MetaModels Documentation

Version 2.0.0

Team MetaModels

janv. 26, 2019

Table des matières

Ceci est la documentation officielle de *MetaModels*, une extension pour le Contao Content Management System.

Cette documentation est divisée en 3 sections :

Dans manuel vous trouverez la documentation générale sur MetaModels.

Dans *Cookbook* vous trouverez des solutions spécifiques à des besoins spécifiques.

Dans références vous trouverez des informations de référence comme une liste des événements.

Si vous souhaitez également contribuer, merci de cliquer sur le lien „Edit on GitHub“ dans le coin supérieur droit ou de visiter la page du projet sur GitHub.

CHAPITRE 1

Manuel

1.1 Introduction à MetaModels

1.1.1 Qu'est-ce que MetaModels ?

MetaModels est une extension pour le système de gestion de contenu Contao. Cette extension vous permet d'ajouter différents types de données structurées et de les afficher sur votre site en utilisant les vues en liste et en détail, des filtres, des tris, la pagination, le multilinguisme et d'autres choses encore. . .

« Données structurées » signifie du contenu habituellement stocké dans une base de données avec différentes tables et relations. MetaModels propose différents types de champs (appelés Attributs) comme, entre autres, le texte, les listes de sélection, les boites à cocher, les boutons radio, les nombres entiers et décimaux, les booléens oui/non, les gestionnaires de fichiers etc. . .

Les champs d'applications possibles sont les catalogues de produits, les événements, les menus, les listes d'adresses ou d'employés, la gestion de maisons ou de locations, les galeries d'images ou du contenu multilangues texte et image.

MetaModels permet de créer ses modèles de données directement dans le backend de Contao. Sans avoir besoin de coder contrairement à une extension spécifique. La création des masques de saisie pour le backend comme des sorties pour le frontend avec filtres optionnels sont partie intégrante de la création d'un MetaModel.

L'extension MetaModels permet une grande flexibilité dans la saisie et l'affichage des données et peut répondre ainsi à de nombreux besoins. Vous pouvez trouver plus de détails dans rst_features. Vous pouvez aussi jeter un œil aux exemples de MetaModels ou consulter le forum Contao pour différents exemples présentés sur le forum allemand.

1.1.2 Histoire de MetaModels

MetaModels a été initialement créé comme la nouvelle génération de la célèbre extension Catalog.

Au fil du temps, "Catalog" est devenue une extension complexe offrant de nombreuses possbilités à Contao. Mais il devenait malheureusement de plus en plus difficile de la maintenir et d'ajouter de nouvelles fonctions.

L'expérience acquise à développer Catalog 1 et Catalog 2 nous a convaincus qu'il nous fallait repartir de zéro.

C'est pourquoi nous avons développé « MetaModels » : une extension entièrement nouvelle intégrant des logiques de programmation modernes. Notre but était de développer un extension avec un code de base flexible et exensible.

La version actuelle MetaModels 2.0 est le résultat de nombreuses heures de discussion sur « quelle est la meilleure solution » et d'un gros travail de programmation.

1.1.3 MetaModels comparé à d'autres outils.

MetaModels différencie le travail d'administrateur et d'éditeur. L'administrateur (ou le développeur) crée le ou les MetaModels avec masques de saisie et options de sortie ; le ou les éditeur(s) ajoute(nt) les contenus comme on le fait d'habitude dans les autres parties du backend de Contao.

Les masques de saisie permettent de définir précisément quelle donnée peut (ou doit) être entrée et de quelle manière. Les extensions « [dma_elementgenerator] » ou « [rocksolid-custom-elements] » offrent des fonctions smilaires. La différence est que MetaModels vous permet également d'afficher des ensembles de données complexes et propose différentes options de sortie et de filtrage.

