Documentation d’EDS Tools

|  |  |  |
| --- | --- | --- |
| Objet du document |  | Ce document décrit le produit EDS Tools |

|  |  |  |
| --- | --- | --- |
| Auteurs |  | Frédéric PEAK (fred@vaushell.com), pour VAUSHELL  Max PAVAGEAU (max@vaushell.com), pour VAUSHELL |
|  |  |  |
| Destinataires |  | Johann GRANDVUILLEMIN (johann.grandvuillemin@mpsa.com), PSA Peugeot-Citroën  Fabien VAUCHELLES (fabien@vaushell.com), VAUSHELL |
|  |  |  |
| Version |  | 2.3 |
|  |  |  |
| Date mise à jour |  | 12/12/2012 |
|  |  |  |

Sommaire

[1 Présentation générale 4](#_Toc343076877)

[1.1 Choix techniques 4](#_Toc343076878)

[1.2 Vue d’ensemble du modèle de données 4](#_Toc343076879)

[1.3 Architecture technique 5](#_Toc343076880)

[1.4 Organisation générale du code source 5](#_Toc343076881)

[2 Squelette de l’application 7](#_Toc343076882)

[2.1 La vue d’ensemble 7](#_Toc343076883)

[2.2 L’en-tête 7](#_Toc343076884)

[2.3 Le pied de page 8](#_Toc343076885)

[2.4 Le menu 8](#_Toc343076886)

[2.5 La barre d’action 9](#_Toc343076887)

[2.6 Le contenu 9](#_Toc343076888)

[2.7 L’administration 10](#_Toc343076889)

[3 Administrer l’application 12](#_Toc343076890)

[3.1 Gérer les libellés 12](#_Toc343076891)

[3.2 Gérer les modèles de fiche et les associations 14](#_Toc343076892)

[3.3 Gérer les fournisseurs/partenaires 16](#_Toc343076893)

[3.4 Gérer les rôles 18](#_Toc343076894)

[3.5 Gérer les utilisateurs 19](#_Toc343076895)

[3.6 Gérer les projets et la duplication 22](#_Toc343076896)

[3.7 Gérer les jalons d’un projet 23](#_Toc343076897)

[3.8 Gérer les partenaires du projet 25](#_Toc343076898)

[4 Rechercher des fiches 27](#_Toc343076899)

[4.1 Caractéristiques 27](#_Toc343076900)

[4.2 Implémentation 28](#_Toc343076901)

[5 Gérer une fiche 30](#_Toc343076902)

[5.1 Structure des formulaires 30](#_Toc343076903)

[5.2 Distinguer la visualisation et l’édition d’un formulaire 31](#_Toc343076904)

[5.3 Reconduire une fiche sans modifications 34](#_Toc343076905)

[5.4 Reconduire une fiche avec modifications 35](#_Toc343076906)

[5.5 Reconsulter une fiche 37](#_Toc343076907)

[5.6 Figer une fiche 38](#_Toc343076908)

[5.7 Abandonner ou rétablir une fiche 39](#_Toc343076909)

[5.8 Verrouiller les formulaires en écriture aux autres utilisateurs 40](#_Toc343076910)

[5.9 S’abonner à une fiche 41](#_Toc343076911)

[6 Définir le contenu des formulaires 44](#_Toc343076912)

[6.1 Gérer les données génériques 44](#_Toc343076913)

[6.2 Gérer les consommations de courant 44](#_Toc343076914)

[6.3 Gérer les tensions d’alimentations 45](#_Toc343076915)

[6.4 Gérer les défaillances veille réveil 45](#_Toc343076916)

[6.5 Gérer le formulaire CSE 45](#_Toc343076917)

[6.6 Gérer le comportement 45](#_Toc343076918)

[6.7 Gérer les profils de missions 45](#_Toc343076919)

[6.8 Gérer les profils d’activation 46](#_Toc343076920)

[6.9 Gérer les indicateurs de la fiche 47](#_Toc343076921)

[6.10 Gérer les fichiers joints 47](#_Toc343076922)

[7 Accéder aux données à travers un Webservices 48](#_Toc343076923)

[7.1 Caractéristiques 48](#_Toc343076924)

[7.2 Implémentation 48](#_Toc343076925)

[8 Accéder une page suivant une URL fixe 50](#_Toc343076926)

[8.1 Caractéristiques 50](#_Toc343076927)

[8.2 Implémentation 50](#_Toc343076928)

[9 S’authentifier 52](#_Toc343076929)

[9.1 Caractéristiques 52](#_Toc343076930)

[9.2 Implémentation 52](#_Toc343076931)

[10 Traduire en anglais et français 54](#_Toc343076932)

[10.1 Caractéristiques 54](#_Toc343076933)

[10.2 Implémentation 54](#_Toc343076934)

[11 Annexe : Génération des classes Hibernate du modèle 56](#_Toc343076935)

# Présentation générale

## Choix techniques

Le langage de programmation est JAVA. Ses avantages sont :

1. sa présence sur de nombreux systèmes comme Windows ou Linux ;
2. la disponibilité de nombreuses librairies libres et maintenues qui facilité le développement ;
3. la mise à disposition d’outil de développement performant comme Netbeans.

Ces avantages accélèrent le développement et facilitent la maintenance des logiciels.

EDS Tools se présentera sous la forme d’un client web riche, ce qui permettra d’éviter de déployer l’application sur les postes des utilisateurs.

|  |  |  |
| --- | --- | --- |
| **Caractéristique** | **Nom** | **Version** |
| Langage de développement | JAVA | 1.6.0\_24 |
| Environnement de développement | Netbeans | 7.0 |
| Analyseur performances & fuites mémoires | Netbeans Profiler | Intégré à Netbeans |
| Navigateurs supportés | Internet Explorer  Mozilla Firefox  Google Chrome |  |
| Format final de l’outil | Archive web (.war) |  |

## Vue d’ensemble du modèle de données

Le modèle de données de l’application est stocké dans une base de données, afin de pouvoir effectuer des recherches rapides sur certains critères de fiches.

Voici une vue générale du modèle de données de l’application :



## Architecture technique

L’outil utilise une architecture 3-tiers :

* Base de données – Application – Navigateur
* Base de données – Web services – Client externe



L’application est développée selon une architecture MVC (Modèle-Vue-Contrôleur).

## Organisation générale du code source

Le code source du projet est organisé en quatre packages principaux :

* com.inetpsa.eds.application
* com.inetpsa.eds.dao
* com.inetpsa.eds.tools
* com.inetpsa.eds.ws

### Le package application

Ce package contient le code source lié aux différentes fonctionnalités de l’application et de l’interface graphique. On y retrouve notamment le contrôleur de l’application et la fenêtre principale.

Il est séparé comme suit :

* actionbar : contient la barre d’action et ses boutons
* content : contient les interfaces du contenu affichable
  + admin : contient les formulaires des paramètres de l’administration
  + dashboard : contient les vues des menus tableau de bord et EDS
  + eds : contient les vues des formulaires d’une fiche EDS
  + home : contient la vue de la page d’accueil
  + project : contient les vues des menus projets
  + userparams : contient les formulaires des paramètres de l’utilisateur
* footer : contient l’interface du pied de page
* header : contient l’interface de l’en-tête
* login : contient la fenêtre de connexion
* menu : contient les éléments du menu
  + admin : contient les nœuds du menu d’administration
  + dashboard : contient les nœuds du tableau de bord
  + eds : contient les nœuds du menu EDS
  + edsnode : contient les nœuds des fiches EDS
  + project : contient les nœuds du menu projet
* popup : contient les différentes fenêtres popup utilisées dans l’application

### Le package dao

Ce package contient le code source lié au modèle de données de l’application. On y retrouve notamment le contrôleur DAO qui permet de communiquer avec la base de données via hibernate, ainsi que des interfaces utiles à des fonctionnalités globales telles que le pilote des dérives, le versionnage ou encore la reconduction d’un formulaire.

Le modèle de données se trouve dans le package « model ».

### Le package tools

Ce package contient le code source des :

* composants utilisés à de multiples endroits de l’application (tableaux de fiche EDS, double listes, doc-infos, etc…)
* outils liés à des fonctionnalités globales (filtrage, localisation, mail, gestion des URI, etc…)

### Le package ws

Ce package contient le code source lié au web service.

Il est séparé comme suit :

* model : contient les modèles des données échangées avec le webservice.
* resources : contient les ressources du webservice
* sample : contient des exemples d’utilisation du webservice

# Squelette de l’application

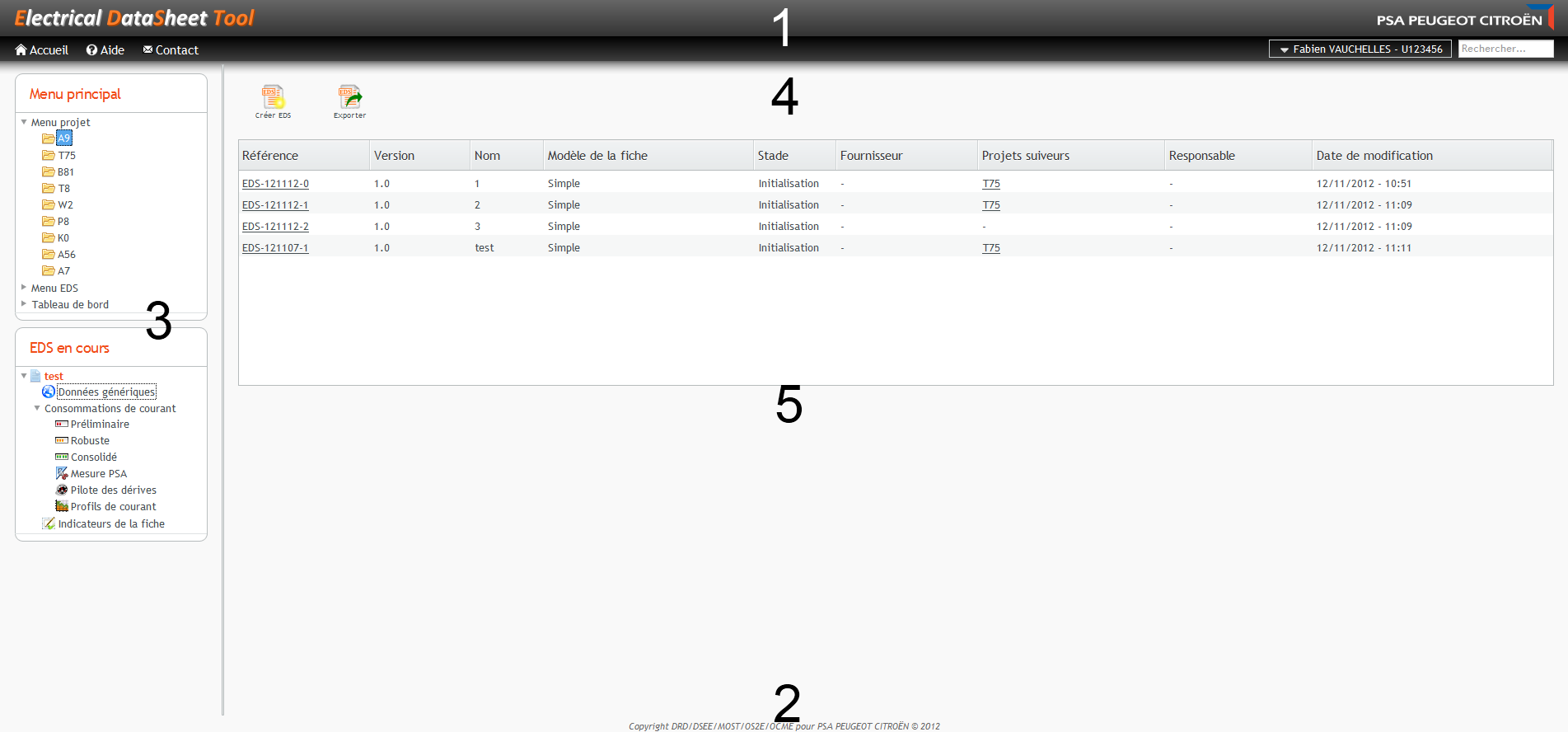
L’interface graphique est séparée en plusieurs parties. À chacune de ces parties correspondent un dossier qui se situe dans *com/inetpsa/eds/application*.

Du code CSS personnalisé est parfois associé à des éléments de la vue. Ce code se trouve dans le fichier « styles.css » du dossier *web/VAADIN/themes/psa*.

## La vue d’ensemble

La page principale de l’application est composée de différentes parties :

* un en-tête (1)
* un pied de page (2)
* un menu (3)
* une barre d’action (4)
* un contenu (5)



## L’en-tête



Les éléments qui composent l’en-tête de l’outil se trouvent dans le dossier « header ». Cet en-tête est composé de deux sous-parties :

* La partie supérieure affiche le nom de l’outil et le logo PSA, qui pointe sur le portail interne de PSA.
* La partie inférieure regroupe différents liens sur la gauche. Sur la droite se trouve le menu de l’utilisateur et le moteur de recherche une fois connecté.

## Le pied de page



Les éléments qui composent le pied de page de l’outil se trouvent dans le dossier « footer ». Ce pied de page est composé d’un texte informatif sur l’outil.

