Agence ou Service : NTSC

Projet : ULISSE/SITOOLS2

Guide des Développeurs SITools2 V2

Rédigé par :	Diffusé à :
David ARPIN Jean-Pascal BOIGNARD Bastien FIORITO Mathieu GOND Matthieu MARSEILLE	Jean-Christophe MALAPERT
Approuvé par : Jean-Pascal BOIGNARD	

GROUPE AKKA Technologies – Pôle Informatique & Système
Siège social 3 rue Thierry Le Luron – 92300 LEVALLOIS-PERRET
SITE DE TOULOUSE – 6, rue Roger Camboulives – BP 13633 – 31036 TOULOUSE CEDEX 1 – Tél. 05.34.61.92.92 – Fax : 05.34.61.92.22

LISTE DES MODIFICATIONS DU DOCUMENT

Vers.	Date	Paragraphe	Description de la modification
01.0	12/11/12		Création du document
01.1	20/12/12	6.7	Gestion des utilisateurs via l'API
01.2	13/02/13	Tous 3.3.6 4.4.5 6.8.3	Quelques corrections Resources plugin sur les applications plugins Tutoriel sur la modification de la taille des formulaires dans la vue composite Génération de la documentation JavaScript
01.3	06/06/13	6.9	Comportement de la sauvegarde des formulaires en mode incrusté
01.4	24/04/13	2.4	Installation de SITools2 à partir des sources sous GIT Configuration des services IHM sur un jeu de données
01.5	23/04/13	6.10 2.4.2	Configuration du serveur Jetty Remarques description du projet client-public

Table des matières

1 Introduction

Bienvenue dans le guide des développeurs de SITools2. Ce document est destiné aux développeurs souhaitant ajouter de nouvelles fonctionnalités à SITools2.

Vous découvrirez à travers ce document la mise en place de l'environnement de développement et vous apprendrez comment développer des extensions pour SITools2, les tester et les packager.

Que contient ce document

- Chapitre 1, *Organisation et outils de développement* : dans ce chapitre, vous verrez les règles de codage relatives à SITools2 et la mise en place de l'environnement de développement et de test
- Chapitre 2, *Développement des extensions du serveur* : dans ce chapitre, vous apprendrez comment développer de nouvelles fonctionnalités à partir des différents points d'extension disponibles dans l'API de SITools2
- Chapitre 3, *Développement des extensions de l'IHM* : dans ce chapitre, vous apprendrez à développer de nouveaux composants graphiques et de nouveaux modules
- Chapitre 4, *Packager un développement* : dans ce chapitre vous apprendrez à packager SITools2 afin d'intégrer par exemple vos extensions avec le CORE de SITools2
- Chapitre 5, *Divers* : dans ce chapitre, vous pourrez trouver diverses informations allant de l'optimisation du JavaScript jusqu'à la mise en place de module HTML5 dans SITools2

Conventions

Dans ce document, vous trouverez plusieurs styles de texte pour différentes informations. Voici quelques exemples de ces styles, et une explication de leur signification.

Les **nouveaux termes** ou les **mots importants** sont montrés en gras.

Une ligne de commande est écrite comme ceci

./sitools.sh start

Les mots désignant du code (objet, méthode, …) sont montrés comme ceci : « Cette classe doit hériter de la classe *AbstractFilter*. »

Un bloc de code est montré de cette manière :

```
// récupère le jeu de données
DataSet ds = datasetApp.getDataSet();
```

2 Organisation et outils de développement

Ce chapitre regroupe les informations générales qu'il est nécessaire de lire avant de commencer toute action de développement sur SITools2. Il précise notamment :

- L'organisation du projet,
- Les règles de codage,
- Les outils de développement nécessaires,
- La mise en place des moyens de développement du CORE de SITools2,
- La mise en place des moyens de développement des extensions de SITools2,
- Le lancement de SITools2 et les tests unitaires.

2.1 Organisation du projet

Origines et histoire du projet

L'archivage vise à collecter et à conserver l'ensemble des informations permettant de décrire l'information à préserver dans le but de la restituer dans une forme compréhensible pour une communauté donnée. Dans le modèle OAIS (Open Archival Information System), une archive est décomposée en six entités fonctionnelles : l'insertion, le stockage, la gestion des données, l'administration, la planification de la préservation et l'accès. Actuellement, il existe un ensemble d'archives OAIS avec des implémentations diverses et variées. Chaque nouvelle implémentation conduit alors à redévelopper des services déjà existants dans d'autres archives.

Basé sur l'identification de ces nouveaux besoins et sur le retour d'expérience des utilisateurs, le CNES décide en accord avec ses partenaires d'implémenter une nouvelle génération de l'outil, appelé SITools2. Ce projet a été initié en août 2010. Cette nouvelle version, open source, implémente la couche d'accès aux données ainsi que sa fonction d'administration. Elle permet également de s'affranchir des limites techniques rencontrées dans la première génération de l'outil et d'utiliser des standards reconnus pour faciliter l'interopérabilité des jeux de données. De plus, afin de pouvoir s'adapter aux nouvelles exigences rencontrées pendant la phase de réalisation et d'affiner les besoins des archives scientifiques, le développement de SITools2 est conduit depuis sa création par une méthode Agile.

Les choix technologiques

Durant les années 2009 et 2010, deux études de veilles technologiques ont été menées par le CNES : l'une concernant l'architecture REST (Representational State Transfer) et l'autre en rapport aux technologies RIA (Rich Internet Application).

L'objectif de la première étude était d'étudier le remplacement de la technologie SOAP qui souffrait de problèmes de performances liés au transfert des gros volumes de données entre le client et le serveur. Un autre objectif était de choisir la technologie la plus simple à utiliser afin de favoriser le développement de services spécifiques par les laboratoires scientifiques. L'objectif de la seconde étude était d'étudier l'apport des RIA dans nos systèmes d'informations. L'étude s'est focalisée, en particulier, sur la richesse des composants graphiques et la personnalisation de ceux-ci.

Suite à ces études, l'architecture REST et les RIA respectivement implémentée par les librairies Restlet et Ext-JS, ont été retenues pour l'implémentation de SITools2.