Avant de débuter un nouveau projet, vous devez vous demander s'il vaut mieux développer votre propre extension ou utiliser MetaModels. Il n'y a pas de réponse générale à cette question parce que les deux solutions permettent de résoudre différents problèmes. Ces différents aspects peuvent vous aider à prendre votre décision :

Pour développer votre propre extension : Le produit à développer doit pouvoir être commercialisé, par exemple une extension commerciale qui devra être facilement mise à disposition des autres utilisateurs de Contao ? Envisagez de développer votre propre extension. Ça vous demandera des compétences minimum en programmation PHP et de connaitre de l'API Contao.

Pour MetaModels : MetaModels est certainement un bon choix lorsque vous souhaitez créer une solution spécifique facilement personnalisable dans le backend de Contao. MetaModels a également des atouts à faire valoir s'il vous faut des fonctions spécifiques comme le multilinguisme. MetaModels permet à l'utilisateur de développer des solution sans programmation. Cependant, des connaissances basiques en PHP, HTML et les bases SQL vous permettront d'utiliser à plein les possibilités offertes par MetaModels.

1.1.4 Ressources

- Site du projet MetaModels
- MetaModels chez Github
- Manuel MetaModels chez Github
- MetaModels dans le wiki de Contao
- Sous-forum MetaModels de la communauté Contao
- Canal IRC de MetaModels sur freenode #contao.mm

1.2 Installation et mise à jour de MetaModels

MetaModels doit être installé sur une version LTS (long time support) de Contao : - la version actuelle est Contao 3.5.x

1.2.1 Installation à l'aide de Composer

MetaModels et ses dépendances peuvent être installés via le "gestionnaire de paquet Composer <https://c-c-a.org/ueber-composer>"

Si votre installation de Contao utilise déjà le nouveau gestionnaire de paquets Composer, vous pouvez sélectionner et installer MetaModels facilement en tapant son nom dans le champs de recherche comme suit : * metamodels/bundle_all

La version actuelle du bundle est la « 2.0.x » qui installera l'ensemble des éléments autour de « MetaModels core ». Le menu de restriction vous permet de choisir entre différentes versions comme « bugfix release », « feature release » etc. « Feature release » active l'ensemble des fonctions de MetaModels.

Si vous n'avez pas besoin de tous les filtres et/ou des attributs, vous pouvez les installer séparément. Vous pouvez aussi sélectionner un autre Bundle. Les paquets mentionnés regroupent les éléments nécessaires à certains besoins.

Vous pouvez voir les paquets déjà installés en affichant le graphe des dépendances (case à cocher) du client Composer de Contao (« Package management »).

1.2.2 Installation via Nightly build

L'alternative à Composer est d'installer MetaModels par FTP. Pour cela, vous devez télécharger la version la plus récente depuis le site du projet http://now.metamodel.me/ Dézippez-le et envoyez-le par FTP sur votre serveur. La plupart des dossiers doivent être placés dans le dossier */system/module* - seuls deux fichiers PHP qui gèrent les fonctions Ajax doivent être placés à la racine du dossier Contao.

Ensuite, vous mettrez à jour la base de données par le « gestionnaire d'extensions ». Si vous obtenez le message : `Fatal error: Class 'MetaModels\Helper\UpgradeHandler' ....! metamodels-tng-branch/config/runonce_0.php` vous devez purger le cache interne. Cette option se trouve dans le menu « Maintenance » du backend de Contao.

1.2.3 Test de paquets spécifiques à l'aide de Composer

Le bundle « bundle_all » contient tous les paquets à jour disponibles pour MetaModels. Il peut y avoir des paquets supplémentaires avec des correctifs de bugs ou de nouvelles fonctions à tester. Par exemple, pour MetaModels core, il peut s'agir de « dev-hotfix-xyz ». Vous pouvez trouver ces paquets sur Github dans le dépôt correspondant (ex MetaModels/core) sous les onglets de "branches" <https://github.com/MetaModels/core/branches>'_.

Pour tester un paquet, vous devez le sélectionner et l'installer séparément par le gestionnaire de paquets. Pour cela, cochez la case « dépendances installées » puis cliquez sur le paquet correspondant (ex : "metamodels/core"). Enfin, dans le menu déroulant, choisissez par exemple "dev-hotfix-xyz".