## Le menu

|  |  |
| --- | --- |
|  | Les éléments qui composent le menu se trouvent dans le dossier « menu ». Des listeners ont été rajoutés pour obtenir des comportements spécifiques, comme autoriser le pliage d’un seul menu à la fois.  Chaque élément du menu est un nœud héritant de A\_EdsNavigationNode. La méthode onEnter du nœud est appelée, ce qui peut par exemple permettre de mettre à jour le contenu de la page, ou les actions disponibles.  Chaque nœud peut également implémenter les interfaces suivantes :   * I\_Abortable pour abandonner une fiche * I\_Cancelable pour annuler l’édition du formulaire * I\_Closable pour fermer une fiche * I\_Discardable pour réinitialiser le formulaire d’édition * I\_Editable pour éditer une fiche * I\_Freezeable pour figer une fiche * I\_Reconductable pour reconduire une fiche * I\_Reconsultable pour reconsulter une fiche * I\_Savable pour sauvegarder l’édition d’une fiche * I\_Subscribable pour gérer l’abonnement aux fiches * I\_XmlExportable pour exporter une fiche ou un projet en xml   L’implémentation de ces interfaces permet de répondre aux événements sur les boutons de la barre d’action, que nous allons voir dans la partie suivante. |

## La barre d’action



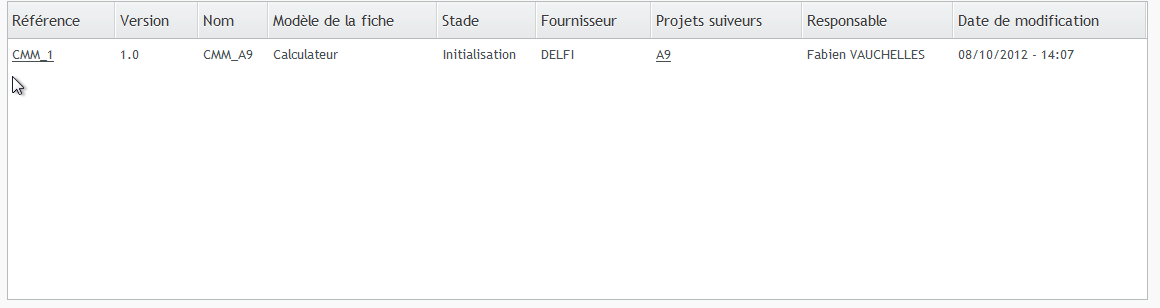
Les éléments qui composent la barre d’action se trouvent dans le dossier « actionbar ». Il s’agit d’une zone qui, lorsqu’elle est présente, propose à l’utilisateur d’effectuer des actions en cliquant sur des boutons. Les actions sont disponibles en fonction des droits de l’utilisateur, du nœud du menu sélectionné et de l’état de la fiche.

Chaque bouton est associé à une des interfaces listées dans la partie « Le menu ». Par exemple, le bouton « Modifier » utilise l’interface I\_Editable. Si le nœud courant implémente cette interface, la méthode « edit », spécifique à cette interface sera appelée.

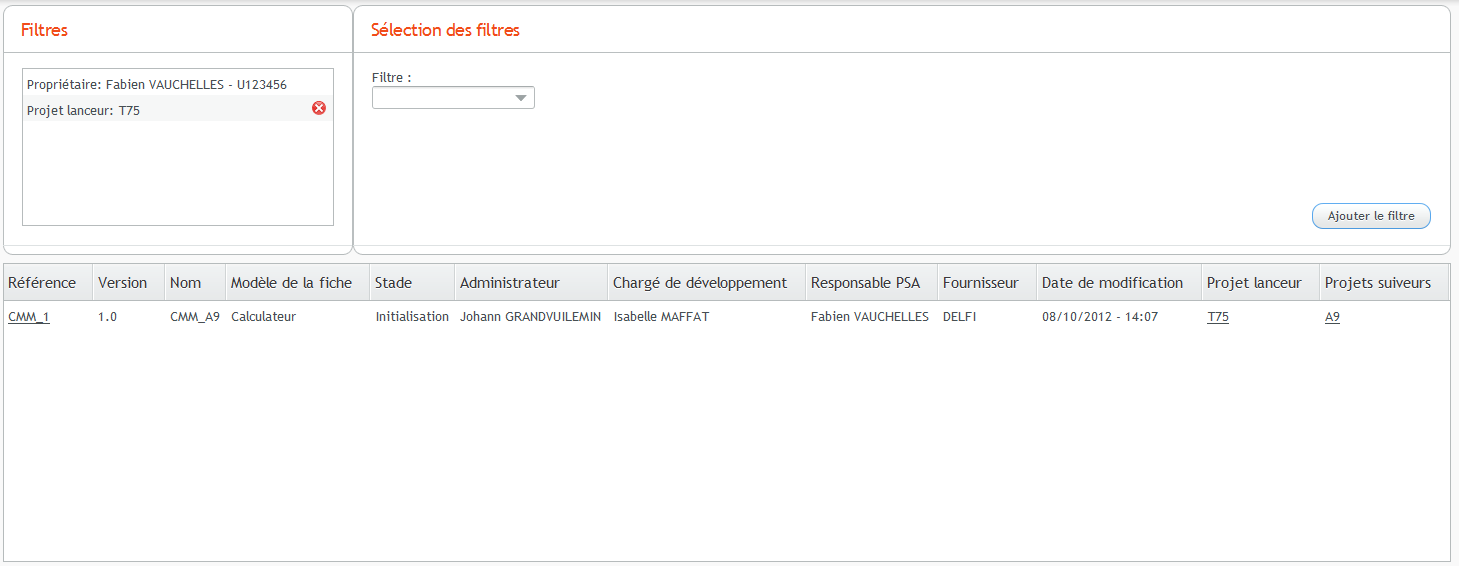
## Le contenu

Le contenu correspond à la zone située sous la barre d’actions. Elle permet d’afficher différentes informations en fonction du nœud sélectionné dans le menu et des actions effectuées. Ces vues d’informations sont regroupées dans le dossier « content ». Il s’agit par exemple de :

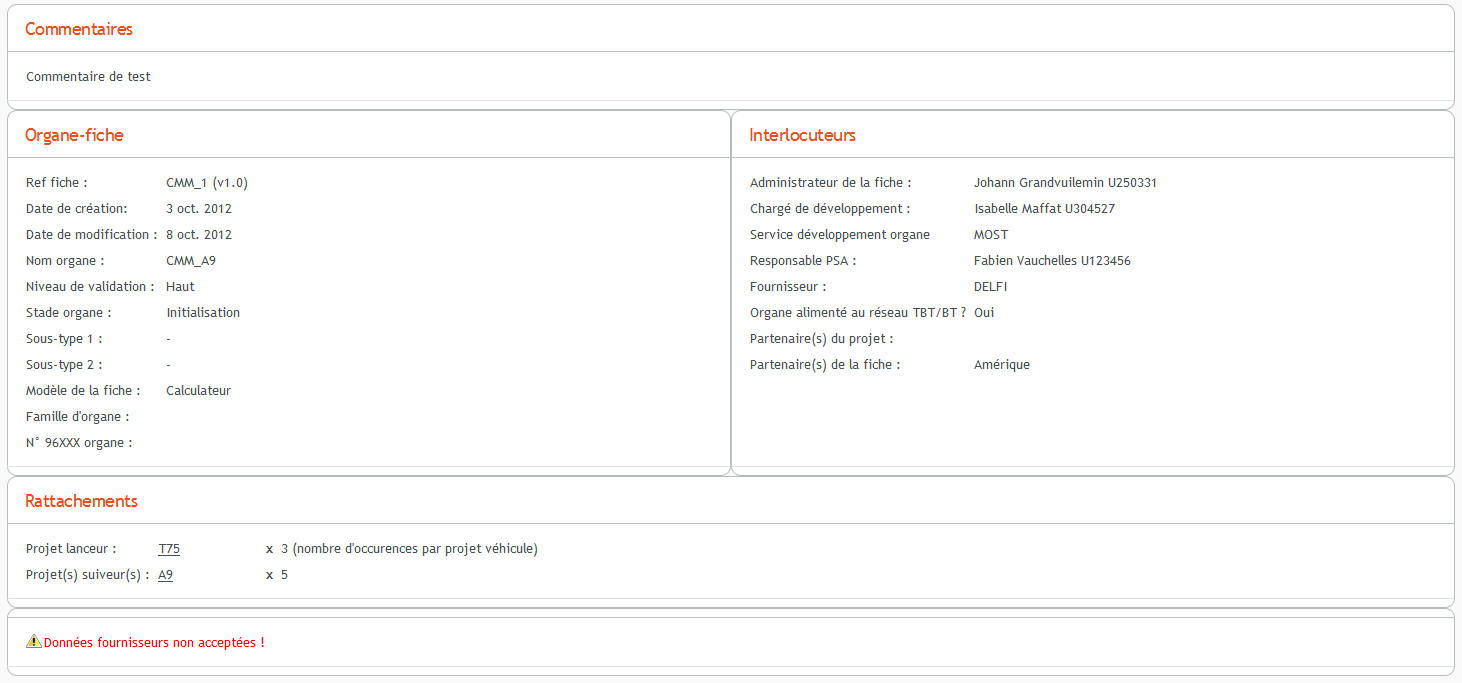
* la liste des fiches d’un projet



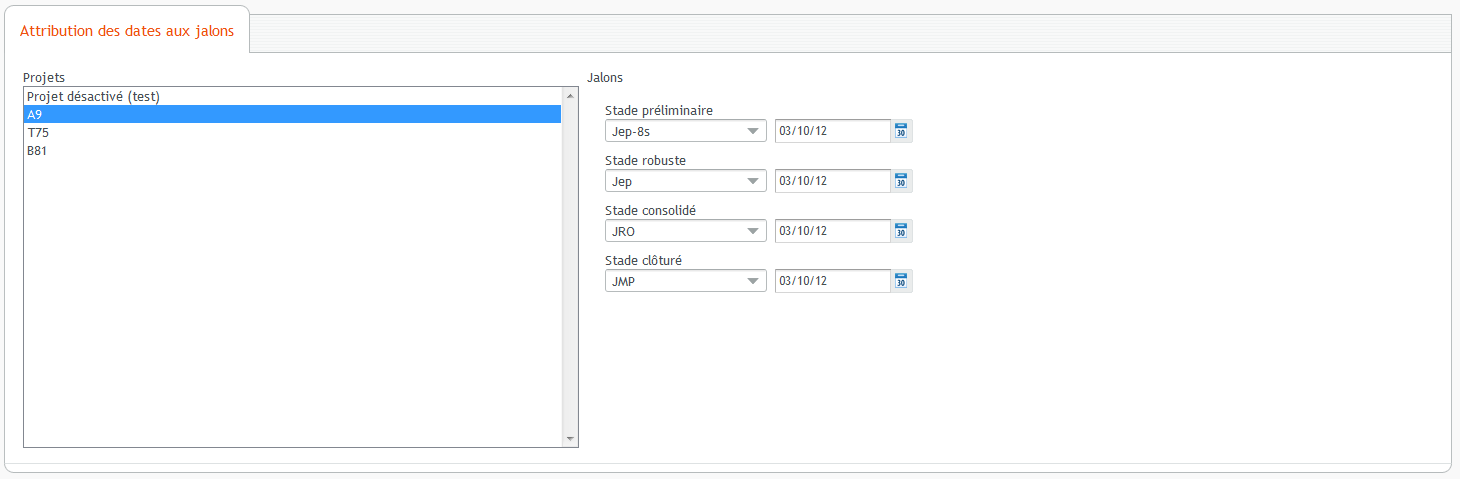
* la liste des fiches en fonction de certains critères de recherche



* le contenu d’un formulaire de fiche en lecture ou écriture

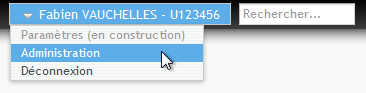


* des options de configuration de l’administrateur

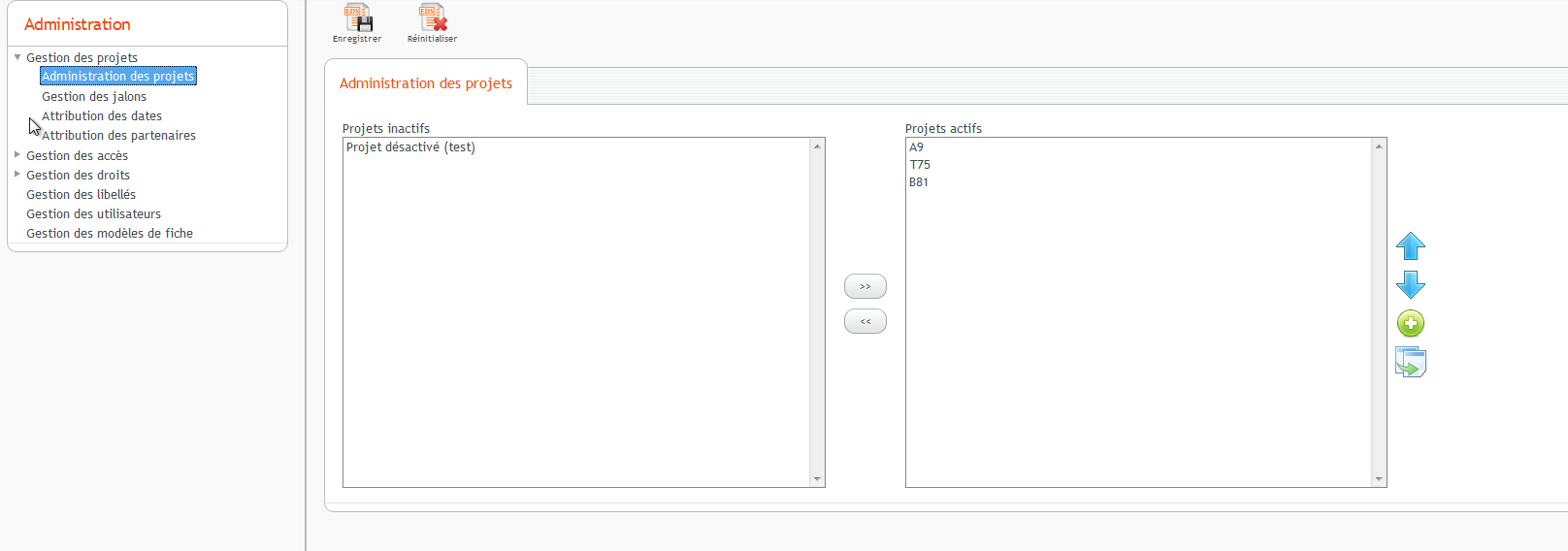


## L’administration

L’utilisateur, s’il en a les droits, peut accéder à la console d’administration via le menu de l’utilisateur.



