test Documentation

Version 0

ff

sept. 19, 2017

Table des matières

CHAPITRE 1

Présentation du projet

@TODO

CHAPITRE 2

License

— OpenSource - GPL V3
— Copyleft 2015 - Parc national des Cévennes

Installation et configuration

Prérequis

Ressources minimum serveur :

Un serveur disposant d'au moins de 1 Go RAM et de 10 Go d'espace disque.

Applications :
— postgresl (>= 9.3)
— postgis (>=2)
— apache
Autres :
— php-cli
— php-curl

Installation et configuration du serveur

Activer le mod_rewrite et les configurations requises pour symfony et redémarrer apache :

```
sudo a2enmod rewrite
sudo apache2ctl restart
```

Installation et configuration de PosgreSQL

Création de 2 utilisateurs PostgreSQL :

```
sudo su postgres
psql
CREATE ROLE simpleuser WITH LOGIN PASSWORD 'monpassachanger';
CREATE ROLE dbadmin WITH SUPERUSER LOGIN PASSWORD 'monpassachanger';
\q
```

Installation et configuration de la base de données

Créer un fichier de configuration de la base de données :

```
cd installation
cp db_settings.ini.sample db_settings.ini
```

Spécifier les bon paramètres de configuration de la base de données

Lancer l'installation :

```
cd installation
sudo ./install_db.sh
```

Installation et configuration de l'application

Configuration de symphony

@TODO

Droits sur les répertoires log et cache :

```
sudo chmod -R 777 app/cache app/log
```

Installation du composer :

```
curl -s https://getcomposer.org/installer | php
```

Mise à jour et téléchargement des dépendances :

```
php composer.phar update
```

Application serveur

Chaque nouvelle application doit être développée dans un bundle qui lui est propre

création d'un nouveau bundle :

```
app/console generate:bundle
```

Base de données

Chaque nouveau module doit définir son propre schéma.

La base de données contient un schema de base à utiliser pour définir ses données.

Les sites doivent utiliser la table suivi.pr_base_site, les informations complémentaires relatives aux protocoles sont à définir dans une table qui y est liée par une relation 1-1

les visites doivent utiliser la table doivent utiliser la table suivi.pr_base_visite, les infos complémentaires suivent la même regle que pour les sites.

Tables de références

layers.l_communes -> codes insee communes

taxonomie.taxref -> liste des taxons

lexique.t_thesaurus -> lexique

utilisateurs.t_roles -> utilisateurs

Entités

Mapping

Le mapping doit être fait en YAML

Il n'est pas recommendé de mapper les relations

Génération des entités :

```
app/console doctrine:generate:entities PNC
```

Modification des entités pour la gestion des erreurs

Les entités doivent hériter de la classe Utils/BaseEntity

```
use PNC\Utils\BaseEntity;

class MonEntite extends BaseEntity {
    ...
}
```

La vérification de la validité des données fournies à l'entité se fait dans les setters

```
function setMaVariable($variable){
    if($variable != "bonne valeur"){
        $this->add_error('maVariable', "la variable n'a pas la bonne valeur");
    }
    $this->maVariable = $variable;
}
```

Création des routes

5 routes par type de page

GET module/objet retourne une liste des objets

GET module/objet/{id} retourne un objet particulier identifié par {id}

PUT module/objet crée un nouvel objet

POST module/objet/{id} met à jour l'objet identifié par {id}

DELETE module/objet/{id} supprime l'objet identifié par {id}

Services utilisables dans les controleurs

dans les routes GET

récupérer une liste d'objets en utilisant EntityService

```
//GET monModule/monObjet
function getAllMonObjetAction(){
    $et = $this->get('entityService');
    $entite = 'monModule:MonObjet';
    $mapping = '../src/PNC/MonBundle/Resources/config/doctrine/monObjet.orm.yml';
    $results = $et->getAll($entite);
    $out = array();
    foreach($results as $result){
        $out[] = $et->normalize($result, $mapping);
    }
    return new JsonResponse($out);
}
```

la fonction présentée utilise le fichier yaml de mapping pour normaliser les objets.

Note : La normalisation d'un objet consiste à le transformer en dictionnaire (tableau associatif) directement sérialisable en JSON

Il est également possible de passer un tableau pour sélectionner les données que l'on souhaite récupérer

```
//...
foreach($results as $result){
    $out[] = $et->normalize($result, array(
        'maVar1'=>null,
        'maVar2'=>'date',
        //...
    ));
}
//...
```

le tableau prend en clé le nom de la variable, et en valeur une déclaration de fonction à utiliser pour transformer la donnée. la valeur *null* implique qu'aucune transformation n'est à faire.