Après avoir cliqué sur « Mark package to install » vous devez modifier le fichier Composer-JSON. Pour cela, dans Package manager, cliquez sur « settings » puis « expert mode ». Le fichier JSON est affiché. Il faut ajouter « as 2.0.0 » à l'entrée concernée sous le nœud « require ». Si vous souhaitez installer plusieurs paquets spécifiques, vous devez le faire pour chacun d'entre eux.

Exemple : `"metamodels/core": "dev-hotfix-xyz"` à modifier `"metamodels/core": "dev-hotfix-xyz as 2.0.0"`

Après l'installation via « update packages », cliquez sur « Clear Composer cache » dans l'onglet « settings » du gestionnaire de paquets.

Comme MetaModels est étroitement lié à DC_general (DCG) vous devrez régulièrement le mettre à jour aussi vers une version plus récente pour pouvoir effectuer vos tests. C'est la même procédure que pour MetaModels, y compris pour la modification de l'entrée correspondante dans le fichier JSON avec « as 2.0.0 ».

Pour revenir à la version initiale, supprimez simplement le paquet par le gestionnaire de paquets.

> N'oubliez pas que vos retours de test sont précieux pour l'équipe de développement. N'hésitez pas à les faire sur Github.

1.2.4 Mettre à jour MetaModels

Si vous avez installé MetaModels via Composer, vous devez l'utiliser aussi pour effectuer les mises à jours.

Si vous avez installé MetaModels manuellement, vous devez prendre en compte certains points.

La procédure suivante s'est révélée la plus efficace :

- supprimez TOUS les anciens dossiers MetaModel (vous pouvez vérifier en comparant avec le précédent téléchargement) - vraiment **TOUS**
- Supprimez le cache de Contao -> /system/cache (tout ce qui est dans ce dossier)
- **NE FAITES SURTOUT PAS** de mise à jour de la base de données (database update) sous peine de perdre toutes vos données

— téléchargez la dernière nightly build, dézippez les fichiers et téléversez les (par FTP)
— mettez à jour la base par /contao/install.php
Vous trouverez les dernières informations sur le forum

1.2.5 Basculez MetaModels de la version « Nighty build » à la version « Composer »

La procédure est similaire à celle pour « mettre à jour MetaModels ». Si vous basculez sur Composer, prenez en compte que c'est une application très gourmande en mémoire. Par expérience, il vous faudra au moins 100 Mo. La taille mémoire exacte requise dépend des autres paquets installés ainsi que de la configuration du serveur chez votre hébergeur.

La procédure suivante s'est révélée la plus efficace :
— installez Composer
— supprimez TOUS les anciens dossiers de MetaModel (vous pouvez vérifier en comparant avec le précédent téléchargement) - vraiment **TOUS**
— Supprimez le cache de Contao -> /system/cache (tout ce qui est dans ce dossier)
— **NE FAITES SURTOUT PAS** de mise à jour de la base de données (database update) sous peine de perdre toutes vos données
— dans Composer, sélectionnez la version de MetaModels voulue, « Mark package to install » puis lancez l'installation
— la mise à jour de la base de données doit vous être automatiquement proposée : acceptez.
Vous trouverez les dernières informations sur le forum .

The first MetaModel Votre premier MetaModel ======================

Install with composer Installation via Composer ————————-

You'll need the MetaModels core and some attributes / filter to get MetaModels running. In you composer search `metamodels/core` and `metamodels/bundle_all` to install the core and all bundles and filters. Pour être fonctionnel, MetaModels nécessite le paquet principal (core) et des attributs et filtres. Dans Composer, recherchez `metamodels/core` et `metamodels/bundle_all` pour sélectionner le cœur et tous les paquets et filtres. Don't forget to run composer install through „Update packages". Lancez ensuite l'installation en cliquant syr „Mettre à jour les paquets". When installed, run the database update and your MetaModels installation is done. Enfin, lancez la mise à jour de la base de données et votre installation de MetaModels est terminée.