La console d’administration possède un menu différent, mais fonctionne de la même façon que les formulaires, à la différence que seule une vue d’édition est présente. Les différents paramètres de l’outil peuvent être configurés.



# Administrer l’application

## Gérer les libellés

### Caractéristiques

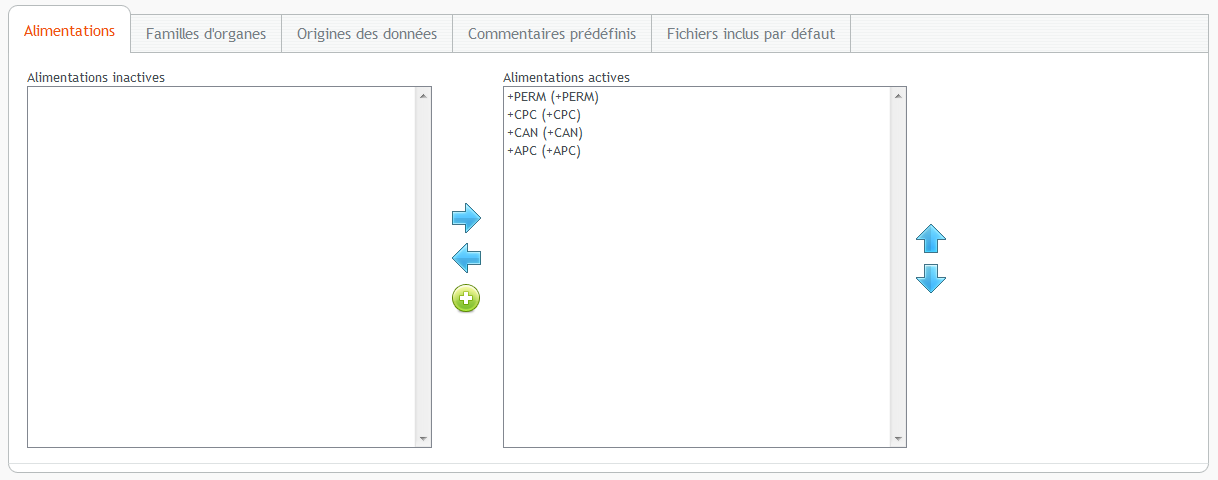
#### Description rapide

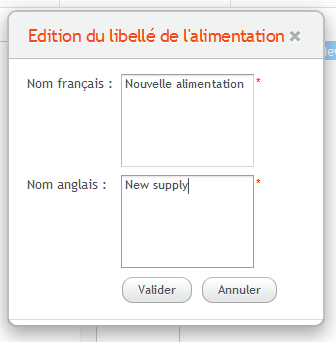
La gestion des libellés contient plusieurs sections permettant leur création, édition, traduction et désactivation. On retrouve les types suivants :

* Alimentations : permet de définir les types possibles d’une alimentation dans les consommations de courant
* Famille d’organes : permet de définir les familles d’organe possibles dans les données génériques.
* Origine des données : permet de définir les origines des données possibles dans les tensions d’alimentation.
* Commentaires prédéfinis : permet de définir les commentaires possibles lors de la validation des fiches.
* Fichier inclus par défaut : permet de définir les fichiers inclus par défaut lors de la création de fiches.

Le libellé correspond à la classe EdsWording.

#### IHM





#### Packages concernés

* com.inetpsa.eds.application.content.admin.wording

### Implémentation

#### Modèle de données



Remarque : l’index 0 indique une entrée inactive.

#### Diagramme de classes

Design pattern utilisé : Rien

#### Utilisation

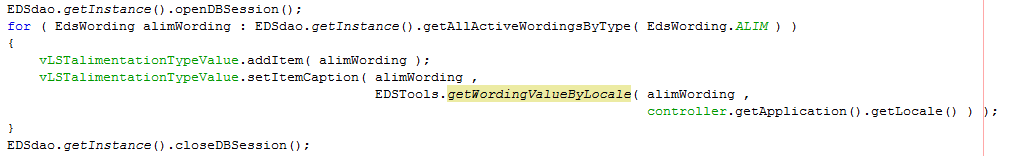
Les libellés servent à lister les différentes valeurs possibles pour certains champs de formulaire.

Dans le cas où le libellé peut être affiché en plusieurs langages, value contient les valeurs des différentes langues sous la forme « Langue : valeur, Langue : valeur…. »

Exemple pour un libellé de type ‘organ-family’ : ‘fra:ALARM;eng:ALARM’

#### Exemple de code source

Exemple de remplissage d’une liste avec les libellés d’alimentation.



## Gérer les modèles de fiche et les associations

### Caractéristiques

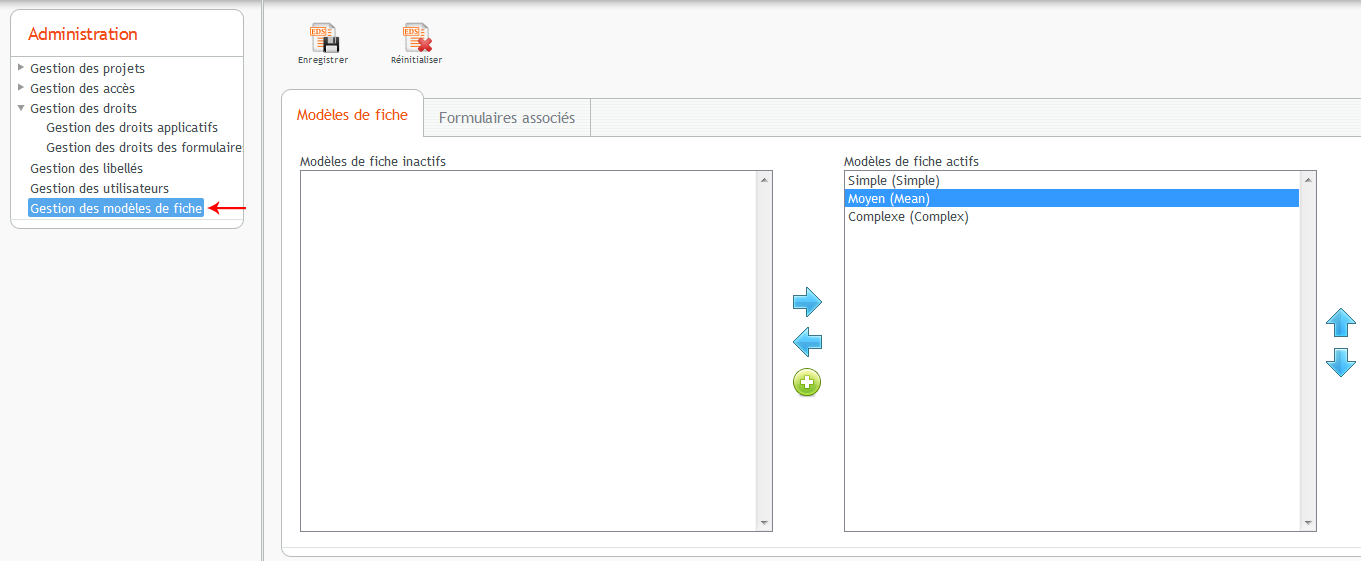
#### Description rapide

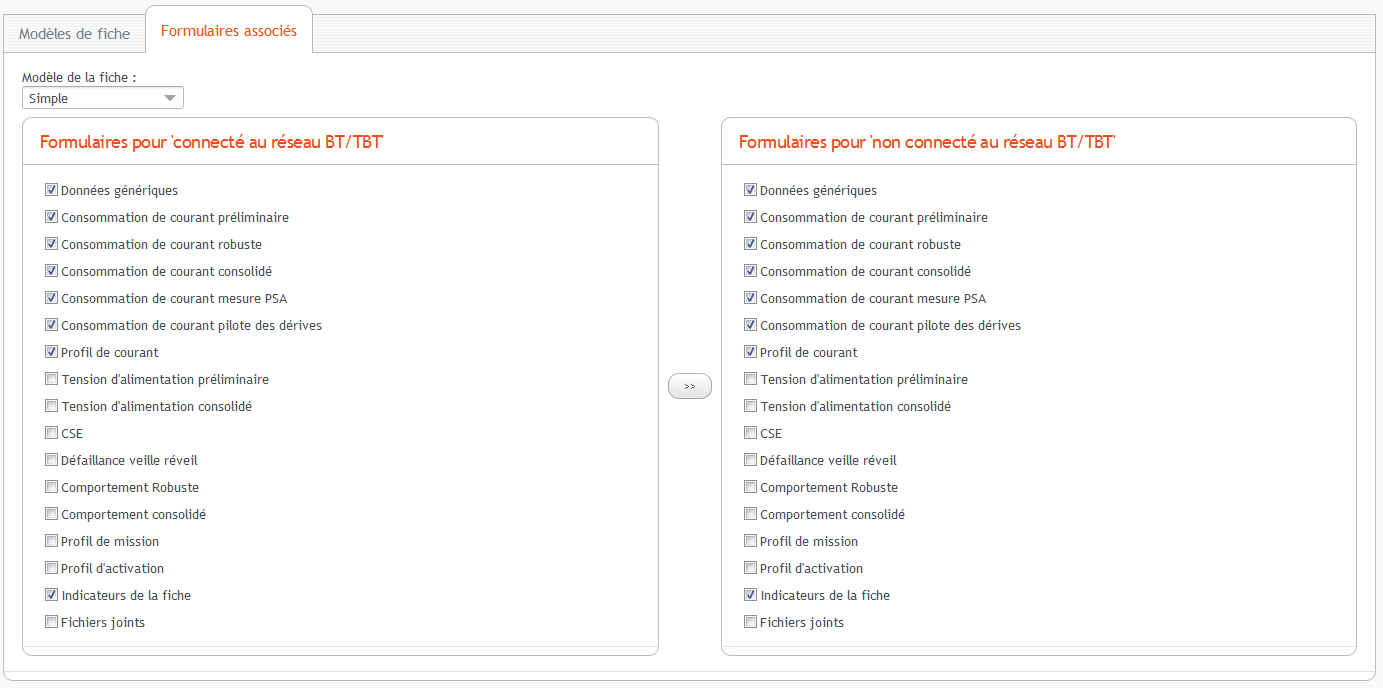
Les modèles de fiche consistent en une collection de formulaires accessibles afin d’éditer une fiche. Chaque modèle peut se distinguer sous deux formes : un modèle pour les fiches « connectées au réseau BT/TBT » et un modèle pour les fiches « non connectées au réseau BT/TBT ».

La gestion des modèles de fiche permet de :

* créer,
* activer / désactiver,
* renommer,
* traduire des modèles de fiche.

#### IHM





#### Packages concernés

com.inetpsa.eds.application.content.admin.componenttype

Les collections de formulaire sont utilisées dans com.inetpsa.eds.application.menu. EdsNavigationMenu dans la méthode addEdsNode( EdsEds eds ) pour créer les différents nœuds constituant la fiche.

### Implémentation

#### Modèle de données



CT\_BTTBT\_OK\_FORMSET contient une liste des identifiants pour les fiches connectées au réseau BT/TBT.

CT\_BTTBT\_KO\_FORMSET contient une liste des identifiants pour les fiches qui ne sont pas connectées.

Les collections de formulaire sont contenues sous forme d’une chaîne de caractère représentant les identifiants des formulaires séparées par des espaces.