Les accès SVN (deprecated)

Le serveur SVN est hébergé chez http://sourceforge.net/projects/sitools2/. Le serveur SVN est organisé en différents répertoires :

- Branches : l'ensemble des extensions non présentes dans le core de SITools2 et les différents développements annexes
- Tags : Les différentes releases de SITools2
- Trunk : La version de SITools2 qui est en cours de développement

Le logiciel étant industrialisé, il n'est pas actuellement envisagé de permettre un accès SVN en écriture pour des causes contractuelles de maintenance.

Attention : Le repository SVN n'est plus mis à jour depuis le passage sur Git

Les accès GITHUB

Le serveur GIT est hébergé chez https://github.com/SITools2/. La page Github contient un ensemble de dépots :

- core-v1 et core-v2 : Le cœur de SITools2 pour les versions 1 et 2.
- Astronomy-Extension-v1 et Astronomy-Extension-v2 : L'extension astronomie pour les versions 1 et 2.
- EarthObservation-v1 et EarthObservation-v2 : L'extension d'observation de la Terre pour les versions 1 et 2.

Chaque dépots est organisé avec plusieurs branches, à savoir :

- master : Contient la dernière version stable de SITools2
- dev : Contient les développements courant, qui peuvent ne pas être très stable.

La passage à Github facilite la collaboration autour du projet, afin de faire participer la communauté de vos propres développements suivez les instructions ci dessous.

Les Mailing-lists et les groupes d'information

Deux Mailing-lists sont disponibles afin de se renseigner et de poser des questions sur SITools2 :

- Une pour les utilisateurs
- Une pour les développeurs

Pour plus d'informations, veuillez-vous consulter la page suivante http://sourceforge.net/mail/?group_id=531341

Plusieurs moyens sont disponibles pour vous tenir informer des évènements majeures :

- Le flux RSS de SITools2 http://sourceforge.net/news/?group_id=531341
- Le compte twitter de SITools2 http://twitter.com/#!/sitools2

Aider le projet

Voici une liste de pistes possibles pour vous permettre d'aider le projet SITools2 :

- Si vous êtes un développeur
  - Faites-vous connaître et créez des extensions pour le projet.
  - Proposez des idées
  - Envoyez-nous des patchs correctifs
- Si vous parlez une langue non anglophone
  - Traduisez SITools2 dans votre langue
  - Soumettez des articles pour promouvoir SITools2 sur les sites d'information

Collaborer avec le projet

- Si vous êtes un développeur

Le principe de collaboration au projet SITools2 se base sur le principe des « pull request » GitHub. C'est-à-dire qu'il faut faire un « fork » du projet initial en se connectant avec son compte sur GitHub, en accédant à la page de SITools2 et en cliquant sur l'icône « fork ».

Une fois cela fait, on dispose, sur son propre compte GitHub, d'une copie du projet SITools2 initial. On peut donc cloner ce projet, réaliser des modifications et les sauvegarder sur GitHub sans polluer les développements de SITools2.

Une fois que l'on est content de ses développements et que l'on veut en faire partager la communauté, il faut faire une « pull request », qui permet de proposer un ensemble de modifications aux administrateurs du dépôt principal. Il est nécessaire de décrire les fonctionnalités apportées lors de la création de la « pull request ». Un échange peut se créer entre vous et l'administrateur pour mieux comprendre les modifications, tout cela se fait via l'interface de GitHub.

La « pull request » peut être acceptée et prise en compte dans le projet principal, supprimée ou repoussée. Une fois qu'elle est prise en compte elle disparait et la fonctionnalité est disponible dans le dépôt initial.

2.2 Règles de codage

Versions

SITools2 doit fonctionner sur :

- Les OS suivants : Windows, Linux, MacOS

- JAVA 1.5 et JAVA 1.6
- Firefox , Safari, Chrome, IE

Normes copyright

Tout fichier doit avoir un entête selon le masque suivant :

```
/*******************************************************************************
 * Copyright YYYY <organisation, nom avec email>
 * Licence Information
*******************************************************************************/
```

Prévention de l'injection SQL pour les filtres de requêtes

Chaque filtre doit se prémunir contre l'injection SQL :

- Si les valeurs attendues sont numériques, il faut « caster » les valeurs attendues en double
- Si les valeurs attendues sont des chaines alphanumériques, utiliser la méthode :

```
String value = SQLUtils.quote(parameters[VALUES])
```

Libération de la connexion du pool de connexion

Il faut éviter de prendre les connexions à la base de données à la création d'une Resource (dans le code d'implémentation de la ressource, ou le *doInit*) et il faut les relâcher lors de la récupération du résultat (dans le code de la *Representation* associée).

Normes pour les dates

Le format utilisé dans tous les échanges client-serveur pour gérer les dates est le suivant : "yyyy-MM-dd'T'HH:mm:ss.SSS".

En java, la classe DateUtils du package fr.cnes.sitools.util fournit un ensemble de méthode utiles pour les dates.

En Javascript, c'est la classe sitools.common.utils.Date, décrite dans le fichier client-public/js/utils/Date.js.

L'encodage UTF8/ISO

Tout fichier source et de configuration doit respecter l'encodage UTF8 sans BOM ceci afin de garantir la portabilité de l'application sur les différents OS.

Exceptions

Les ressources doivent émettre des ResourceException en respectant les codes de status d'erreur http définis dans la classe Restlet Status.

Cf. http://www.restlet.org/documentation/2.0/jse/api/org/restlet/data/Status.html

Une SitoolsException existe mais uniquement pour les traitements internes.

Version des extensions

Les différentes extensions doivent être versionnées en utilisation setClassVersion. Il est important que si une modification est effectuée dans une extension que le numéro de version de celle-ci soit changé. Sans cette règle, il sera impossible aux instances SITools2 de savoir que la configuration d'une extension a changé.

2.3 Environnement et outils de développement

Installation du PROJET Eclipse

Les différents chemins sont donnés à titre indicatif sur ce qui est fait chez AKKA et doivent être adaptés au contexte du développeur.

Installation d'Eclipse

La dernière version d'Eclipse se trouve à l'adresse suivante : http://www.eclipse.org/downloads/. La version d'Eclipse nécessaire pour une utilisation optimale du projet SITools2 et la version Eclipse **IDE JAVA EE Developer**. Le dossier archivé, une fois décompressé, a pour nom « eclipse ».