Pour les données géométriques, EntityService permet d'organiser facilement les données pour le format GeoJSON

```
//...
$geoLabel = array(
    'label'=>'monObjet %s',
```

```
    'refs'=>array('id')
);
foreach($results as $result){
    $normalized = $et->normalize($result, array(
        'maVar1'=>null,
        'maVar2'=>'date',
        //...
    ));
    $out[] = $et->getGeoJsonFeature($normalized, $geoLabel, 'geom'); //geom est le␣
↪nom du champ qui contient la géométrie
}
```

récupérer une liste d'objets en utilisant le service Pagination

```
//GET monModule/monObjet
function getAllMonObjetAction(Request $request){
    $ps = $this->get('pagination');
    $entite = 'monModule:MonObjet';
    $mapping = '../src/PNC/MonBundle/Resources/config/doctrine/monObjet.orm.yml';
    $results = $ps->filter_request($entite, $request);
    $out = array();
    foreach($results as $result){
        $out[] = $et->normalize($result, $mapping);
    }
    return new JsonResponse($out);
}
```

récupérer un seul objet

```
//GET monModule/monObjet/{id}
function getOneMonObjetAction($id){
    $et = $this->get('entityService');
    $entite = 'monModule:MonObjet';
    $mapping = '../src/PNC/MonBundle/Resources/config/doctrine/monObjet.orm.yml';
    $results = $et->getOne($entite, array('id'=>$id));
    $out = $et->normalize($result, $mapping);
    return new JsonResponse($out);
}
```

créer un objet

```
//PUT monModule/monObjet
function createMonObjetAction(Request $request){
    $et = $this->get('entityService');
    $data = json_decode($request->getContent(), true);
    $mapping = '../src/PNC/MonBundle/Resources/config/doctrine/monObjet.orm.yml';
    $config = array($mapping => array(
            'entity' => new MonObjet(),
            'data' => $data
        )
    );
    try{
        $result = $et->create($config);
        $monObjet = $result[$mapping];
        return new JsonResponse(array('id'=>$monObjet->getId()));
    }
    catch(DataObjectException $e){
        return new JsonResponse($e->getErrors());
```

```
    }
}
```

mettre à jour un objet

```
//POST monModule/monObjet/{id}
function updateMonObjetAction(Request $request, $id){
    $et = $this->get('entityService');
    $data = json_decode($request->getContent(), true);
    $mapping = '../src/PNC/MonBundle/Resources/config/doctrine/monObjet.orm.yml';
    $entite = 'monModule:MonObjet';
    $config = array($mapping => array(
            'repo' => $entite,
            'data' => $data,
            'filter' => array('id'=>$id)
        )
    );
    try{
        $result = $et->update($config);
        $monObjet = $result[$mapping];
        return new JsonResponse(array('id'=>$monObjet->getId()));
    }
    catch(DataObjectException $e){
        return new JsonResponse($e->getErrors());
    }
}
```

supprimer un objet

```
//DELETE monModule/monObjet/{id}
function deleteMonObjetAction($id){
    $et = $this->get('entityService');
    $mapping = '../src/PNC/MonBundle/Resources/config/doctrine/monObjet.orm.yml';
    $entite = 'monModule:MonObjet';
    $config = array($mapping => array(
            'repo' => $entite,
            'filter' => array('id'=>$id)
        )
    );
    try{
        $result = $et->delete($config);
        $monObjet = $result[$mapping];
        return new JsonResponse(array('id'=>$monObjet->getId()));
    }
    catch(DataObjectException $e){
        return new JsonResponse(array(), 404);
    }
}
```

Configuration de l'appli cliente

Pour ne rien avoir à coder coté client, l'appli AngularJS propose des vues génériques qui déterminent l'url à contacter pour récupérer leur configuration en se basant sur leur propre url.