#### Diagramme de classes

(Non rempli)

#### Diagramme de séquence OU Enchaînement des principales fonctions OU Algorithme

(Non rempli)

#### Exemple de code source

(Non rempli)

## Gérer les fournisseurs/partenaires

### Caractéristiques

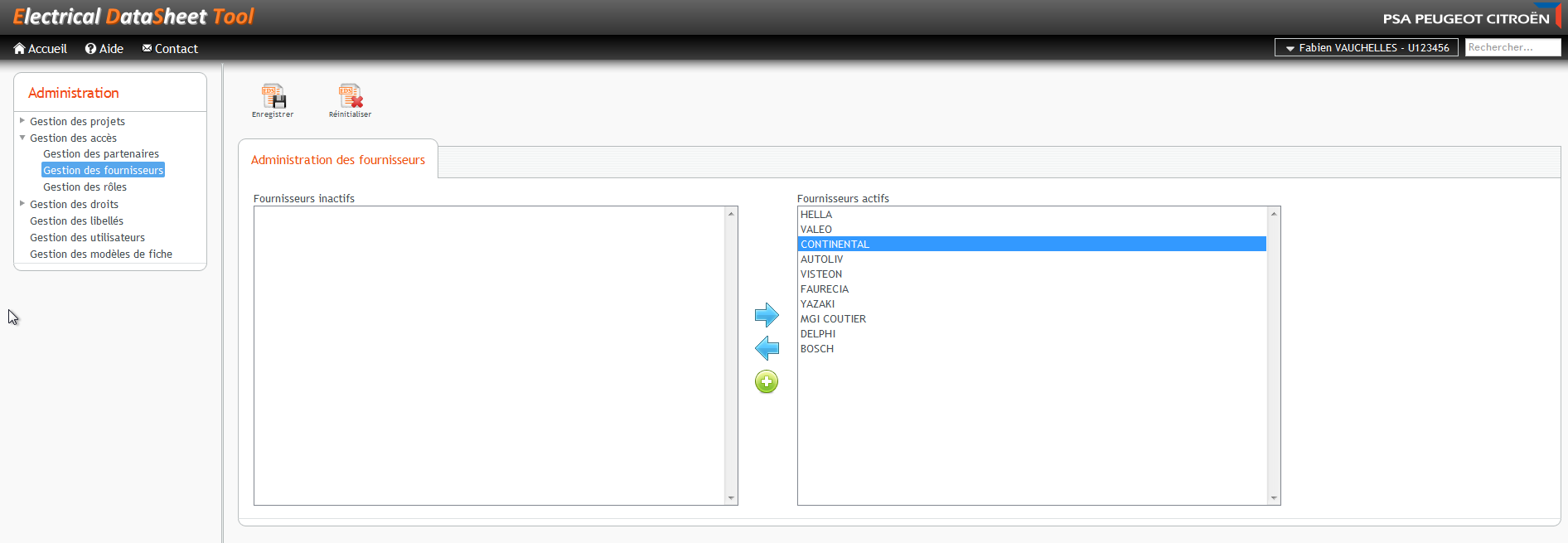
#### Description rapide

La gestion des fournisseurs et des partenaires comprend leur création, leur activation ou désactivation ainsi que leur nommage. Ces éléments servent de libellés.

On les retrouve dans les rôles des utilisateurs partenaires et fournisseurs, les périmètres des projets et des EDS, et les fournisseurs des EDS.

Ils sont utilisés pour la gestion des droits et des accès.

#### IHM



#### Packages concernés

* com.inetpsa.eds.application.content.admin.access.supplier
* com.inetpsa.eds.application.content.admin.access.perimeter

### Implémentation

#### Modèle de données

* Fournisseur



* Partenaire



#### Diagramme de classes

Design pattern utilisé :

* Rien

#### Utilisation

Les fournisseurs peuvent être associés à des utilisateurs et des EDS.

Les (utilisateurs) fournisseurs n’ont accès qu’aux EDS associés au même fournisseur.

Les partenaires peuvent être associés à des utilisateurs, des projets et des EDS.

Les (utilisateurs) partenaires n’ont accès qu’aux projets et EDS associés au même partenaire.

Des conditions spécifiques aux rôles de fournisseurs et partenaires limitent leur accès aux fiches : seules les fiches pour lesquelles ils sont respectivement fournisseurs et partenaires leur sont accessibles.

#### Exemple de code source

EdsApplicationController.userCanAccessEds() permet de vérifier si un utilisateur a accès à une fiche EDS.

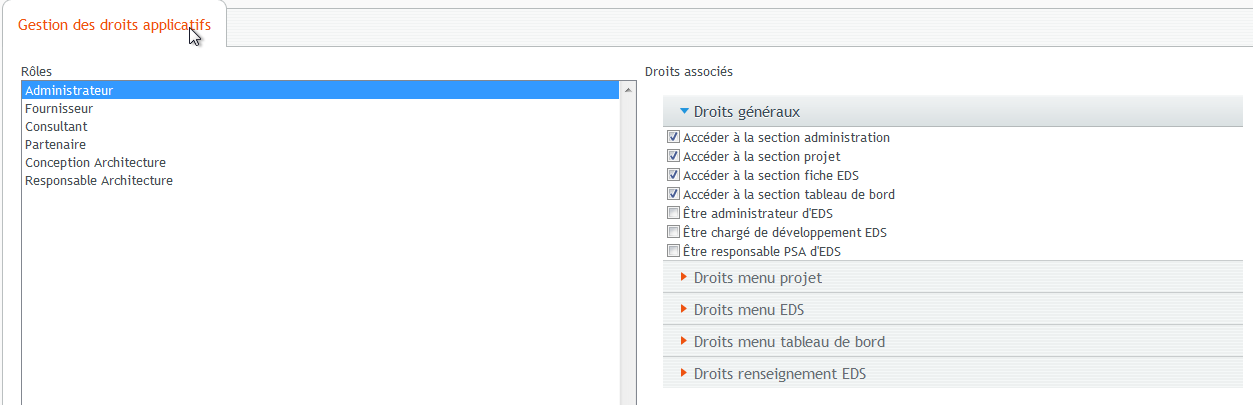
## Gérer les rôles

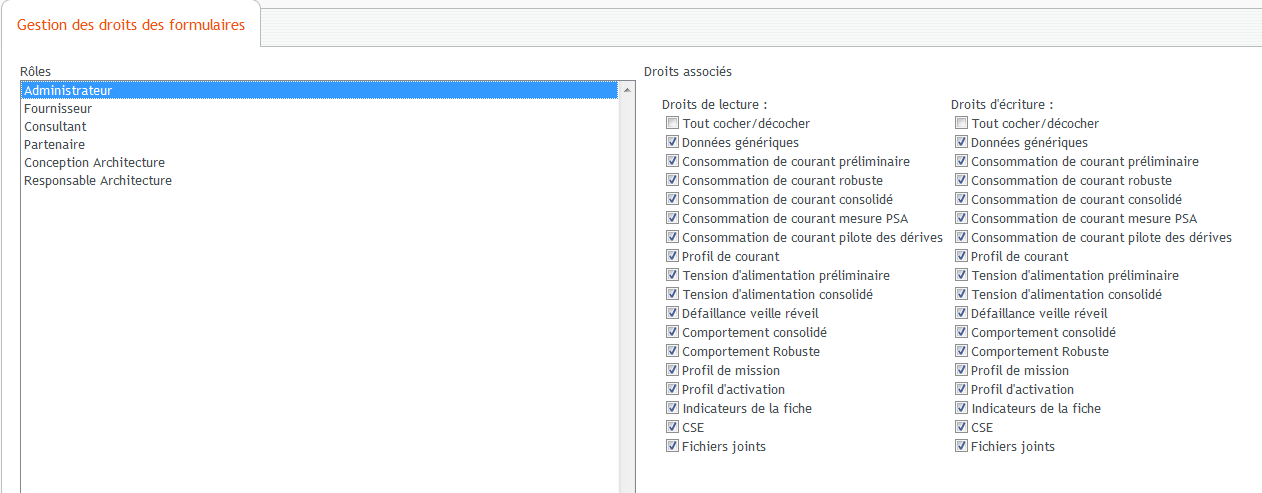
### Caractéristiques

#### Description rapide

Chaque utilisateur est associé à un rôle. Un rôle est constitué d’un ensemble de droits applicatifs et de droits en lecture/écriture des formulaires. Les droits de ces deux listes sont stockés dans la base de données sous forme d’un CLOB.

#### IHM





#### Packages concernés

* com.inetpsa.eds.application.content.admin.access.role

### Implémentation

#### Modèle de données



#### Diagramme de classes

Rien

#### Utilisation

La liste des droits existants se trouve dans la classe EdsRights.

Chaque droit est associé à un identifiant unique.

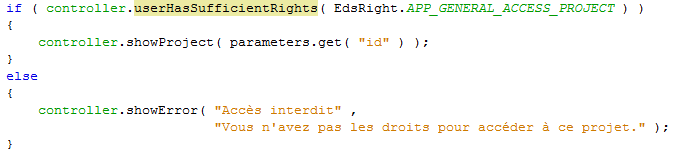
Un rôle possède la liste des identifiants des droits qu’il détient.

De plus, pour les droits applicatifs, cet identifiant est aussi utilisé dans les bundles qui servent à la localisation de l’application, afin de récupérer la description du droit dans la langue sélectionnée par l’utilisateur.

La méthode suivante est utilisée pour vérifier que l’utilisateur courant possède un certain droit : EdsApplicationController.userHasSufficentRights( String rightId ).

#### Exemple de code source

Affichage du contenu d’un contenu de projet si les droits sont suffisants.



## Gérer les utilisateurs

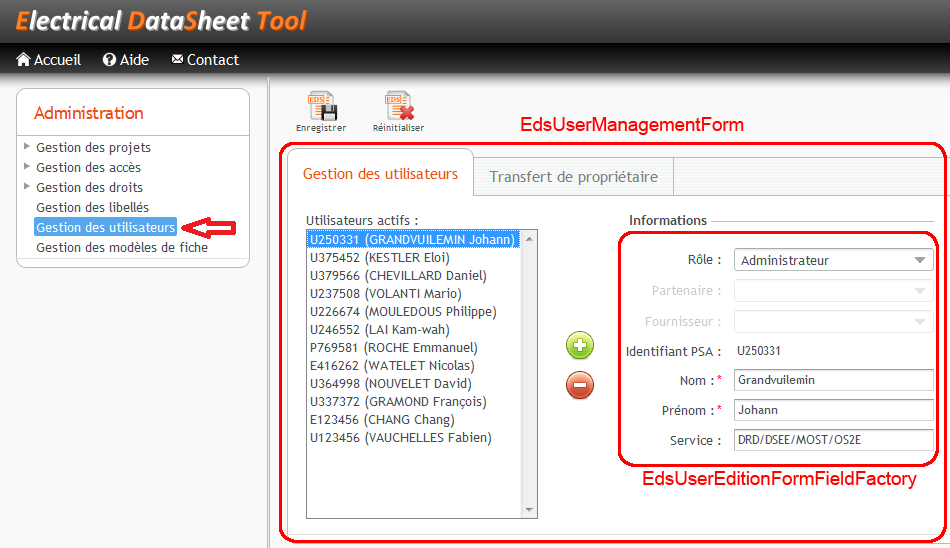
### Caractéristiques

#### Description rapide

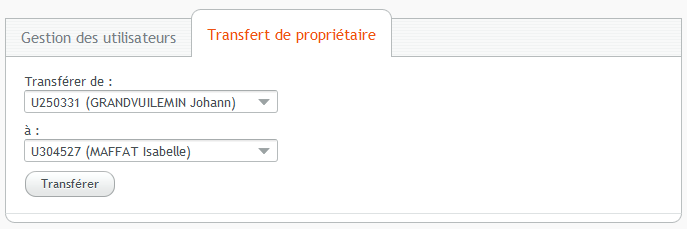
La gestion des utilisateurs dans la section d’administration de l’application contient 2 grandes fonctionnalités :

* « Gestion des utilisateurs » : fournit des outils pour la création/modification/suppression des utilisateurs, notamment l’identité de l’utilisateur ainsi que son rôle,
* « Transfert de propriétaire » : permet de modifier toutes les fiches dont le premier utilisateur est un propriétaire (i.e. soit administrateur de fiche, soit responsable PSA, soit chargé de développement) en y remplaçant cet utilisateur par le second.

#### IHM







#### Classes concernées

com.inetpsa.eds.application.content.admin.user

### Implémentation

#### Modèle de données



#### Diagramme de classes

(Non rempli)

#### Cas : Activation d’un utilisateur

Activer un utilisateur signifie rendre actif dans l’outil EDS un utilisateur présent sur le LDAP PSA. Une fois l’identifiant PSA entré, les données vont être automatiquement récupérées depuis le LDAP PSA (fonctionnalité encore absente).

#### Cas : Suppression d’un utilisateur

Supprimer un utilisateur ne le supprimera pas définitivement de la base de données, cela va simplement modifier son état de actif à inactif (U\_ACTIVE). Ainsi, on pourra toujours voir les données relatives à cet utilisateur (e.g. s’il est inscrit comme ayant modifié un champ dans un formulaire, il y sera toujours visible et une mention « utilisateur supprimé » y sera ajoutée).

#### Exemple de code source pour le transfert de propriétaire

Méthode utilisée :

EDSdao.shiftOwnership( fromUser, toUser );

## Gérer les projets et la duplication

### Caractéristiques

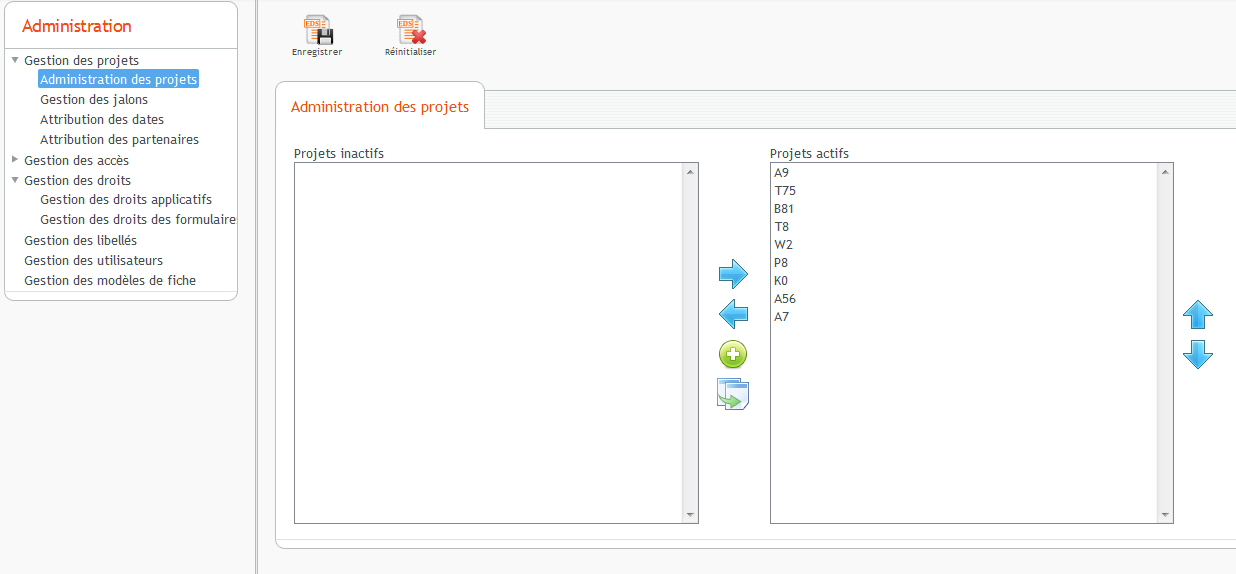
#### Description rapide

La gestion des projets comprend leur création, activation, désactivation, nommage, ordonnancement dans le menu utilisateur et duplication.