Eclipse est installé dans :

- D:\eclipse (Windows)
- <HOME>/eclipse (Linux)

Patch pour une installation sous Linux derrière un proxy :

Afin de permettre à Eclipse d'accéder au réseau derrière un proxy, une étape préliminaire peut être nécessaire sous Linux. Il s'agit d'ajouter la ligne :

```
-Dorg.eclipse.ecf.provider.filetransfer.excludeContributors=org.eclipse.ecf.provider.filetransfer.httpclient
```

au fichier eclipse.ini (dossier racine de l'installation d'Eclipse). Cet ajout nécessite un redémarrage d'Eclipse.

Installation des outils GIT

SITools2 est désormais distribué sur Github, il est donc nécessaire de disposer des outils GIT pour récupérer et manipuler le code source.

Sous Ubuntu, il suffit d'installer le paquet git :

```
sudo apt-get install git
```

Sous Windows, télécharger la distribution de Git pour Windows (http://git-scm.com/download/win) et l'installer. Laisser les options par défaut lors de l'installation.

Une fois l'installation terminée, ouvrir l'outil « Git Bash » qui propose une console de style unix et qui permet d'exécuter toutes les commandes GIT.

2.4 Développement du CORE et des extensions de SITools2

Ce chapitre décrit les étapes permettant de récupérer les sources de SITools2 afin de développer le cœur du Framework et ses extensions.

2.4.1 Récupération des sources du projet SITools2

Récupération du core

Ouvrir une console Git Bash et créer un répertoire qui contiendra le projet SITools2.

Se positionner dans ce répertoire et « cloner » le dépôt principal de SITools2

```
git clone https://github.com/SITools2/core-v1.git  .
```

Remarque : Remplacer v1 par v2 pour récupérer la version 2 de SITools2.

Récupération des extensions de SITools2

Les extensions de SITools2 (Observation de la Terre, Astronomie, Métacatalogue…) sont distribuées sur des dépôts différents. Le principe est de créer un dossier nommé « extensions » dans le répertoire du projet SITools2 e d'utiliser la commande « git clone » sur les différentes url de projet.

2.4.2 Configuration de Eclipse

Lancement d'Eclipse

Sélection de l'espace de travail (workspace) :

Au premier lancement d'Eclipse, celui-ci demande la configuration de l'espace de travail. **Configurer cet espace en indiquant le chemin vers le dossier workspace** (cf. **Erreur ! Source du renvoi introuvable.**, choisir le dossier workspace créé à la suite du clone GIT).

Pour que cet espace de travail reste le même au démarrage d'Eclipse, cochez la case correspondante dans la même fenêtre de configuration (cf. **Erreur ! Source du renvoi introuvable.**).

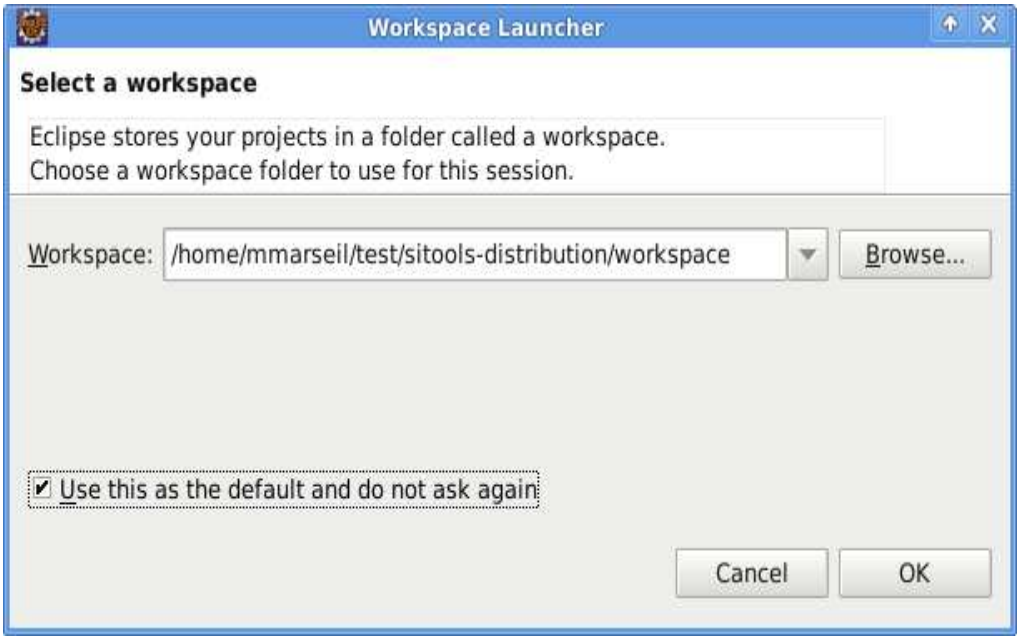


Figure 1 : Sélection de l'espace de travail au premier démarrage

Configuration d'un proxy sous Linux (cf. **Erreur ! Source du renvoi introuvable.** :

L'installation des plugins nécessaire à **l'utilisation de SVN sous Linux nécessite une connexion réseau**. Si un proxy doit être configuré, cela est possible via la fenêtre accessible dans Window > Preference > General > Network connections. Utiliser alors la configuration Active Provider > Manual pour **rajouter vos paramètres de proxy** en éditant les trois protocoles disponibles (HTTP, HTTPS et SOCKS). Appliquez les changements et faites OK.