Par exemple *#/monModule/monObjet/list* contactera l'url *serveur.tld/monModule/config/monObjet/list*

Vues serveur pour la configuration du client

Déclaration du module pour l'application

En premier lieu, pour ajouter un module, il faut le déclarer dans *PNC/BaseAppBundle/Resources/clientConf/application.yml* :

```
- id: 1
  name: monModule
  base_url: "g/monModule/monObjet"
  img: "chemin/vers/image/appli" #non obligatoire
  appId: 150
  menu:
    -   url: "#/g/monModule/monObjet"
        label: "mon module"
        restrict: 1
```

ID : Numéro de module utilisé par l'appli cliente name : nom du module tel qu'il apparaitra dans la barre d'entête base_url : url de base du module vers laquelle l'utilisateur est renvoyé par défaut img : image d'accueil pour la sélection du module (non obligatoire, dans ce cas c'est "name" qui est affiché appId : identification de l'application dans userHub menu : liste des différents sous-modules (par ex le module chiro : Sites/Inventaire/Validation

> url : url de base du sous-module label : libellé du sous-module restrict : niveau de droit nécéssaire à l'utilisateur pour afficher le sous module.

Si le module ne comporte pas de sous modules, le menu reste obligatoire et dans ce cas il ne contient qu'un élément.

Schémas

Différents schémas à définir

pour chaque objet il faut définir au minimum 3 schémas :

- list
- detail
- form

Dans le cas d'utilisation des vues génériques, il est nécéssaire de créer une route pour chacun d'eux :

```
/monModule/config/monObjet/+leSchema
```

Controleurs config

```
//GET monModule/config/monObjet/list
function getMonObjetListSchema(){
    $cs = $this->get('configService');
    $schema = $cs->get_config('chemin/vers/mon/schema.yml')
    return new JsonResponse($schema);
}
```

Schema liste

configuration de base d'une vue générique

```
title: "monObjet"
emptyMsg: "Aucun monObjet"
createBtnLabel: "nouveau monObjet"
```

```
createUrl: "#/g/monModule/monObjet/edit"
editUrl: "#/g/monModule/monObjet/edit/"
detailUrl: "#/g/monModule/monObjet/detail"
dataUrl: "monModule/monObjet"
mapConfig: "fichier_config_map.json"
mapSize: large
editAccess: 1
```

- title : Titre de la page
- emptyMsg : texte affiché lorsqu'il n'existe aucune donnée
- createBtnLabel : libellé du bouton de création d'un nouvel objet
- createUrl : url angular du formulaire de création
- editUrl : url angular du formulaire de mise à jour
- detailUrl : url angular de la vue détaillée
- dataUrl : url serveur à contacter pour charger les données
- mapConfig : url du fichier (ou vue) de configuration des fonds carto - supprimer si pas de carte
- mapSize : taille de la carte (large|small)
- editAccess : niveau de droit nécéssaire pour éditer un objet

filtrage des données

```
filtering:
    limit: 200
    fields:
        -   name: ma_var_a
            label: "Var A"
            type: string
        -   name: ma_var_b
            label: "Var B"
            type: date
```

- filtering : définit les options de filtrage - le controleur qui renvoie les données doit alors utiliser paginationService plutôt que entityService
- limit : nombre maximum de données renvoyées par défaut - null équivaut à aucune limite
- fields : liste des champs qui permettent de filtrer les données - non obligatoire si aucun filtre n'a besoin d'être appliqué
- name : nom de l'attribut de l'objet à filtrer (au format mot_mot et non camelCase)
- label : libellé du filtre
- type : type de donnée : détermine les différents comparateurs

Note : Pour une utilisation des vues génériques, la directive de filtrage doit obligatoirement être déclarée.

liste des champs

```
fields:
    -   name: maVarA
        label: "Var A"
        type: text
    -   name: maVarB
        label: "Var B"
        type: date
    -   name: maVarC
        type: select
        thesaurusID: 1
```

- fields : liste des champs de l'objet à afficher

- name : nom du champ (format camelCase)
- label : libellé du champ (titre de la colonne)
- type : type de donnée
- thesaurusID : utilisable uniquement sur les champs select - cherche les lignes référent au chiffre fourni dans le lexique et complete le schéma avec les options de la liste déroulante

Schema detail

configuration de base d'une vue générique

```
dataUrl: "monModule/monObjet/"
mapConfig: "fichier_config_map.json"
mapData: "monModule/mesDonneesmap"
mapSize: large
editAccess: 1
subEditAccess: 1
subSchemaUrl: "monModule/config/monSubObjet/list"
subDataUrl: "monModule/monSubObjet/monObjet/"
```