Les fiches EDS sont associées à un ou plusieurs de ces projets. Toute fiche a obligatoirement un projet lanceur. Il s’agit du projet à partir duquel la fiche est créée. Une fiche peut posséder des projets suiveurs suite à une reconduction.

La duplication permet d’effectuer des reconductions et reconsultations multiples de fiches EDS à partir d’un projet source vers un projet de destination.

#### IHM



#### Packages concernés

* com.inetpsa.eds.application.content.admin.project.administration

### Implémentation

#### Modèle de données



#### Diagramme de classes

* Rien

#### Utilisation

* Rien

#### Exemple de code source

* Rien

## Gérer les jalons d’un projet

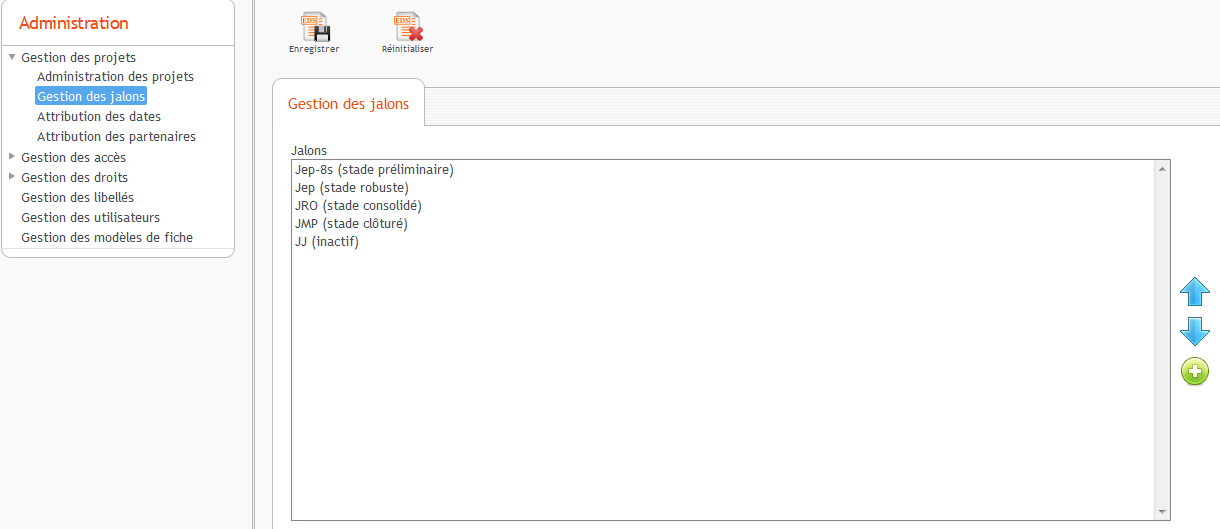
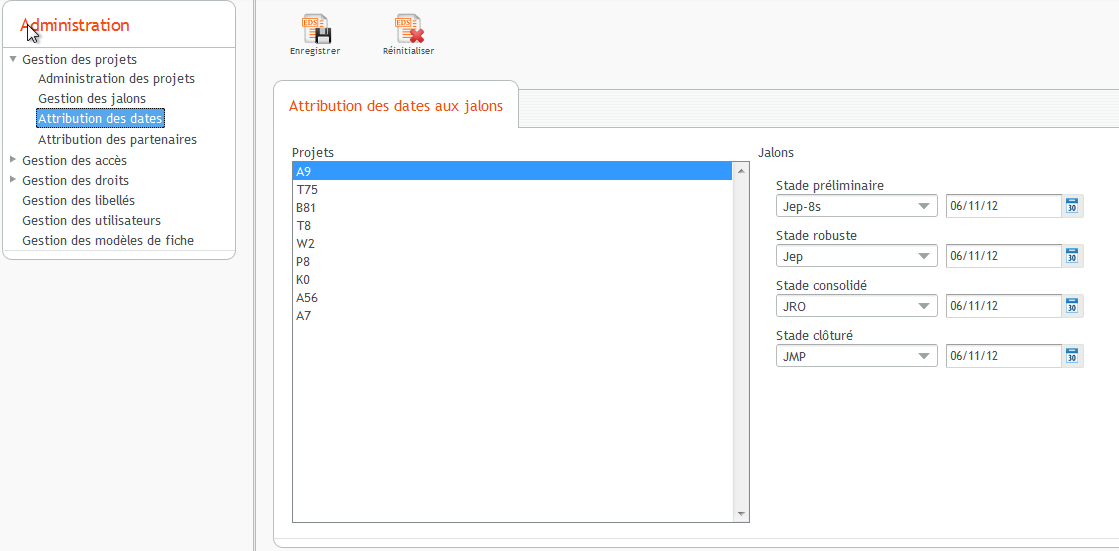
### Caractéristiques

#### Description rapide

La gestion des jalons comprend leur création, nommage et ordonnancement. Seuls les quatre premiers sont actifs. Ils correspondent respectivement aux jalons des stades préliminaire, robuste, consolidé et clôturé.

Pour chaque projet, on peut ensuite définir les jalons et les dates des différents stades.

#### IHM

#### Packages concernés

* com.inetpsa.eds.application.content.admin.project.milestones
* com.inetpsa.eds.application.content.admin.project.date

### Implémentation

#### Modèle de données

Il s’agit d’un libellé (voir la gestion des libellés), à la différence que value contient directement le nom du jalon.

#### Diagramme de classes

* Rien

#### Utilisation

Chaque projet actif est associé à quatre jalons et une date correspondante, un pour chaque stade.

Cette information est ensuite utilisée dans les chronogrammes des fiches d’un projet pour déterminer leur éventuel retard.

#### Exemple de code source

EdsChronogram utilise ces informations.

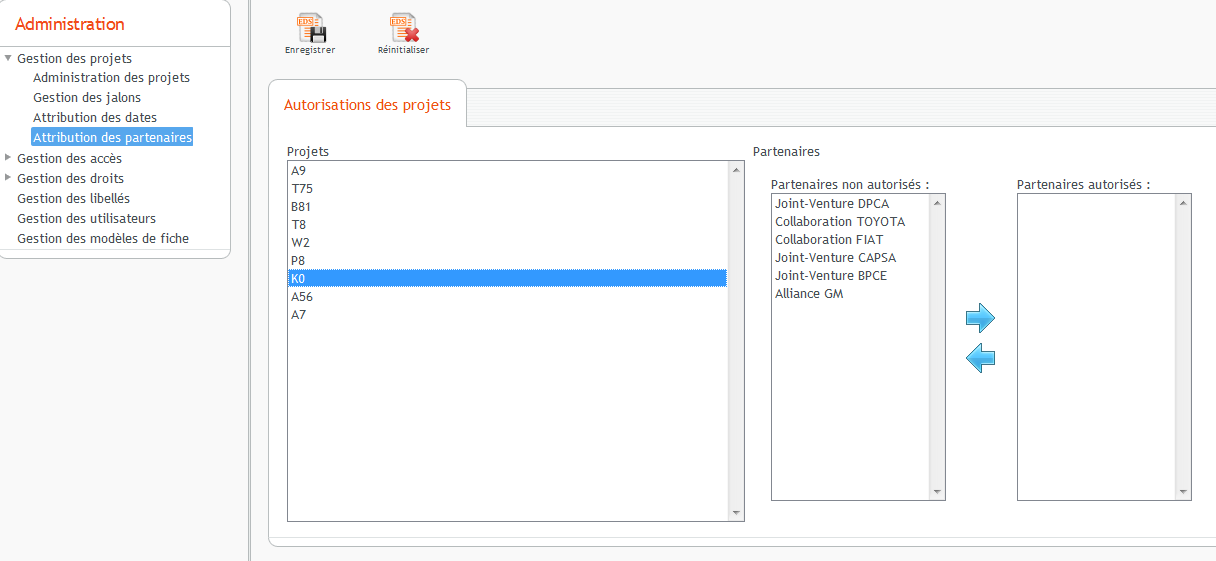
## Gérer les partenaires du projet

### Caractéristiques

#### Description rapide

La gestion des partenaires du projet consiste à définir quels partenaires ont l’autorisation d’accéder aux fiches du projet en question.

#### IHM



#### Packages concernés

* com.inetpsa.eds.application.content.admin.project.perimeter

### Implémentation

#### Modèle de données



#### Diagramme de classes

* Rien

#### Utilisation

EDSdao.isEdsIn( EdsEds eds , EdsPerimeter perimeter ) retourne si un partenaire peut accéder à la fiche.

#### Exemple de code source

* Rien

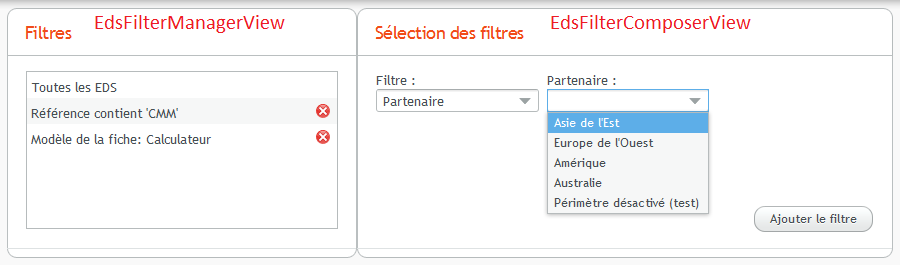
# Rechercher des fiches

## Caractéristiques

### Description rapide

Il est possible d’appliquer des filtres d’affichage des fiches dans les sous-menus des menus « Fiche EDS » et « Tableau de bord ». Pour cela, un éditeur de filtre est mis à disposition dans chacun de ces menus. L’affichage du gestionnaire de filtre est géré par la classe EdsFilterManagerView. En supplément du gestionnaire de filtres, il existe un composant de composition de filtre appelé EdsFilterComposerView.

### IHM



### Packages concernés

L’ensemble des composants et des filtres disponibles sont présents dans le package com.inetpsa.eds.tools.filter.

Composant d’affichage de la liste des EDS : com.inetpsa.eds.tools.table.EdsTable.

## Implémentation

### Modèle de données



Les filtres sont gérés par un gestionnaire appelé EdsFilterManager. Le gestionnaire de filtres possède une liste de filtres utilisés pour générer la requête servant à lister les fiches EDS à afficher à l’utilisateur.

Chaque filtre hérite de la classe abstraite A\_EdsFilter. Chaque filtre possède un identifiant unique permettant de ne pas inclure deux fois le même filtre (auquel cas des conflits sont possibles lors de la génération de la requête de listing des EDS). Par défaut, les filtres sont supprimables mais il est possible de les rendre fixes à l’aide de la méthode setRemoveable(boolean). Les filtres ne pouvant pas être supprimés n’ont pas de bouton de suppression (croix rouge) et la plupart sont uniquement utilisés par l’application pour certains sous-menus et inaccessibles à l’utilisateur.

Le compositeur possède une liste de factories permettant à l’utilisateur de construire et ajouter ses filtres au gestionnaire de filtres. Les factories (de classe A\_FilterFactory) ont un identifiant unique et possèdent une classe éditrice héritant de A\_FilterEditor. Les éditeurs doivent proposer une IHM d’édition du filtre et sont capables de créer des A\_EdsFilter à partir des inputs de l’utilisateur.

### Diagramme de classes

Design pattern utilisé :

L’EdsFilterManager suit un pattern builder. Il appelle chacun de ses filtres à la suite afin de construire la requête finale qui sera utilisée pour filtrer la liste des EDS visibles grâce à la méthode buildQuery() de la classe abstraite A\_EdsFilter.

La création des filtres répond à un pattern Abstract Factory grâce à la classe abstraite A\_FilterFactory.

### Enchaînement des principales fonctions

Pour mettre à jour l’affichage de la liste des EDS, il suffit d’appeler la méthode updateEdsList(). Celle-ci appellera la méthode buildQuery() du EdsFilterManager puis mettre à jour le tableau lié au manager.

Le champ de recherche rapide ouvre l’adresse : #search=recherche où recherche est la chaîne entrée par l’utilisateur encodée au format URL. Le lancement d’une recherche rapide ouvre le nœud Tableau de bord > toutes les fiches avec un filtre de recherche (filtre EdsSearchFilter).

### Exemple de code source

Pour ajouter le composant de filtrage des EDS, il suffit d’ajouter le composant EdsFilterPanel en lui passant le tableau d’affichage des EDS :

Layout container;

EdsTable edsListView = new EdsTable( edsApplicationController );

EdsFilterPanel filterPanel = new EdsFilterPanel( esdListView );

container.addComponent( filterPanel );

container.addComponent( edsListView );

# Gérer une fiche

## Structure des formulaires

### Caractéristiques

#### Description rapide

Les fiches EDS sont composées de données séparées en formulaires. La classe principale EdsEds contient les données du formulaire de données génériques. Tous les autres formulaires possèdent une référence vers cette classe. Leur implémentation est donc très libre.