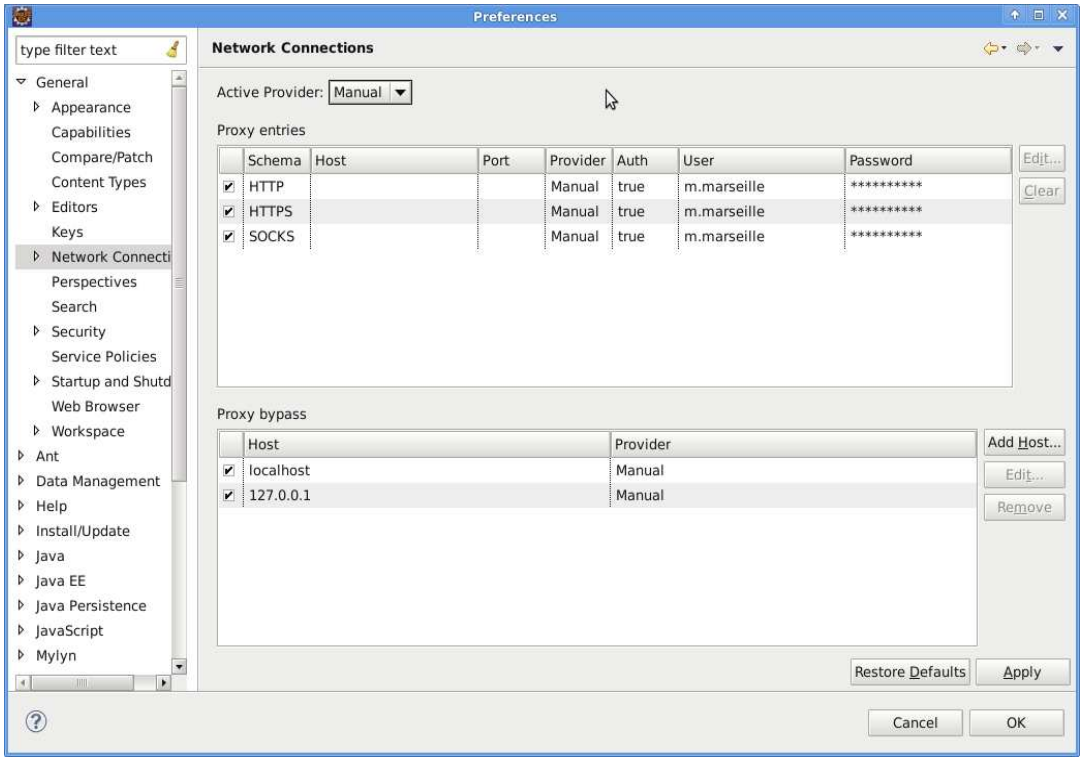


Figure 2 : Configuration d'un proxy (N.B. : les hôtes et ports ne sont pas encore renseignés dans cette capture d'écran)

Installation du plugin EGIT

Le plugin EGIT est indispensable pour recueillir et utiliser les projets de SITools2.

EGIT est déjà présent sur Eclipse Juno 4.2

L'installation de plugins pour Eclipse passe par la fenêtre disponible via Help > Install new Software. Il suffit alors de renseigner l'adresse http://download.eclipse.org/egit/updates dans le champ « Work with » et de taper Entrée pour obtenir la liste des plugins disponibles. On peut également cliquer sur « Add... » et renseigner les champs « Name » et « Location » (ce dernier avec l'URL, cf. **Erreur ! Source du renvoi introuvable.**).

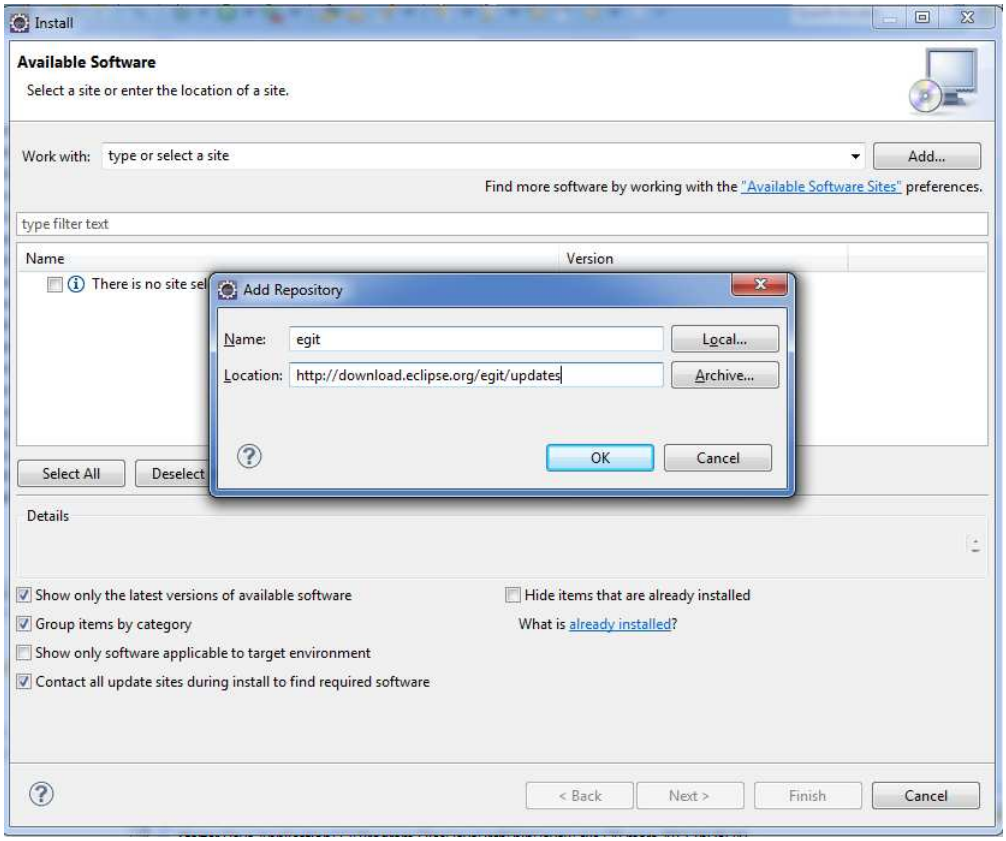


Figure 3 : Ajout d'une URL de téléchargement de plugins pour Eclipse

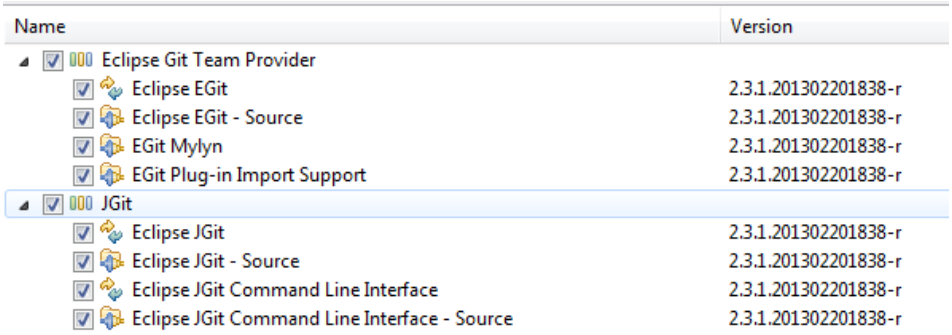


Figure 4 : On sélectionne l'ensemble des plugins proposés et on clique sur « Next ».