- dataUrl : url à contacter pour récupérer l'objet à afficher (complétée par l'appli angular avec ID passé en param de l'url)
- mapConfig : fichier de configuration des fonds carto (si omis, pas d'affichage carto)
- mapData : url des données carto (contexte de l'objet)
- mapSize : taille de la carte (large|small)
- editAccess : droits nécéssaires pour éditer la donnée
- subEditAccess : droits nécéssaires pour ajouter une sous donnée
- subSchemaUrl : adresse à contacter pour le schema de la liste des sous données
- subDataUrl : adresse pour charger les sous données

liste des champs

```
groups:
    - name: "monGroupe1"
      glyphicon: glyphicon-info-sign
      fields:
        -   name: maVarA
            label: "Var A"
            type: string
        -   name: maVarB
            label: "Var B"
            type: date
    - name: "monGroupe2"
      fields:
        -   name: maVarC
            label: "Var C"
            type: select
            thesaurusID: 1
```

- groups : liste de groupes de données - seront affichés sous forme de boites à onglets.
- groups.name : nom et libellé du groupe
- glyphicon : glyphicon décorative pour l'onglet - facultatif
- **fields : liste des champs affiché dans l'onglet**
  - name : nom de la variable (camelCase)
  - label : libellé
  - type : type de donnée

— thesaurusID : uniquement pour les types select - permet d'afficher le libellé correspondant à la valeur numérique du champ

schema formulaire

configuration de base d'une vue générique

```
editAccess: 1
deleteAccess: 1
formTitleCreate: "nouveau monObjet"
formTitleUpdate: "edition de "
formTitleRef: maVarA
createSuccessMessage: "monObjet créé"
updateSuccessMessage: "monObjet modifié"
deleteSuccessMessage: "monObjet supprimé"
formDeleteRedirectUrl: "g/monModule/monObjet/list"
formCreateCancelUrl: "g/monModule/monObjet/list"
```

— editAccess : droits nécéssaires pour éditer, en cas de droits insuffisant l'utilisateur est redirigé
— deleteAccess : droits nécéssaires pour faire apparaitre le bouton de suppression
— formTitleCreate : titre du formulaire de création d'un objet
— formTitleUpdate : titre du formulaire de modification (complété avec le contenu de formTitleRef
— formTitleRef : variable à utiliser pour compléter le titre du formulaire (cf ci dessus)
— createSuccessMessage : message affiché lorsqu'un objet est créé
— updateSuccessMessage : message affiché lorsqu'un objet est modifié
— deleteSuccessMessage : message affiché lorsqu'un objet est supprimé
— formDeleteRedirectUrl : url de redirection en cas de suppression de la donnée
— formCreateCancelUrl : url de redirection en cas d'abandon de création (en modification, l'url de redirection est la vue détaillée de l'objet)

```
groups:
    -   name: monGroupe1
        fields:
            -   name: maVarA
                label: "Var A"
                type: string
            -   name: maVarB
                label: "Var B"
                type: date
    -   name: monGroupe2
        fields:
            -   name: maVarC
                label: "Var C"
                type: select
                thesaurusID: 1
```

— **groups : liste des groupes de champs - affiché sous forme de boite à onglets avec sous validation (genre wizard)**

  — name : nom du groupe
  — **fields : liste des champs composant le groupe**
    — name : nom de la donnée (camelCase)
    — label : libellé du champ
    — type : type de champ

Définition des schémas

Liste des types de données qui peuvent être déclarés dans les différents schémas

Les options sont définies dans la variable “options” du champ

```
name: maVar
label: "Ma variable"
type: text
options:
    visible: true
default: "valeur par défaut"
```

Note : Certaines options peuvent être utilisées quelque soit le type de champ
- l’option **visible (bool)** rend la colonne de la liste visible par défaut.
- style (xl|l|s|xs) : définit la taille de la colonne

Configuration des filtres

Les filtres sont définis dans la variable “filter” du champ

```
name: maVar
label: "Ma variable"
type: text
filter:
    maVar: text
```

Des filtres spéciaux peuvent être utilisés en ajoutant la directive “filterFunc : nom_du_filtre”

```
name: maVar
label: "Ma variable"
type: text
filter:
    maVar: text
filterFunc: monFiltre
```

Il existe deux fonctions de filtres actuellement : starting : qui recherche une chaîne débutant par la recherche écrite (LIKE recherche%) integer : qui permet de rechercher un entier avec des comparaisons < = ou >

Types de données utilisables dans les listes

text

Affiche un texte sans la moindre transformation

```
name: maVar1
label: "Ma variable 1"
type: text
```

num

Affiche une donnée numérique sans transformation. Peut être utilisé avec la fonction de filtrage “integer”

select

Affiche un libellé correspondant à une valeur numérique sélectionné dans une liste fournie en options

Afin de faciliter l’utilisation de ce genre de type, lorsque la liste des libellés est issue du lexique, il est possible de faire une référence directe au lexique via *thesaurusID*

date

Le serveur renvoie des données brutes avec des dates au format YYYY-MM-DD. Les champs de type date transforment cette date au format DD/MM/YYYY

Types de données utilisables dans les formulaires

Note : l’option **required** est autorisée pour tous les types de champs afin de rendre la saisie obligatoire.

l’option **multi** permet de répéter un champ à volonté afin d’obtenir une liste plutot qu’une simple donnée. N’est pas disponible pour les champs de type *text*, *sum*, *geom*, *group* ou *file*

Note : Certaines options sont obligatoires en fonction du type de champ. Ces options sont signalées dans la description du type.