#### IHM

* Rien

#### Packages concernés

* com.inetpsa.eds.application.content.eds

### Implémentation

#### Modèle de données

Formulaire principal (données génériques) :



Autres formulaires :



La table EDS\_FORM\_DATA représente les données du formulaire. Cette table doit aussi posséder une clé étrangère vers la fiche (table EDS\_EDS) qui la contient.

#### Diagramme de classes

Les formulaires sont accessibles grâce à un nœud ajouté sous le nœud de la fiche dans le menu de gauche « EDS en cours ». Ce nœud permet de consulter le formulaire sous deux formes : une forme de lecture affichée directement au clic sur le nœud du formulaire et une forme d’édition disponible en cliquant sur le bouton « Modifier » de la barre d’action du formulaire.

Chaque formulaire doit avoir un builder permettant de construire le nœud ainsi que les composants de lecture et d’édition du formulaire qui implémente l’interface I\_FormBuilder. Tous les builders doivent être listés dans la classe EdsFormFactory pour apparaître dans les nœuds du formulaire et dans les droits d’administration des formulaires.



Les nœuds implémentent la classe EN\_FormNode et contiennent les deux formes de formulaire :

* Formulaire de lecture
* Formulaire d’écriture

Tous les FormData doivent implémenter l’interface I\_FormData.

#### Diagramme de séquence OU Enchaînement des principales fonctions OU Algorithme

* Rien

#### Exemple de code source

Regarder EdsApplicationController.openEds( EdsEds eds )

## Distinguer la visualisation et l’édition d’un formulaire

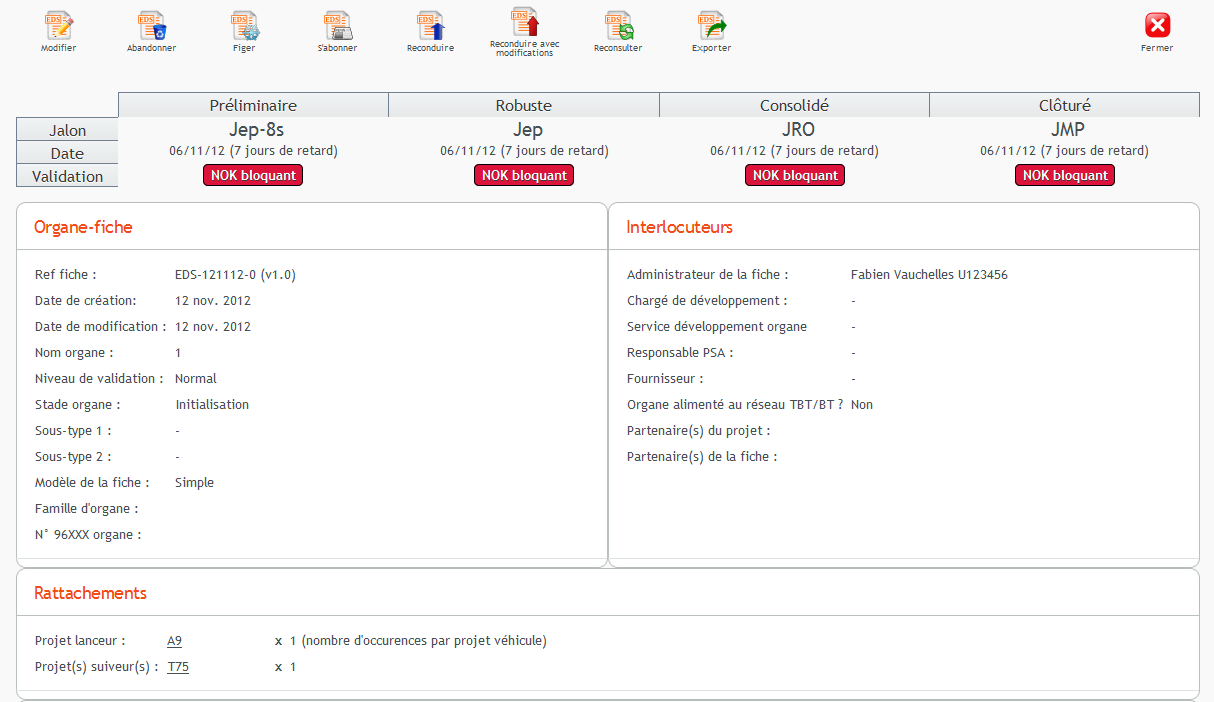
### Caractéristiques

#### Description rapide

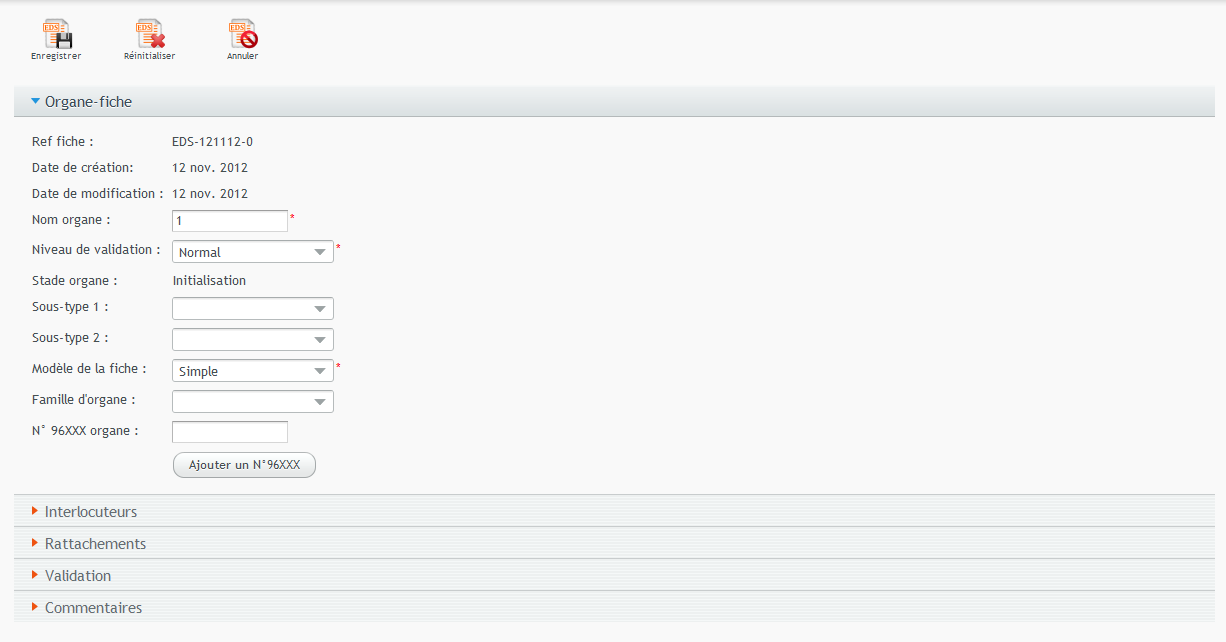
Chaque formulaire possède deux vues : la vue en lecture et la vue en écriture.

#### IHM

Vue en lecture du formulaire de données génériques :



Vue en écriture du formulaire de données génériques :



#### Packages concernés

* com.inetpsa.eds.application.content.eds

### Implémentation

#### Modèle de données

* Chaque vue en lecture hérite de A\_EdsFormReadView
* Chaque vue en écriture hérite de A\_EdsFormEditView



#### Diagramme de classes

Design pattern utilisé : Interface design pattern

Ce design pattern permet de manipuler toutes les vues de la même façon pour l’interaction avec l’utilisateur.

#### Principales fonctions

* Vue en lecture
  + Le formulaire de lecture permet d’afficher les informations présentes dans son modèle de données, qu’on appellera **FormData**.
  + Hérite de A\_EdsReadForm. Il faut notamment implémenter correctement :
    - getFormName : retourne le nom du formulaire qui sera affiché dans le menu de navigation,
    - refreshViewData : actualise l’affichage des données de l’IHM à partir du FormData.
    - getID : retourne l’identifiant unique du formulaire.
* Vue en écriture
  + Le formulaire d’écriture permet d’éditer les informations présentes dans le modèle de données.
  + Hérite de A\_EdsEditForm. Il faut notamment implémenter correctement :
    - isValid : vérifie que les données du formulaire sont valides. Cette méthode est appelée lors de la sauvegarde du formulaire.
    - commitChanges : modifie le contenu du FormData à partir des éléments de l’IHM.
    - discardChanges : actualise l’affichage des données de l’IHM à partir du FormData

getAllItemsToSave : retourne tous les objets à sauvegarder dans la base de données. Attention, l’ordre est important : si un objet A contient un objet B, l’objet B doit être sauvegardé avant A. (Le système est à simplifier plus tard)

#### Exemple de code source

Voir EN\_FormNode.

## Reconduire une fiche sans modifications

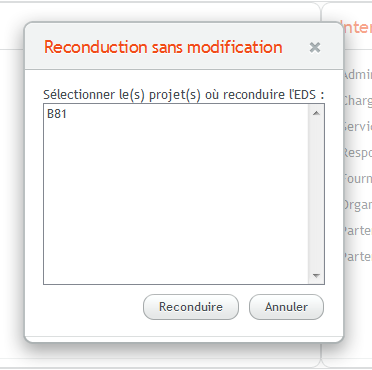
### Caractéristiques

#### Description rapide

La reconduction d’une fiche consiste à y ajouter un ou plusieurs projets suiveurs.

Lorsqu’elle est reconduite sans modification, seul un projet suiveur est ajouté. Les données spécifiques à ce projet seront communes à celles du projet lanceur.

#### IHM



#### Packages concernés

* com.inetpsa.eds.application.actionbar.reconduct

### Implémentation

#### Modèle de données



#### Diagramme de classes

* rien

#### Principales fonctions

EdsApplicationController.reconduct( EdsEds eds ,

List<EdsProject> projects ,

boolean withModif ,

boolean keepData ,

A\_EdsReadForm readForm )

#### Exemple de code source

* Rien

## Reconduire une fiche avec modifications

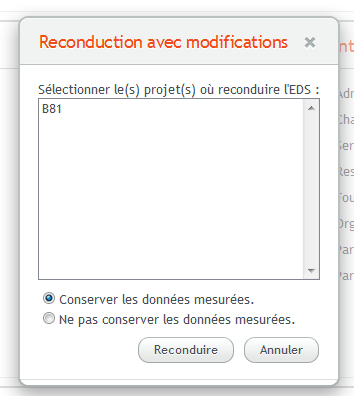
### Caractéristiques

#### Description rapide

La fiche peut être reconduite avec modifications. Cela signifie qu’en plus d’ajouter un ou plusieurs projets suiveurs, des données spécifiques à ce projet doivent être créées et leur être associés. Actuellement, seul tableau de consommation de cycle présent pour chaque alimentation dans le formulaire « Mesure PSA » est concerné.

#### IHM

Pour effectuer une reconduction avec modifications, l’utilisateur doit sélectionner le ou les projets vers lesquels il souhaite reconduire la fiche, et indiquer si les données doivent être conservées ou non.



#### Packages concernés

* com.inetpsa.eds.application.actionbar.reconduct

### Implémentation

#### Modèle de données



#### Diagramme de classes

La gestion des données spécifiques à un projet est déléguée aux modèles de données concernés. Ils doivent implémenter l’interface I\_ProjectSpecificFormData et implémenter les fonctions suivantes :

* addProjectReconductionData( EdsProject project, boolean keepData ) : crée des données spécifiques à project dans le formulaire.
* removeProjectReconductionData( EdsProject project ) : supprime les données spécifiques à project dans le formulaire.

#### Principales fonctions

EdsApplicationController.reconduct( EdsEds eds ,

List<EdsProject> projects ,

boolean withModif ,

boolean keepData ,

A\_EdsReadForm readForm )

#### Exemple de code source

* Rien

## Reconsulter une fiche

### Caractéristiques

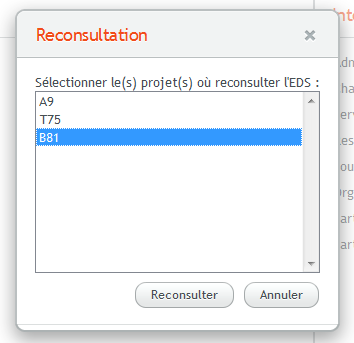
#### Description rapide

La reconsultation d’une fiche consiste à créer une nouvelle fiche, en reprenant certaines valeurs des formulaires de la fiche à reconsulter, à savoir :

* une partie des données génériques
* le formulaire de consommation de courant préliminaire
* le formulaire de consommation de courant robuste
* le formulaire de profil de mission

#### IHM

Comme pour les reconductions, l’utilisateur doit choisir le ou les projets de destination.



#### Packages concernés

* com.inetpsa.eds.application.actionbar.reconsult

### Implémentation

#### Modèle de données



#### Diagramme de classes

À l’exception des données génériques, les formulaires repris lors d’une reconsultation doivent implémenter l’interface I\_ReconsultableFormData.

* getReconsultableCopy( EdsEds eds ) : retourne une copie du formulaire avec les données à reconsulter préremplis.