Les plugins s'installent et on redémarre Eclipse

Ajout d'un dépôt local GIT

Afin de récupérer les différents projets qui constituent SITools2, il faut d'abord créer un lien vers le dépôt GIT local créé lors du clone initial. Pour cela, il faut tout d'abord ouvrir la perspective GIT Repositories désormais disponible dans Eclipse.

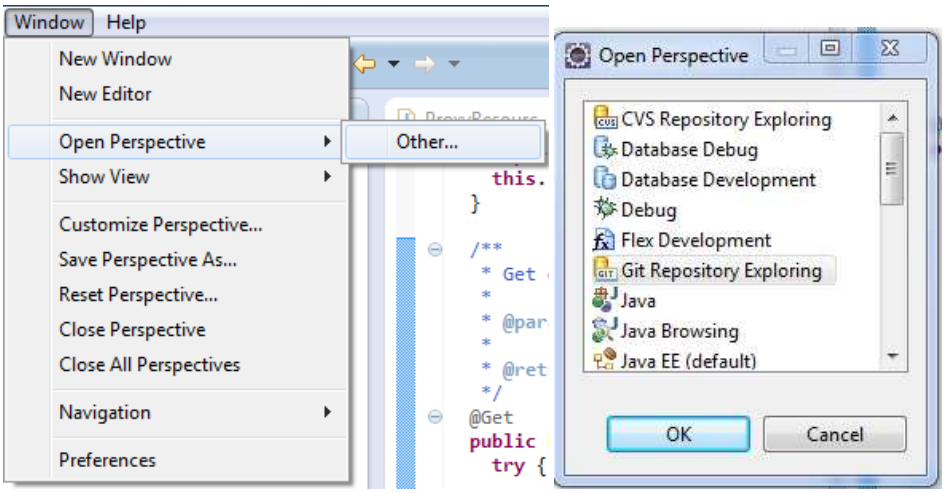


Figure 5 : Ouverture de la perspective GIT repositories

Ensuite, sur le panneau de gauche, rajouter l'URL du dépôt GIT en cliquant sur le lien Add an existing local Git repository ou directement en cliquant sur l'icône ( )

Ensuite choisir le dossier qui contient le dépôt GIT local, c'est-à-dire le dossier dans lequel on a fait le clone intial puis cliquer sur finish pour ajouter le dépôt à eclipse.

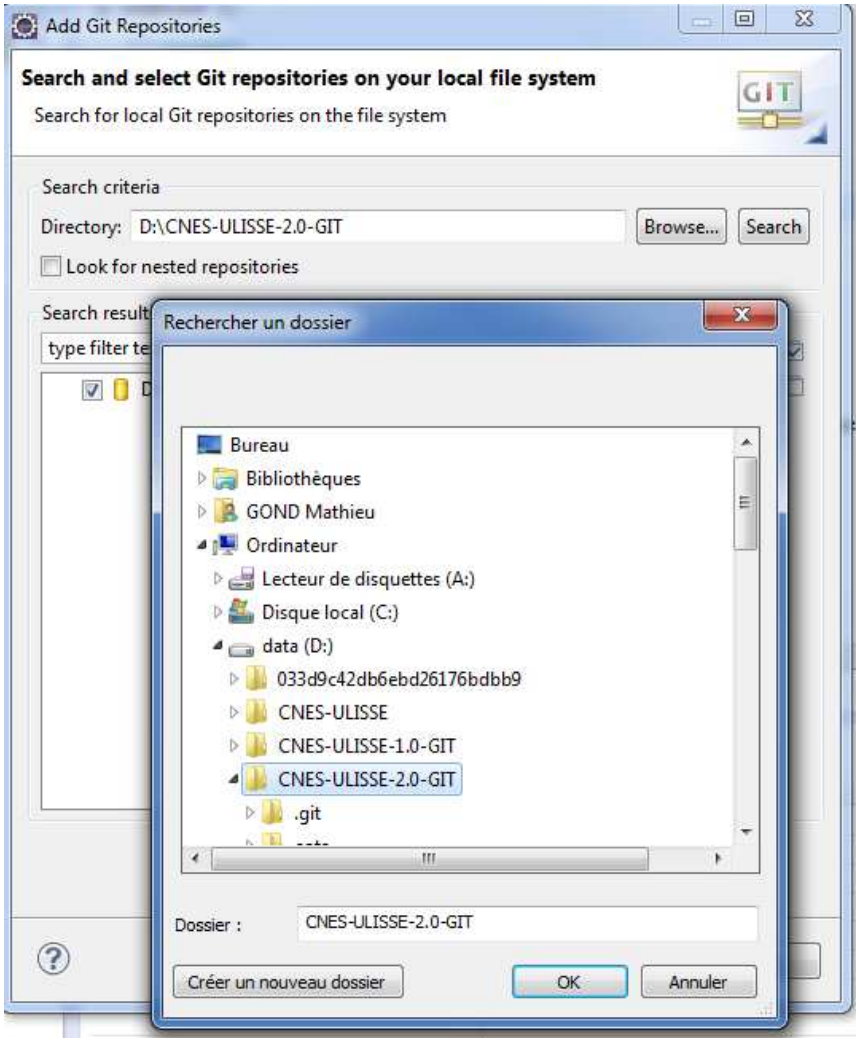


Figure 6 : Création d'un nouveau dépôt GIT local

Création des projets Eclipse

Une fois le dépôt disponible, Eclipse est capable d'y repérer tous les projets présents, afin de les instancier.

