de segment
- Apparition onglet Formulaire :
    - Configuration du formulaire intégré sur page

V. Fonctionnalité manquante

Il reste à faire sur ce projet pour avoir un module équivalent à celui de Prestashop :

- La page pour les formulaires (elle est commencé mais par terminé)
- L'implémentation des évènements (1/5 de réalisé)
- L'implémentation de l'abandon de panier