#### Principales fonctions

EdsApplicationController.reconsult( EdsEds eds ,

EdsProject project )

#### Exemple de code source

* rien

## Figer une fiche

### Caractéristiques

#### Description rapide

Chaque fiche possède un numérotage de version composé de deux entiers indiquant respectivement la version majeure et la version majeure de la fiche qui possède la fonction suivante :

#### IHM



#### Packages concernés

com.inetpsa.eds.application.actionbar.freeze

### Implémentation

#### Modèle de données

(Non rempli)

#### Diagramme de classes

(Non rempli)

#### Algorithme

La version mineure de la fiche est incrémentée lors de la validation d’un stade d’une fiche via son formulaire de validation. La version majeure est quant à elle incrémentée lors d’un figeage de la fiche. Tout changement de version d’une fiche entraîne une copie de ses données.

Le figeage d’une fiche consiste à créer une copie exacte des données de la fiche dans son état actuel. Une nouvelle version sera attribuée à cette copie et elle deviendra la version actuelle. L’ancienne version sera définitivement figée en changeant son état de ‘normal’ à ‘figée’. Dès lors, l’ancienne version ne sera plus accessible qu’à partir du formulaire d’historique des versions (feature pas encore présente).

#### Exemple de code source

Méthodes DAO pour versionner une fiche :

dao.versionMinorEdsInDB( eds );

dao.versionMajorEdsInDB( eds );

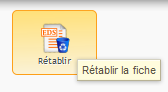
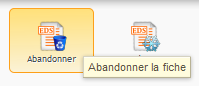
## Abandonner ou rétablir une fiche

### Caractéristiques

#### Description rapide

Les fiches EDS peuvent être mises en état d’abandon et, le cas échéant, être rétablies pour être réutilisées.

#### IHM



#### Packages concernés

com.inetpsa.eds.application.actionbar.abort

### Implémentation

#### Modèle de données

(Non rempli)

#### Diagramme de classes

(Non rempli)

#### Enchaînement des principales fonctions

EdsEds eds;

controller.abortEds( eds );

appelle

eds.setEdsState( EdsEds.ABORTED\_STATE );

puis

dao.update( eds );

#### Exemple de code source

Méthodes controller pour abandonner / rétablir une fiche :

EDSApplicationController.abortEds( EdsEds eds );

EDSApplicationController.unabortEds( EdsEds eds );

## Verrouiller les formulaires en écriture aux autres utilisateurs

### Caractéristiques

#### Description rapide

Lorsqu’un utilisateur demande l’édition d’un formulaire d’une fiche EDS, celui-ci prend la main sur ce formulaire et tout autre utilisateur désirant éditer ce même formulaire sera alerté par l’application qu’il doit attendre que l’utilisateur éditeur libère l’accès en édition. Si une fiche est en cours d’édition, les fonctionnalités lourdes (figeage, abandon, etc) ne sont plus possibles.

#### IHM

(Non rempli)

#### Classes concernées

com.inetpsa.eds.dao.AccessToken

com.inetpsa.eds.dao.E\_AccessLocked

com.inetpsa.eds.dao.EDSdao

Le verrouillage des formulaires est utilisé dans la classe com.inetpsa.eds.application.menu.edsnode.EN\_FormNode dans la méthode edit().

Le déverrouillage des formulaires est utilisé dans les méthodes cancel(), save() et onExit().

### Implémentation

#### Modèle de données



#### Diagramme de classes

(Non rempli)

#### Diagramme de séquence OU Enchaînement des principales fonctions OU Algorithme

(Non rempli)

#### Exemple de code source

Pour locker un formulaire, il faut procéder comme suit :

try

{

EDSdao.getInstance().lockAccess( user, edsId, formId );

// Lancer l’édition du formulaire

}

catch( E\_AccessLocked ex )

{

// L’accès est déjà locké par un autre utilisateur.

// C’est l’occasion de prévenir l’utilisateur.

// L’utilisateur ayant locké l’accès est accessible par

// ex.getUser().

}

## S’abonner à une fiche

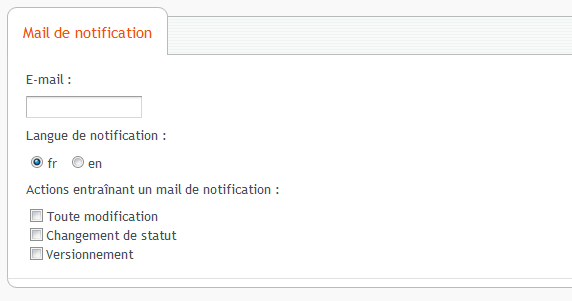
### Caractéristiques

#### Description rapide

L’abonnement a une fiche permet de recevoir des mails de notification suite à une modification de cette fiche.

#### IHM

Les paramètres de l’abonnement se trouvent dans le menu utilisateur.



#### Packages concernés

* com.inetpsa.eds.application.actionbar.subscribe
* com.inetpsa.eds.application.content.userparams
* com.inetpsa.eds.tools.mail

### Implémentation

#### Modèle de données

* Rien

#### Diagramme de classes

* (Non utilisé)

#### Diagramme de séquence

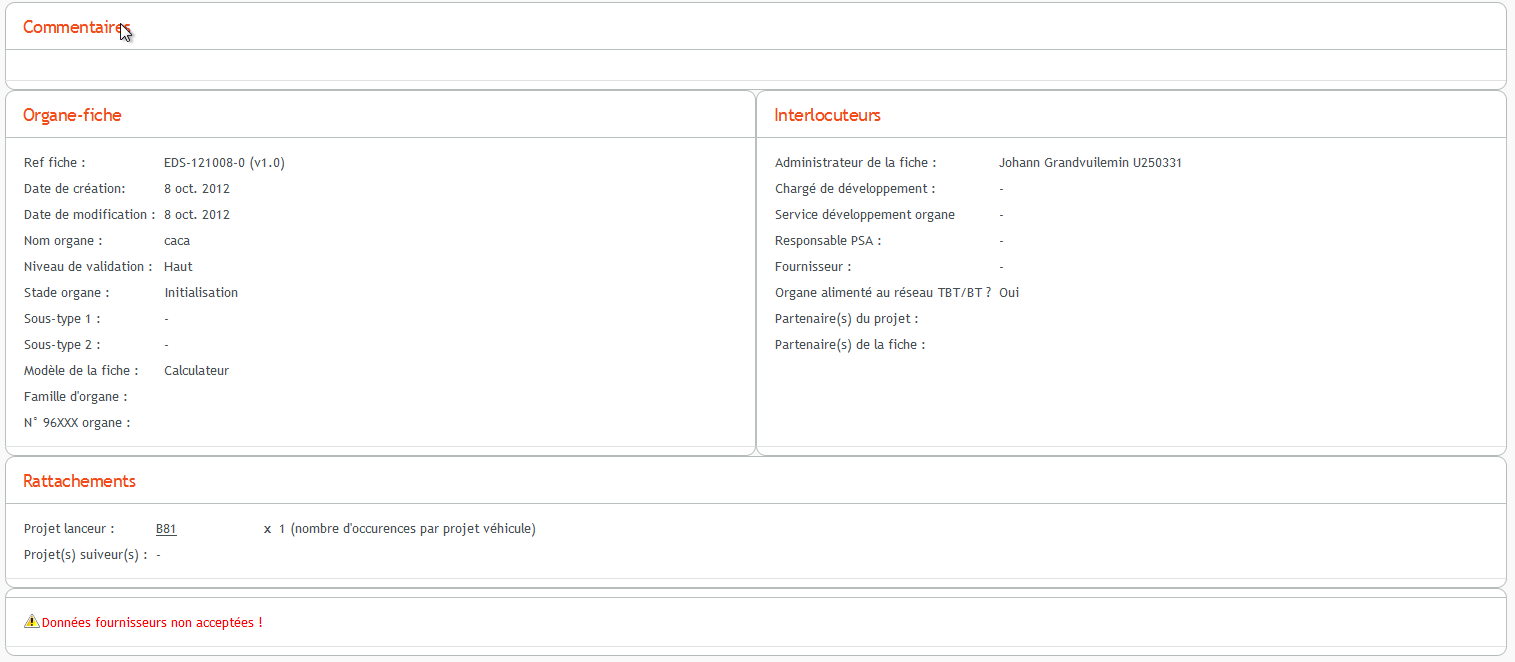


#### Exemple de code source

Voir l’implémentation et l’utilisation de EdsApplicationController.sendNotificationMail()

# Définir le contenu des formulaires

## Gérer les données génériques



Le formulaire de données génériques a la particularité de ne pas posséder de modèle de données distinct de la fiche EDS. Il est composé de nombreux champs faisant référence à d’autres entités, comme les projets, ou les utilisateurs. Toutes les informations de ce formulaire sont directement stockées dans la fiche (classe EdsEds).

Ce choix s’explique car de nombreuses fonctionnalités de l’application, comme la reconduction, le modèle de la fiche ou encore le filtrage des EDS, dépendent de ces données. Il s’agit en quelque sorte du noyau d’une fiche EDS.

## Gérer les consommations de courant

### Gérer les consommations de courant préliminaire

(spécifique au code)

### Gérer les consommations de courant robuste

(spécifique au code)

### Gérer les consommations de courant consolidé

### Gérer les mesures PSA

(spécifique au code)

### Gérer le pilote des dérives (calculé à partir d’autres)

(spécifique au code)

## Gérer les tensions d’alimentations

### Gérer les tensions d’alimentations préliminaires

(spécifique au code)

### Gérer les tensions d’alimentations consolidées

(spécifique au code)

## Gérer les défaillances veille réveil

(spécifique au code)

## Gérer le formulaire CSE

(spécifique au code)

## Gérer le comportement

### Gérer le comportement robuste

(spécifique au code)

### Gérer le comportement consolidé

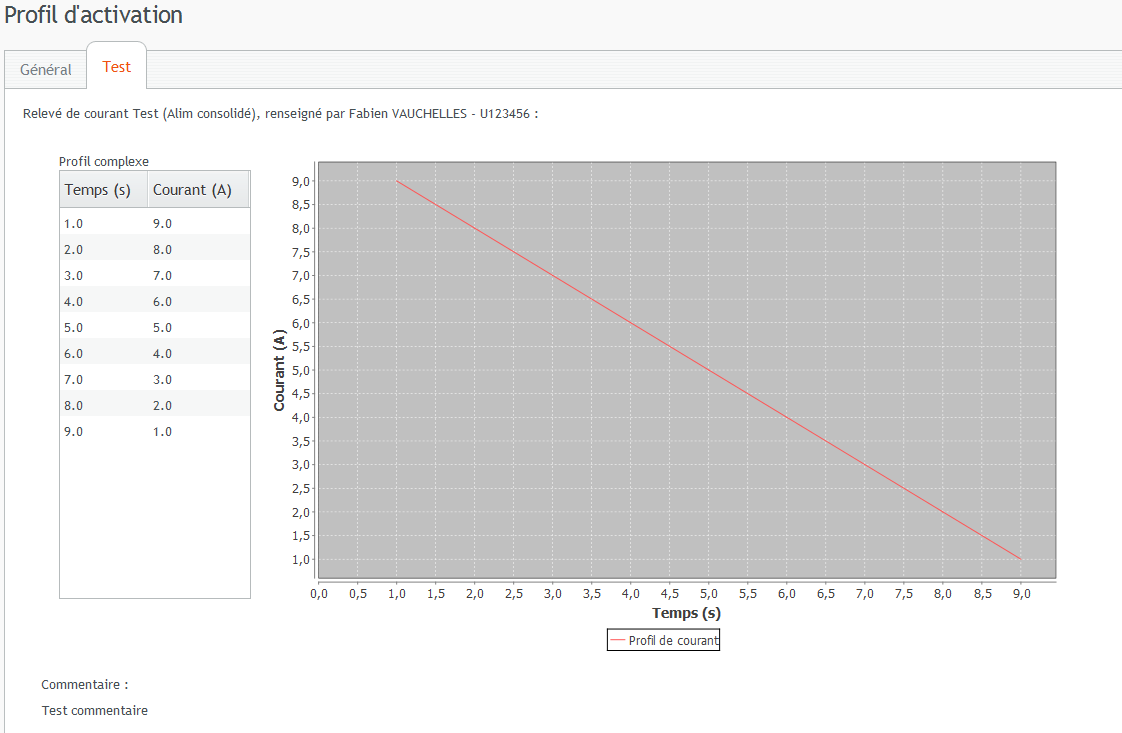
(spécifique au code)

## Gérer les profils de missions

(spécifique au code)

## Gérer les profils d’activation

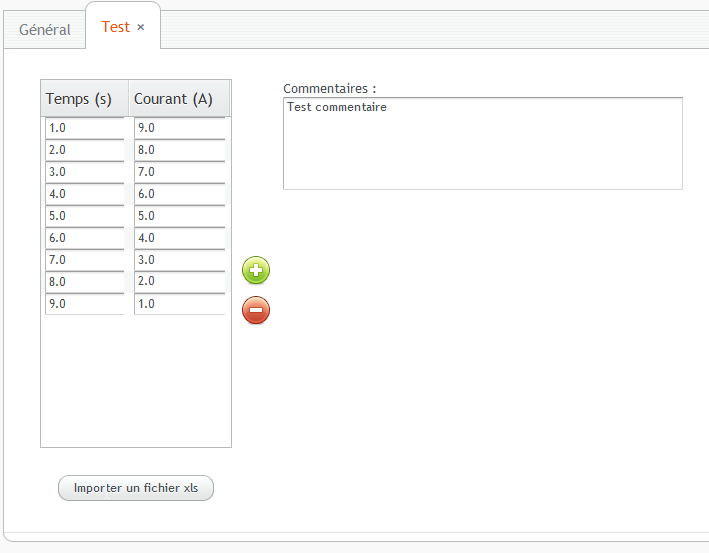
### Les relevés de courant



Des relevés de courant peuvent être associés à la classe EdsMissionActivationFormData.