Note : If you know that you don't need all attributes and/or filter you can install every single package by it's own.

1.3 Create MetaModels

To get started with MetaModels we need at least one MetaModel, jai ! We will build a small MetaModel, non translated, MetaModel for real estate references.

In our example we need two MetaModels :

reference (the MetaModel which contain the real estate objects)

category (the MetaModel to define categories for references)

Create reference and category metamodels.

1.4 Create attributes

An (empty) MetaModel is just a container for your data objects. But before you can store data in your MetaModel, you need to define some types of data which you like to store.

In MetaModels there are several „attributes" to store different kind of data. Most of the time you need at least a text attribute (e.g. to store a name).

1.4.1 mm_reference

Our reference will contain these attributes :

- Name (text)
- Alias (alias)
- Published (checkbox)
- Description (longtext)
- Keyfacts (tabletext)
- Category (multiple select)
- Highlight-Picture (file)
- Picture Gallery (file, multiselect)

Name

Attribute Type text

Column Name name

Name Name

Description Name of reference

Alias

The alias is an (optional) unique Name / identifier for the data record.

Attribute Type alias

Column Name alias

Name Alias

Unique Yes

Description Alias of reference

Alias-Fields Name [text]

Published

Attribute Type checkbox

Column Name published

Name Published

Published yes

Description

Attribute Type longtext

Column Name description

Name Description

Description Description of reference

Keyfacts

Attribute Type tabletext

Column Name keyfacts

Name Keyfacts

Label Entry

Width 500

Category

Attribute Type multi select

Column Name category

Name Category

Description Select a category for the reference

Database table mm_category

Currently, we haven't added attributes to our `mm_category` MetaModel. So for the moment leave the other selects blank, we'll get back later.

Highlight picture

Attribute type file

column name picture_highlight

Name Highlight picture

Customize filetree (optional) select a „content" folder where the reference pictures are stored

Gallery

Attribute type file

column name picture_gallery

Name Gallery

Customize filetree (optional) select a „content" folder where the reference gallery pictures are stored

multiselect yes

1.4.2 mm_category

For our category MetaModel we just need four attributes :

— name (text ; „name")
— alias (alias ; „alias")
— published (checkbox ; „published")
— description (longtext ; „description")

Create the attributes as you have just learned in the reference MetaModel.

1.4.3 Select configuration

Early, we introduced in our „reference" MetaModel a select attribute but leaved it's configuration nearly blank.

The real power of MetaModel now gets obvious here. With a simple select attribute you can easily connect MetaModels (or any other sql-table) and optional filter the objects. Filter. . . ? We'll talk about this later.

Edit the „multi select" attribute in your „References".

Choose :

table mm_category

Name name - text

Alias alias - alias

Sorting sorting

1.5 Create Rendersettings

For now, we have two MetaModel with some attributes and a link between booth. But we didn't want just to store some data, we also like to display them.

A render setting contains some global settings, attributes you like to display and there settings. No matter if you like to display data in the backend or fronted you need at least one render setting. But we recommend to create at least one setting for the backend and one for the frontend.

Note : Prefix your render setting name with BE / FE for easy relocating*.

Basic-settings

Note : MetaModels provides a set of well organized, solid templates. There are templates for each render setting (e.g. metamodel_prerendered). You can create your own templates the contao why (Backend > Templates > Create > select the template you like to overwrite > Save (maybe with a new / name addition) > Edit > Choose)

-metamodel_prerendered All attributes are rendered with there template and settings (if available) -metamodel_unrendered All attributes are rendered in „raw“ to the frontend (the settings of the child attributes are

ignored)

Output Format :

-HTML 5 Renders as HTML5 content (This is the default format in Contao and therefore suggested). -XHTML Renders as xhtml (this format is deprecated in Contao and therefore not suggested). -Text Renders the „content“ as plaintext.