---

hidden

crée un champ de type <*input type=”hidden”*>. Ce champ accepte l’option “referParent : true” qui permet de faire référence au parametre d’identifiant passé par l’url ou l’option “ref : userId” qui permet de faire référence à l’ID de l’utilisateur.

string

affiche un champ de saisie du type <*input type=”text”*>

options :
- minLength : longueur minimum valide
- maxLength : longueur maximale autorisée

text

afficher un champ de saisie du type <*textarea*>

options :
- minLength : longueur minimum valide
- maxLength : longueur maximale autorisée

num

affiche un champ de type <*input type="number"*>

options :
- min : valeur minimum
- max : valeur maximum
- step : pas d'incrément pour l'incrémentation à la souris et pour l'activation des décimales.

sum

affiche un champ de type <*input type="number"*> dont la valeur est calculée en fontion d'autres champs *num*

options :
- min : valeur minimum
- max : valeur maximum
- step : pas d'incrément pour l'incrémentation à la souris et pour l'activation des décimales.
- ref : liste des champs servant de référence pour le calcul de la valeur
- modifiable (bool :true) : permet de mettre le champ en lecture seule.

bool

affiche une case à cocher type <*input type="checkbox"*>

select

affiche une liste déroulante dont les éléments sont passés en option

```
name: varX
label: "ma selection"
type: select
options:
    choices:
        -   id: 1
            libelle: "option 1"
        -   id: 2
            libelle: "option 2"
```

Note : Il est possible d'utiliser le raccourci *thesaurusID : num* au lieu de définir les différents choix pour les listes de sélection faisant référence au lexique

```
name: varX
label: "ma selection"
type: select
thesaurusID: 1
```

multisel

affiche une liste de choix sous forme de cases à cocher. La syntaxe est la même que celle du *select* hormis le type : *multisel*

date

affiche un champ date sous forme de calendrier

file

affiche une directive d'upload de fichier

options requises :
- target : dossier de stockage des fichier uploadés
- maxSize : "poids" maximum autorisé (en octets)
- accepted : liste des types d'extensions autorisés

autres options :
- unique : permet d'indiquer qu'il n'est possible d'uploader qu'un seul fichier.

xhr

affiche un champ de saisie du type *<input type="text">* pour les références avec autocompletion par appel au serveur

options requises :
- url : url à contacter pour obtenir les données d'autocompletion
- reverseurl : url à contacter pour obtenir le libellé lié à une référence

group

Le type group n'est pas un champ à part entière. Il permet de regrouper un nombre de champs en tableau de saisie

```
name: mesVars
type: group
titles:
    -   "colonne 1"
    -   "colonne 2"
fields:
    -   name: lig1
        label: "ligne 1"
        fields:
            -   name: l1c1
                label: "ligne1/colonne1"
                type: num
            -   name: l1c2
                label: "ligne1/colonne2"
                type: num
    -   name: lig2
        label: "ligne2"
        fields:
            -   name: l2c1
                label: "ligne2/colonne1"
                type: num
            -   name: l2c2
                label: "ligne2/colonne2"
                type: num
```

geom

affiche une carte pour la saisie des données géométriques

options :
- geometryType (point|linestring|polygon) : type de géométrie traçable
- dataUrl : url des données de contexte pour l'édition d'une géométrie

Note : Il est préférable que le champ geom soit dans un groupe dédié

subform

Permet de définir un groupe de champs répétables à la manière de sous formulaires.

```
name: subFormTest
label: "test subform"
type: subform
fields:
    -   name: sftest1
        label: "champ 1"
        type: string
        options:
            maxLength: 10
    -   name: sftest2
        label: "champ 2"
        type: num
    -   name: sftest3
        label: "champ 3"
        type: text
```

Types de données utilisables dans les vues détaillées

Note : L'option **multi** est utilisable pour tous les types de données pour afficher une liste

string

Affiche une donnée sans transformation.

bool

Affiche une donnée booléenne sous la forme Oui/Non

date

Affiche une date au format YYYY-MM-DD reformatée vers DD/MM/YYYY

xhr

Affiche le libellé d'une donnée par un appel au serveur

options requises :
— url : l'url à contacter pour obtenir le libellé correspondant à la donnée

select

Affiche un libellé sélectionné dans la liste passée en options

Note : Équivalent du type *select* disponible pour les formulaires, et se définit exactement de la même manière

multisel

Affiche une liste de libellés sélectionnés d'après la liste passée en options

A l'instar du type *select*, il se définit exactement de la même manière que pour les formulaires.