Chaque relevé permet de définir un profil de courant et est associé à une alimentation du stade consolidé. La représentation graphique de ces données utilise la librairie JFreeChart, et un add-on de Vaadin qui permet d’afficher les courbes ainsi créées.

Il est possible d’importer des données à partir d’un fichier Excel.



## Gérer les indicateurs de la fiche

Le formulaire de validation haute permet aux ayants droit de venir valider les différents stades d’évolution d’une fiche EDS dans cet ordre : préliminaire, robuste, consolidé puis clôturé. Il existe 5 niveaux de droit, ceci afin de séparer la validation globale d’une fiche par au minimum 5 personnes (d’où la validation haute). Le tableau suivant donne les validations nécessaires par stade en fonction des niveaux de droit :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Niveaux de droit** | **Stade préliminaire** | **Stade robuste** | **Stade consolidé** | **Stade clôturé** |
| **Niveau 1 (RS)** |  |  |  |  |
| **Niveau 2 (Responsable Énergie)** |  |  |  |  |
| **Niveau 3 (DBEE statique)** |  |  |  |  |
| **Niveau 4 (DBEE dynamique)** |  |  |  |  |
| **Niveau 5 (CADE)** |  |  |  |  |

Chaque ayant droit peut venir modifier le niveau de validation de son ou ses stades parmi trois valeurs : NOK bloquant, NOK bloquant et OK. Le niveau de validation d’un stade est égal au niveau de validation le plus bas parmi tous les niveaux de validation du stade. Lorsque tous les ayants droit d’un stade ont validé à OK, le stade entier est validé. Entre autre, passer la validation d’un stade à OK entraîne un versionnage mineur de la fiche.

## Gérer les fichiers joints

(spécifique au code)

# Accéder aux données à travers un Webservices

## Caractéristiques

### Description rapide

L’export XML des informations d’une fiche ou d’un projet peut se faire via un web service. L’API Jersey est utilisé pour cette fonctionnalité. Cela permet de récupérer des données à partir d’outils externes à EDS Tools.

### IHM

* Rien

### Packages concernés

* com.inetpsa.eds.ws

## Implémentation

### Modèle de données

* Ressource de connexion : /resources/authenticate/login
* Ressource de déconnexion : /resources/authenticate/logout
* Ressource d’EDS : /resources/eds
* Ressource de projets : /resources/project

### Diagramme de classes

Design pattern utilisé :

* Rien

### Diagramme de séquence

### Exemple de code source

Des exemples en Java sont disponibles dans le package com.inetpsa.eds.ws.sample.

# Accéder une page suivant une URL fixe

## Caractéristiques

### Description rapide

La gestion des URLs est effectuée par un composant VAADIN appelé URIFragmentUtility. Ce composant est ajouté au contrôleur de l’application (EdsApplicationController) via un FragmentChangeListener et intercepte tous les changements de fragment dans la fenêtre. Le fragment est la partie de l’URL se situant après le caractère dièse (#). Il s’agit en HTML d’une ancre.

### IHM

(Non rempli)

### Packages concernés

com.inetpsa.eds.tools.uri

## Implémentation

### Modèle de données

Les fragments acceptés par l’application se composent en premier lieu d’un identifiant puis d’une suite de couples clé/valeur. Ils doivent respecter les spécifications HTML. De plus, ils suivent les mêmes règles que pour la composition des URLs des requêtes de type GET. Ainsi, les fragments se distinguent sous 2 formes :

* #identifiant[&clé[=valeur]]…
* #clé=valeur[&clé[=valeur]]…

, la deuxième forme étant l’équivalent de la première avec un identifiant de valeur null.

Les caractères acceptés pour les clés et identifiants ne sont pas les mêmes que pour les valeurs. L’identifiant et les clés acceptent :

* les caractères alphanumériques (i.e. de ‘a’ à ‘z’, de ‘A’ à ‘Z’ et de ‘0’ à ‘9’),
* les caractères suivant : le tiret (-), le underscore (\_) et le point (.).

Les valeurs acceptent aussi les caractères précédents ainsi que les caractères suivants : l’astérisque (\*), le plus (+) et le pourcentage (%).

Les valeurs doivent être au format UTF-8 et encodées pour être acceptées en tant qu’URL (on utilise les classes URLEncoder et URLDecoder). Ce choix a été fait pour permettre d’effectuer des recherches rapides incluant des caractères spéciaux tels que les caractères accentués de la langue française.

Une clé n’est pas nécessairement accompagnées d’une valeur (i.e. la partie =valeur est omise), auquel cas la valeur vaut null.

Chaque fragment handler possède un identifiant et gère les fragments possédant ce même identifiant.

Il existe 5 fragments différents :

* FH\_Home : répond à l’identifiant ‘home’ et gère l’affichage de la page d’accueil,
* FH\_EdsProject : répond à l’identifiant ‘project’, nécessite la clé ‘id’ avec comme valeur l’identifiant du projet et gère l’affichage des projets,
* FH\_EdsEds : répond à l’identifiant ‘eds’, nécessite la clé ‘ref’ avec comme valeur la référence de la fiche, accepte une clé ‘v’ avec comme valeur la version de la fiche et gère l’ouverture d’une fiche,
* FH\_Admin : répond à l’identifiant ‘admin’ et gère l’accès la partie d’administration de l’application,
* FH\_Null : répond à l’identifiant null et gère les fragments sans identifiant.
* FH\_UserParams : répond à l’identifiant user-params et gère les paramètres utilisateur.

### Diagramme de classes



Design pattern utilisé :

La gestion des URL utilise un pattern commande où chaque A\_FragmentHandler répond à une commande.

### Enchaînement des principales fonctions



### Exemple d’URL acceptées

* <http://localhost:8080/EDSserverWEB#home>
* <http://localhost:8080/EDSserverWEB#eds&ref=EDS-121112-0&v=1.0>

# S’authentifier

## Caractéristiques

### Description rapide

L’authentification permet à un utilisateur de se connecter à l’application. Une fois connecté, il a accès à son contenu en fonction de son rôle.

L’authentification est déléguée au LDAP de PSA.

### IHM

* Rien

### Packages concernés

* com.inetpsa.eds.application.login

## Implémentation

### Modèle de données

* Rien

### Diagramme de classes

Design pattern utilisé :

* Rien

### Diagramme de séquence





### Exemple de code source

Voir LoginWindow.login()

# Traduire en anglais et français

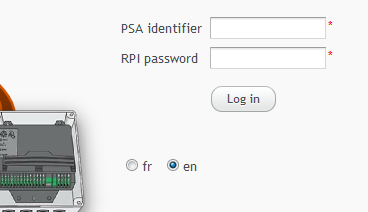
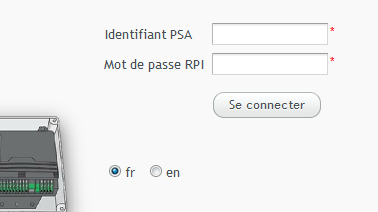
## Caractéristiques

### Description rapide

Le choix de la langue s’effectue sur la page de connexion. La langue sélectionnée est associée au contrôleur de l’application via un « Locale ».

### IHM

Choix de la langue d’affichage sur la page d’accueil.



### Packages concernés

* com.inetpsa.eds.tools.localisation

## Implémentation

### Modèle de données

* Rien

### Diagramme de classes

Rien

### Algorithme

Le « Locale » associé est ensuite utilisé dans les libellés possédant une traduction pour récupérer la valeur correspondante à la langue, mais aussi pour récupérer le bundle qui permettra d’afficher les chaînes de caractères de l’application dans la langue définie. Les valeurs de ces chaînes de caractères sont définies dans des fichiers .properties.

Seul l’affichage de la page d’accueil se met dynamiquement à jour en fonction de la langue sélectionnée.

### Exemple de code source

Récupération de la valeur du libellé en fonction d’un « Locale ».

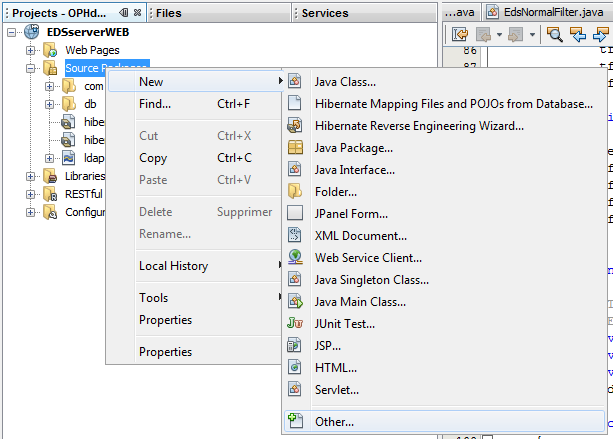


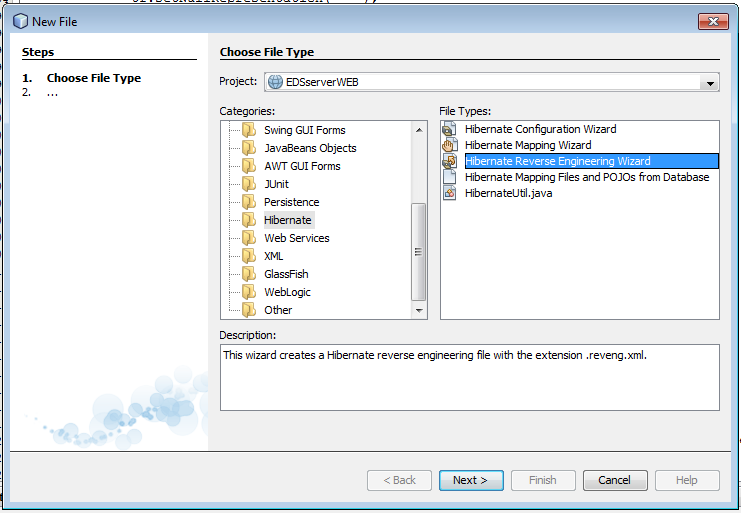
Récupération de la valeur d’un texte dans le bundle en fonction d’un « Locale ».

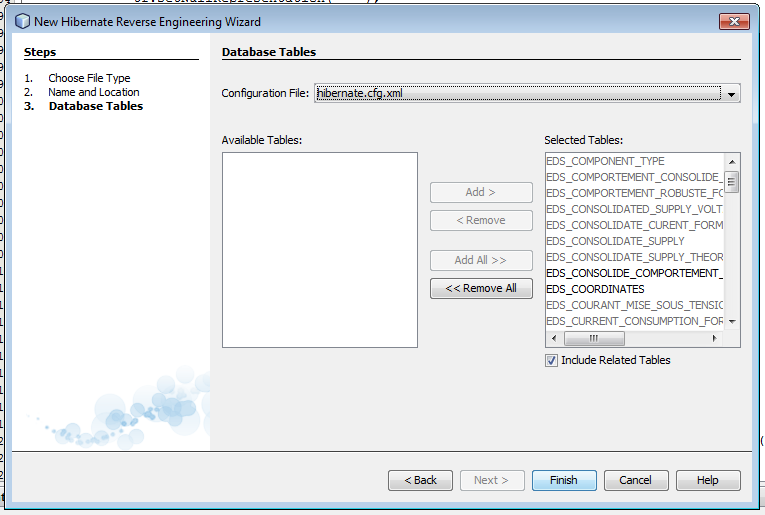


# Annexe : Génération des classes Hibernate du modèle

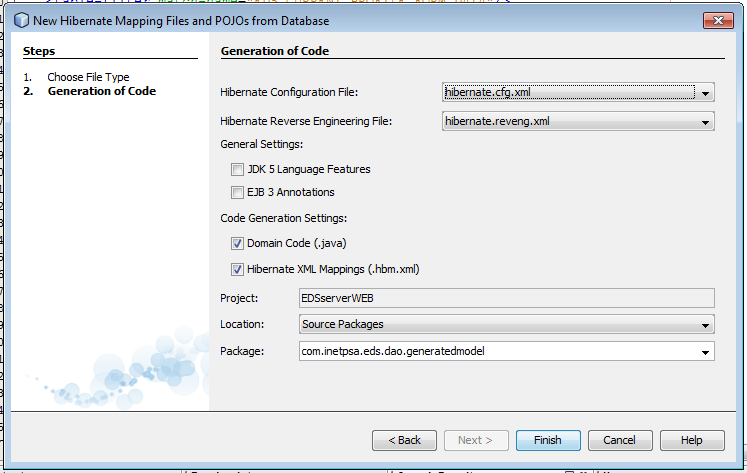
Pour générer les objets à partir de la base de données, il faut commencer par générer un fichier Hibernate de reverse engineering dans la racine du dossier source (src) en faisant un clic-droit puis New > Other > Catégorie : Hibernate > Type de fichier : Hibernate reverse Engineering Wizard. Cliquer sur Next, sélectionner le fichier de configuration hibernate.cfg.xml puis Add All et enfin Finish.







Enfin, il faut générer les classes Java dans un dossier à part (generatedmodel). Pour cela, il faut créer, de la même manière que le fichier précédent dans le package src.com.inetpsa.eds.dao.generatedmodel, un fichier de type Hibernate Mapping Files and POJOs from Database à partir du fichier de configuration hibernate.cfg.xml et du fichier de reverse engineering hibernate.reveng.xml.



Enfin, il faut copier les objets de formulaire qui viennent d’être générés depuis le package generatedmodel dans le package model. Ne pas oublier de changer les chemins de classes (generatedmodel devient model) dans les fichiers de mapping des classes (fichiers \*.hbm.xml) et ajouter les nouveaux mappings dans le fichier hibernate-eds-cfg.xml.