Pour cela, toujours dans la perspective Git Repositories, cliquer droit sur le dépôt et choisir « import projects… »

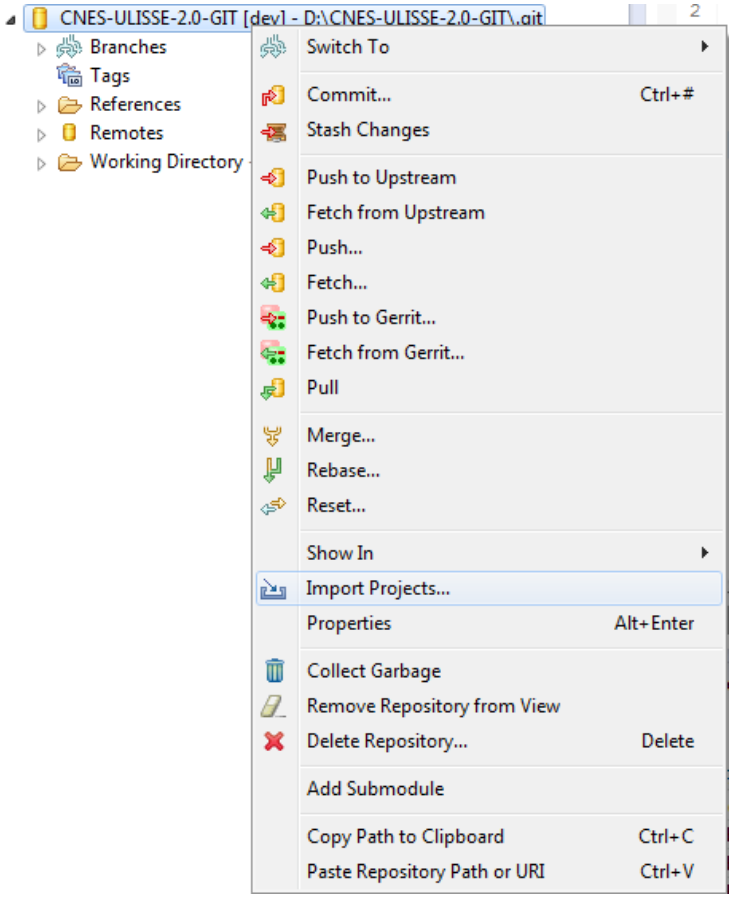


Figure 7: Ajout des projets étape 1

Choisir workspace dans l'arbre puis sélectionner l'ensemble des projets disponibles. Cliquer sur finish pour importer l'ensemble des projets.

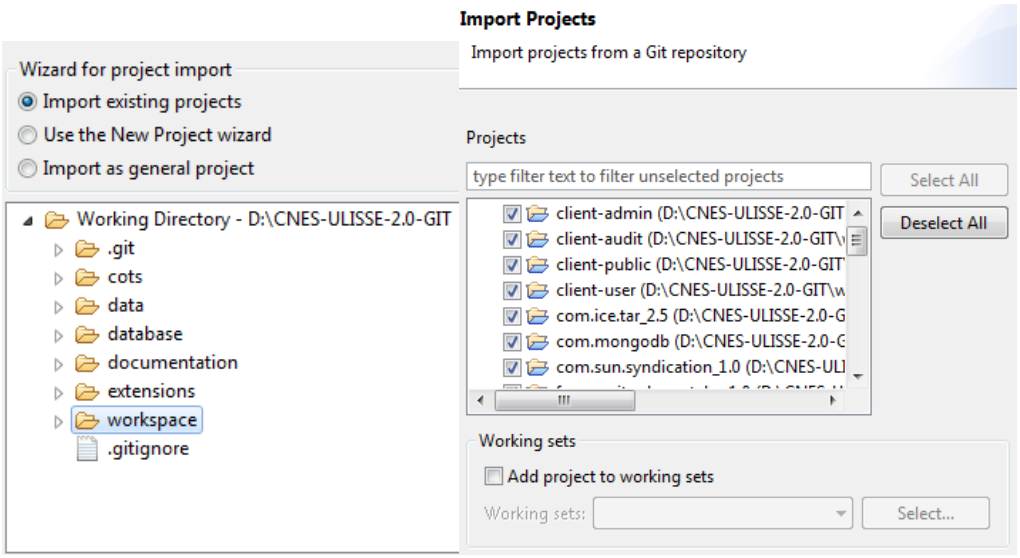


Figure 8 : Ajout des projets étape 2

Organiser ses projets en « working set » (facultatif)

Changer de perspective pour la perspective Java

Choisir l'organisation des packages par working set (flèche vers le bas, en haut à droite du Package Explorer)