Jump-to-Page

The jump-to-page comes into the game when we like to display our references as list with a detail link to one item. So you need to define a jump-to-page where the user gets redirected if he clicks on a „detail“ link of one of our reference objects.

The filter setting define the rules for the target, your detail page.

Expert-settings

hide empty entries yes

hide labels yes

1.5.1 Create a rendersetting (backend)

Go to the „render settings“ of „reference“.
- Create a render setting called „BE : references“
- Add „all attributes“
- After adding, activate „name“ and „category“

Note : When you (later) add attributes to your MetaModel you need to add them also in your render setting.*

1.5.2 Create a rendersetting (frontend list)

Go to the „render settings“ of „reference“.
- Create a render setting called „FE : references list“
- Add „all attributes“
- After adding, activate „name“, „category“, „picture_highlight“

1.5.3 Create a rendersetting (frontend detail)

Go to the „render settings“ of „reference“.
- Create a render setting called „FE : reference detail“
- Add „all attributes“
- After adding, activate „name“, „description“, „category“, „picture_highlight“, „picture_gallery“

1.6 Input Screens

For now there are two MetaModels with some Attributes and Rendersetting. But how do we get data in our MetaModels ? With input screens !

Input Screens could hold a collection of these attributes which are necessary to grep some data. Most times you just add all attributes in one Input Screen, but with the power of different input screen you can create different edit masks for different kind of user(groups).

But in our tutorial we just need one input screen for our users.

Basic-settings

So create a Input Screen with the following settings :

Name BE : References

Standard yes

Panel-Layout -leave this empty-

Integration standalone

Backend-Section Content

Render mode Flat

Data manipulation permission We want to allow editing, creating and deleting items - so choose all three.

1.6.1 Select configuration

Okay. Now we got the empty Input Screen container with a few settings. But to get things working, we need (remember the render setting !) some attributes in it.

Switch to the „settings“ of your currently created Input Screen and choose „add all“.

1.6.2 Define Attribute settings

Our input screen is ready. But we need tweak the attributes a little bit. For example we always want a name, description and Highlight Picture.

To get this done, we choose in these attribute settings the „mandatory“.

1.6.3 Grouping and sorting settings

In the grouping & sorting section you need to create at least one object to sort & maybe group your entries.

For example : « Enable manual sorting » without grouping.

1.7 View conditions

View conditions are the easy part in MetaModels. But, you might guess that you also need here at least one to get things work.

The view conditions define who could see and use which render setting and input screen.

1.7.1 Define a view condition

Define one view condition with following settings :

member-group -leave this empty-
user-group administrator
input screen BE : Referenz
Rendersetting BE : Referenz

1.8 We are ready to enter Data

Some time ago, we started with just a MetaModels package and already arrived to create data. Easy, hm ?

Continue to the new „Referenz“ entry in your „content“ navigation and add a first item.

1.9 Filter Setting

(Todo)

CHAPITRE 2

Cookbook

2.1 MetaModels cookbook

In this cookbook you will find several snippets and example classes along with example configurations.

2.1.1 Filter cookbook

Dummy filter

If you are looking into writing excessive own extensions for MetaModels, please make yourself familiar with the base dc-general

CHAPITRE 3

Références

3.1 MetaModels API

The MetaModels API consists of serveral interfaces which are the only API that should be considered immutable. Classes of the core and their private, protected and even public methods should generally NOT be considered immutable and may be changed over minor versions and patch releases.

During the alpha and beta phase of a new MetaModels major release, there may be changes to interfaces as well. Therefore the API should not be considered immutable during major development cycles.

An deprecation phase will be provided during minor cycles, denoting that a certain feature of the API will get dropped in the next major release. We will try to put the replacement already in place but for bigger breaks this will not be possible. The breaks will however be announced in an draft, along with an upgrade guide, prior to release as soon as the new interfaces are defined.

3.1.1 Core Interfaces

3.2 MetaModels reference

This reference is mainly intended for developers that want to enhance MetaModels with own attributes and/or filters etc.

CHAPITRE 4

Indices and tables

— genindex