HOWTO - Créer un nouveau module

Etape 1 - Création du bundle et déclaration pour l'application cliente

Création du nouveau bundle

Répondre yml à la question concernant les choix de configuration

Répondre oui à toutes les autres questions.

correction du fichier *app/config/routing.yml* :

```
pnc_how_to:
    resource: "@PNCHowToBundle/Resources/config/routing.yml"
    prefix:   /howto/
```

Il est nécéssaire de déclarer le nouveau module à l'application cliente et le contrôleur qui permettra à celle ci de récupérer les fichiers de configuration des vues.

Déclaration du nouveau module à l'application cliente

Modification du fichier *PNC/BaseAppBundle/Resources/clientConf/application.yml* :

```
-   id: 2
    name: Howto
    base_url: "g/howto/howto/list"
    appId: 1000006
    menu:
        -   url: "#g/howto/howto/list"
            label: "Howto"
            restrict: 1
```

Etape 2 - Génération de la BDD

Création d'une table howto et insertions de données de test

```
CREATE SCHEMA howto;
CREATE TABLE howto.t_howto (
    id serial,
    ht_nom VARCHAR(100),
    ht_valeur INTEGER,
    ht_commentaire VARCHAR(1000)
);


--Ajout de l'application au système d'authentification

INSERT INTO utilisateurs.t_application (nom_application) values ('howto') RETURNING␣
↪id_application;
--valeur retournée : 1000006 (à noter utilisée plus tard pour la déclaration de l
↪'application cliente)

--en considérant qu'il y a un utilisateur Admin dont l'id_role = 1
INSERT INTO utilisateurs.cor_role_droit_application (id_role, id_droit, id_
↪application) VALUES (1, 6, 1000006);


-- insertion de données test
INSERT INTO howto.t_howto (ht_nom, ht_valeur, ht_commentaire) VALUES ('test1', 1,
↪'test1a'), ('test2', 2, 'test2b'), ('test3', 3, 'test3c')
```

Etape 3 - Création des mappings

3.1 Création du schéma

fichier *PNC/HowToBundle/Resources/config/doctrine/howto.orm.yml* :

```
PNC\HowToBundle\Entity\Howto:
    type: entity
    table: howto.t_howto
    schema: howto
    id:
        id:
            type: integer
            id: true
            generator:
                strategy: AUTO
    fields:
        ht_nom:
            type: text
        ht_valeur:
            type: integer
        ht_commentaire:
            type: text
```

Note : Les mappings étant réalisés pour une table existante, il est possible d'être un peu laxiste sur le typage des données.

Il est par contre nécéssaire de controler les données dans les mutateurs de la classe Entité générée.

3.2 Génération de l'entité

Dans une console :

```
app/console doctrine:generate:entities PNC
```

Note : Cette méthode régénere toutes les entités existantes dans l'application. Les modifications apportées aux entités régénérées ne sont cependant pas affectées.

3.3 Modification de l'entité générée

fichier *PNC/HowToBundle/Entity/Howto.php (condensé)* :

```
<?php
namespace PNC\HowToBundle\Entity;

use Doctrine\ORM\Mapping as ORM;
use PNC\Utils\BaseEntity;

class Howto extends BaseEntity{
    private $id;
    private $ht_nom;
    private $ht_valeur;
    private $ht_commentaire;

    //...
    public function setHtNom($nom){
        if(strlen($nom)>100){
            $this->add_error('htNom', 'La longueur doit être inférieure à 100␣
↪caractères');
        }
        $this->ht_nom = $nom;
    }
    //...
}
```

Cette modification permet d'utiliser la classe BaseEntity pour la gestion des erreurs.