Créer des working set y ajouter des projets

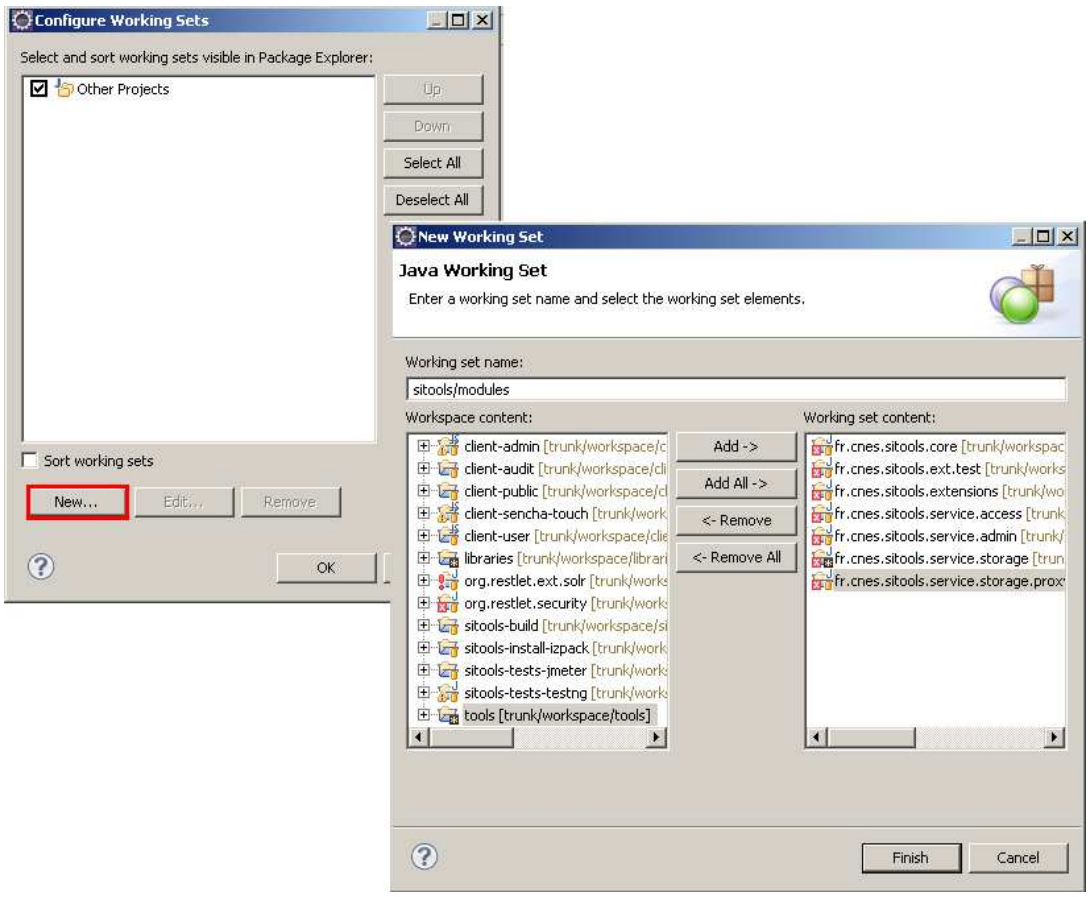
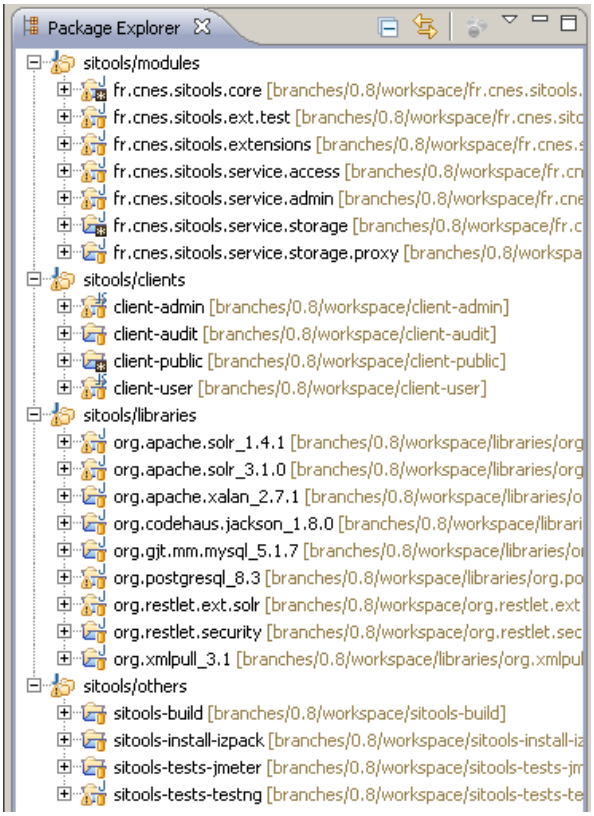


Figure 9 : Création de Working sets

On peut donc obtenir une organisation comme celle-ci :



Configuration de la plateforme de développement de Plugins Eclipse

Cette étape est indispensable pour le bon fonctionnement des projets SITools2 sous Eclipse.

Attention *: SITools n'est pas compatible avec la version 7 de JAVA, il faut veiller à ce que Eclipse soit configuré avec la version 6 de JAVA*

fin de pouvoir lancer Sitols2 dans Eclipse, il est nécessaire de définir une « Target platform ». Celle-ci contient les librairies nécessaires à l'exécution de Sitools2 en mode OSGI et en mode standard.

- Dans Eclipse, ouvrir les préférences
- Choisir « Target platform » dans la partie « Plug-in development »