Etape 4 - Création du contrôleur liste

4.0 Configuration de l'application cliente

Déclaration de la route pour le contrôleur dans le fichier PNC/HowToBundle/Resources/config/routing.yml :

```
howto_config:
    path: /config/howto/{view_name}
    defaults: { _controller: PNCHowToBundle:Default:config }
    requirements:
        _method: GET
```

Création du controleur :

```
public function configAction($view_name){
    $configs = array(
        'list'=>__DIR__ . '../Resources/clientConf/howto/list.yml',
        'detail'=>__DIR__ . '../Resources/clientConf/howto/detail.yml',
        'form'=>__DIR__ . '../Resources/clientConf/howto/form.yml',
    );

    // initialisation configservice
    $cs = $this->get('configService');

    if(isset($config[$view_name])){
        return new JsonResponse($cs->get_config($configs[$view_name]));
    }
    else{
        return new JsonResponse(array(), 404);
    }
}
```

4.1 Controleur

Ajout au fichier PNC/HowToBundle/Resources/config/routing.yml :

```
howto_list:
    path: /howto
    defaults: { _controller: PNCHowToBundle:Default:list }
    requirements:
        _method: GET
```

Création du controleur (fichier PNC/HowToBundle/Controller/DefaultController.php) :

```
<?php
namespace PNC\HowToBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

use Symfony\Component\HttpFoundation\JsonResponse;
use Symfony\Component\HttpFoundation\Request;

class DefaultController extends Controller{
    public function listAction(Request $req){
        // entité a charger
        $entity = 'PNCHowToBundle:Howto';

        // schéma utilisé pour la normalisation
        $schema = array(
            'id'=>null,
            'htNom'=>null,
            'htValeur'=>null
        );

        // initialisation des services
        $ps = $this->get('pagination');
        $es = $this->get('entityService');

        // requête
```

```
        $result = $ps->filter_request($entity, $req);

        // mise en forme du résultat
        $out = array();
        foreach($result['filtered'] as $item){
            $out[] = $es->normalize($item, $schema);
        }

        $result['filtered'] = $out;
        return new JsonResponse($result);
    }
}
```

À cette étape, l'url *appurl/howto/howto* doit renvoyer la liste des données sous forme de JSON.

4.2 Creation du controleur de configuration

Création du fichier de configuration *PNC/HowToBundle/Resources/clientConf/howto/list.yml* :

```
title: "howto"
emptyMsg: "Aucun howto enregistré"
dataUrl: "howto/howto"
editAccess: 6
createBtnLabel: "Nouveau howto"
createUrl: "#/g/howto/howto/edit"
editUrl: "#/g/howto/howto/edit/"
detailUrl: "#/g/howto/howto/detail/"
filtering:
    limit: null
fields:
    -   name: id
        label: ID
        type: text
        filter:
            id: text
        options:
            visible: false
    -   name: htNom
        label: "Nom"
        type: text
        filter:
            htNom: text
        options:
            style: xl
            visible: true
    -   name: htValeur
        label: "Valeur"
        type: text
        filter:
            htValeur: text
        options:
            style: xl
            visible: true
```

À cette étape, l'url *appurl/#/g/howto/howto/list* doit afficher un tableau de données

Etape 5 - Création du contrôleur détails

5.1 Controleur

Ajout au fichier PNC/HowToBundle/Resources/config/routing.yml :

```
howto_detail:
    path: /howto/{id}
    defaults: { _controller: PNCHowToBundle:Default:detail }
    requirements:
        _method: GET
```

Création du controleur (fichier PNC/HowToBundle/Controller/DefaultController.php) :

```
<?php
namespace PNC\HowToBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

use Symfony\Component\HttpFoundation\JsonResponse;
use Symfony\Component\HttpFoundation\Request;

class DefaultController extends Controller{
    public function listAction(Request $req){
        //...
    }

    public function detailAction(Request $req, $id){
        // entité
        $entity = 'PNCHowToBundle:Howto';

        // schéma utilisé pour la normalisation
        // ici on utilise le fichier de mapping de l'entité puisqu'on
        // veut en récupérer toutes les données
        $schema =  '../src/PNC/HowToBundle/Resources/config/doctrine/Howto.orm.yml';
        // initialisation des services
        $es = $this->get('entityService');
        $data = $es->getOne($entity, array('id'=>$id));
        if($data){
            return new JsonResponse($es->normalize($data, $schema));
        }
        // objet inexistant
        return new JsonResponse(array(), 404);

    }
}
```

5.2 Creation du controleur de configuration

Création du fichier de configuration *PNC/HowToBundle/Resources/clientConf/howto/detail.yml* :

```
editAccess: 3
dataUrl: "chiro/obs_taxon/"
groups:
    -   name: "Général"
        fields:
```

```
            -   name: id
                type: hidden
            -   name: htNom
                label: "Nom"
                type: string
            -   name: htValeur
                label: "Valeur"
                type: num
    -   name: "Commentaires"
        fields:
            -   name: htCommentaire
                label: "Commentaire"
                type: string
```

Etape 6 - Création du contrôleur d'ajout

6.1 Controleur

Ajout au fichier PNC/HowToBundle/Resources/config/routing.yml :

```
howto_detail:
    path: /howto
    defaults: { _controller: PNCHowToBundle:Default:create}
    requirements:
        _method: PUT
```

Création du controleur (fichier PNC/HowToBundle/Controller/DefaultController.php) :

```
//ajouter avant la déclaration de classe
//use PNC\HowToBundle\Entity\Howto;

function createAction(Request $request){
    $et = $this->get('entityService');
    $data = json_decode($request->getContent(), true);
    $mapping =  '../src/PNC/HowToBundle/Resources/config/doctrine/Howto.orm.yml';
    $config = array($mapping => array(
            'entity' => new Howto(),
            'data' => $data
        )
    );
    try{
        $result = $et->create($config);
        $howto = $result[$mapping];
        return new JsonResponse(array('id'=>$howto->getId()));
    }
    catch(DataObjectException $e){
        return new JsonResponse($e->getErrors());
    }
}
```

6.2 Creation du controleur de configuration

Création du fichier de configuration *PNC/HowToBundle/Resources/clientConf/howto/form.yml* :

```
editAccess: 3
deleteAccess: 3
dataUrl: "howto/howto/"
createSuccessMessage: "Création d'un nouvel objet"
updateSuccessMessage: "Modification de l'objet réussie"
deleteSuccessMessage: "Suppression réussie"
formDeleteRedirectUrl: "g/howto/howto/list"
formCreateCancelUrl: "g/howto/howto/list"
groups:
    -   name: "Général"
        fields:
            -   name: id
                type: hidden
            -   name: htNom
                label: "Nom"
                type: string
                options:
                    minLength: 1
                    maxLength: 100
            -   name: htValeur
                label: "Valeur"
                type: num
    -   name: "Commentaires"
        fields:
            -   name: htCommentaire
                label: "Commentaire"
                type: text
                    maxLength: 1000
```

Etape 7 - Création du contrôleur de mise à jour

7.1 Controleur

Ajout au fichier PNC/HowToBundle/Resources/config/routing.yml :

```
howto_update:
    path: /howto/{id}
    defaults: { _controller: PNCHowToBundle:Default:update}
    requirements:
        _method: POST
```

Création du controleur (fichier PNC/HowToBundle/Controller/DefaultController.php) :

```
function updateAction(Request $request, $id){
    $et = $this->get('entityService');
    $data = json_decode($request->getContent(), true);
    $mapping =  '../src/PNC/HowToBundle/Resources/config/doctrine/Howto.orm.yml';
    $entity = 'PNCHowToBundle:Howto';

    $config = array($mapping => array(
            'repo' => $entity,
            'filter'=>array('id'=>$id),
            'data' => $data
        )
    );
    try{
```

```
        $result = $et->update($config);
        $howto = $result[$mapping];
        return new JsonResponse(array('id'=>$howto->getId()));
    }
    catch(DataObjectException $e){
        return new JsonResponse($e->getErrors());
    }
}
```

Note : L'application cliente utilise le même schéma pour la mise à jour que pour la création.

Etape 8 - Création du contrôleur de suppression

8.1 Controleur

Ajout au fichier PNC/HowToBundle/Resources/config/routing.yml :

```
howto_update:
    path: /howto/{id}
    defaults: { _controller: PNCHowToBundle:Default:delete}
    requirements:
        _method: DELETE
```

Création du controleur (fichier PNC/HowToBundle/Controller/DefaultController.php) :

```
function deleteAction(Request $request, $id){
    $et = $this->get('entityService');
    $mapping =  '../src/PNC/HowToBundle/Resources/config/doctrine/Howto.orm.yml';
    $entity = 'PNCHowToBundle:Howto';

    $config = array($mapping => array(
            'repo' => $entity,
            'filter'=>array('id'=>$id),
        )
    );
    try{
        $result = $et->delete($config);
        $howto = $result[$mapping];
        return new JsonResponse(array('id'=>$howto->getId()));
    }
    catch(DataObjectException $e){
        return new JsonResponse($e->getErrors());
    }
}
```