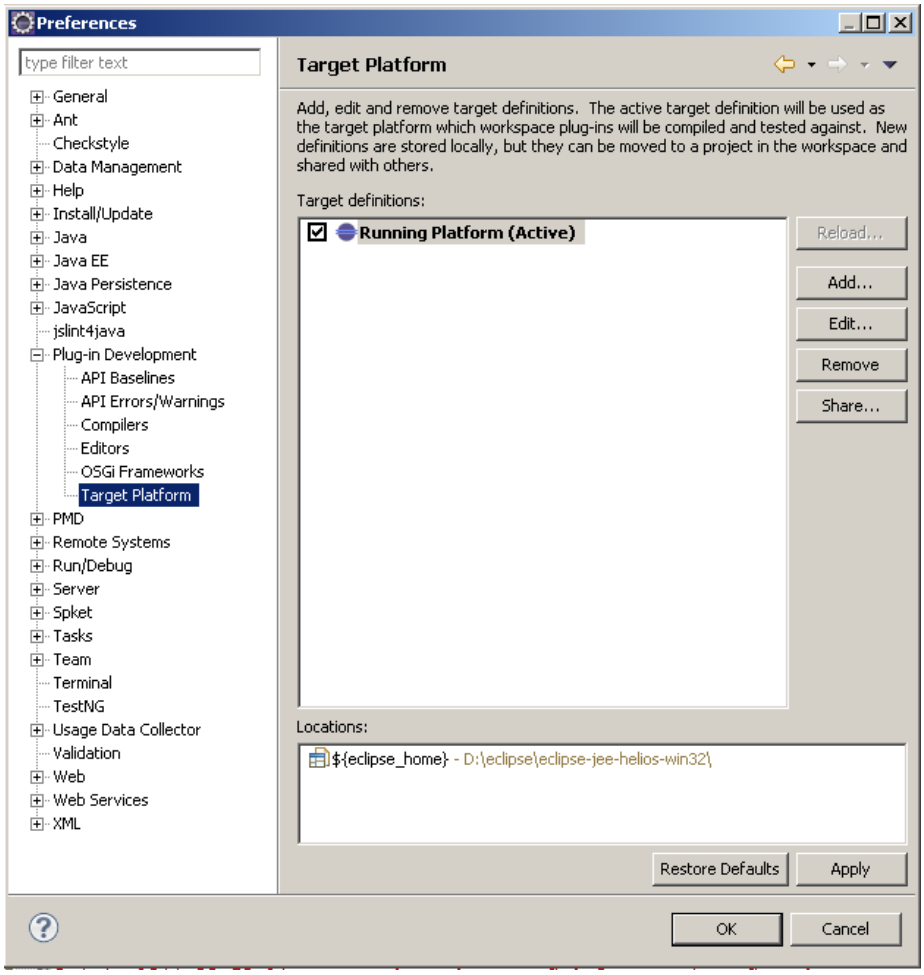


Figure 10 : Edition des préférences de la plateforme de développement de Plugins

- Ajouter une nouvelle plateforme (bouton « Add... »), initialisation de zéro (cf. Figure 10 : Edition des préférences de la plateforme de développement de Plugins)
- Ajouter deux dossiers à la plateforme ; (cf. Figure 11 : Ajout d'une plateforme cible, à partir de zéro)
    - Le dossier Restlet : <HOME>/sitools-distribution/cots/restlet-2.0.5-patched
    - Le dossier où est installé Eclipse : <HOME>/eclipse

Figure 11 : Ajout d'une plateforme cible, à partir de zéro

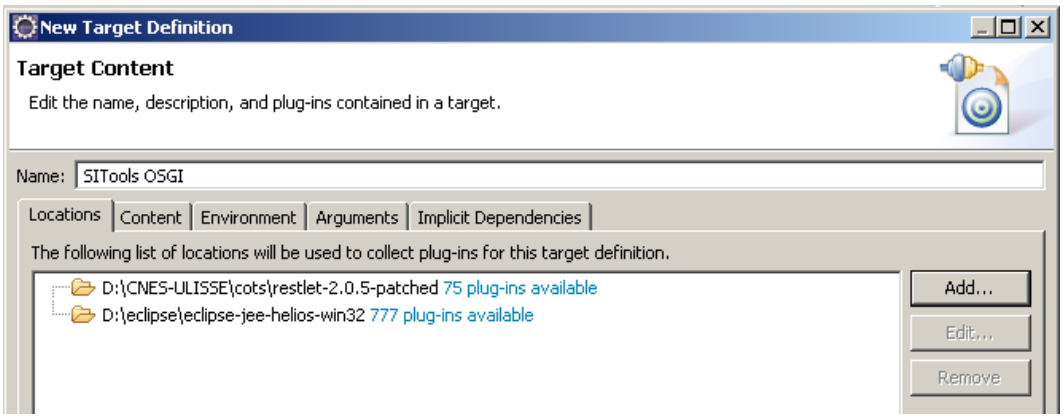


Figure 12 : Ajout des chemins contenant les plugins nécessaires

- Sélectionner :
  - L'ensemble des bibliothèques du dossier Restlet
  - Les trois librairies OSGI du dossier Eclipse (cf. Figure 12 : Ajout des chemins contenant les plugins nécessaires)
    - org.eclipse.osgi.services
    - org.eclipse.osgi.util
    - org.eclipse.osgi

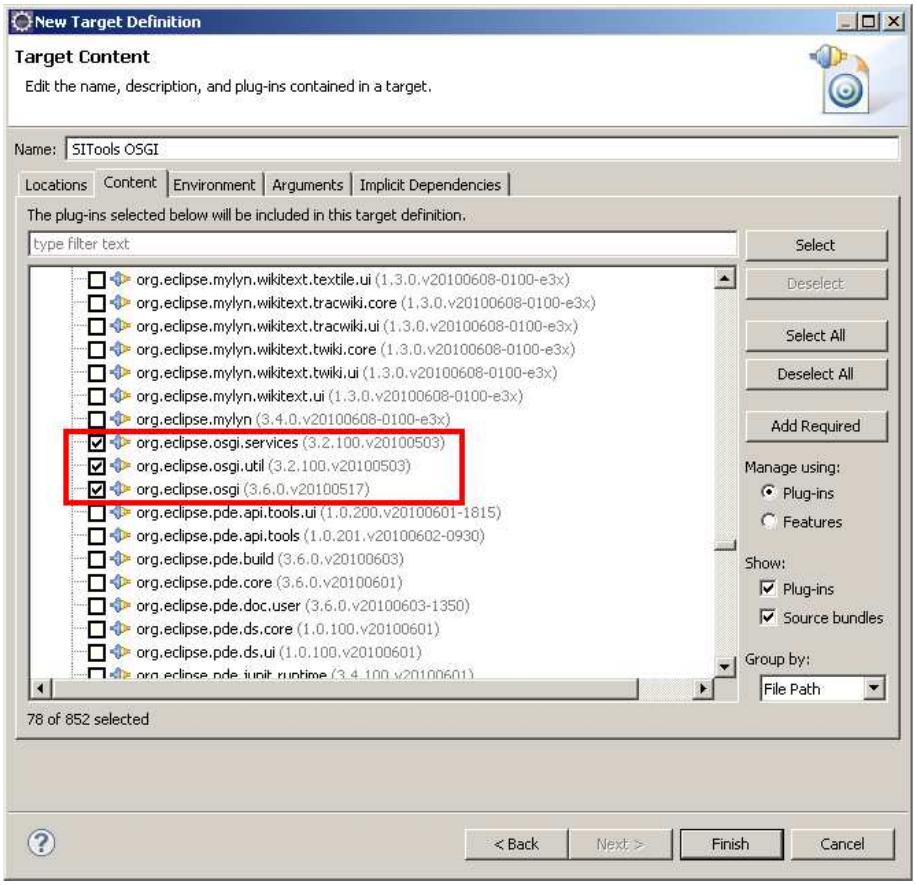


Figure 13 : Ajout des plugins Eclipse pour OSGi

- Cocher la « target platform » ajoutée pour l'activer (cf. Figure 14), en appuyant sur « Apply » : le Workpace va se reconstruire et, au bout de cette étape, tous les projets ne doivent plus présenter d'erreurs de compilation. Pour fermer cette fenêtre, cliquez sur « OK ».

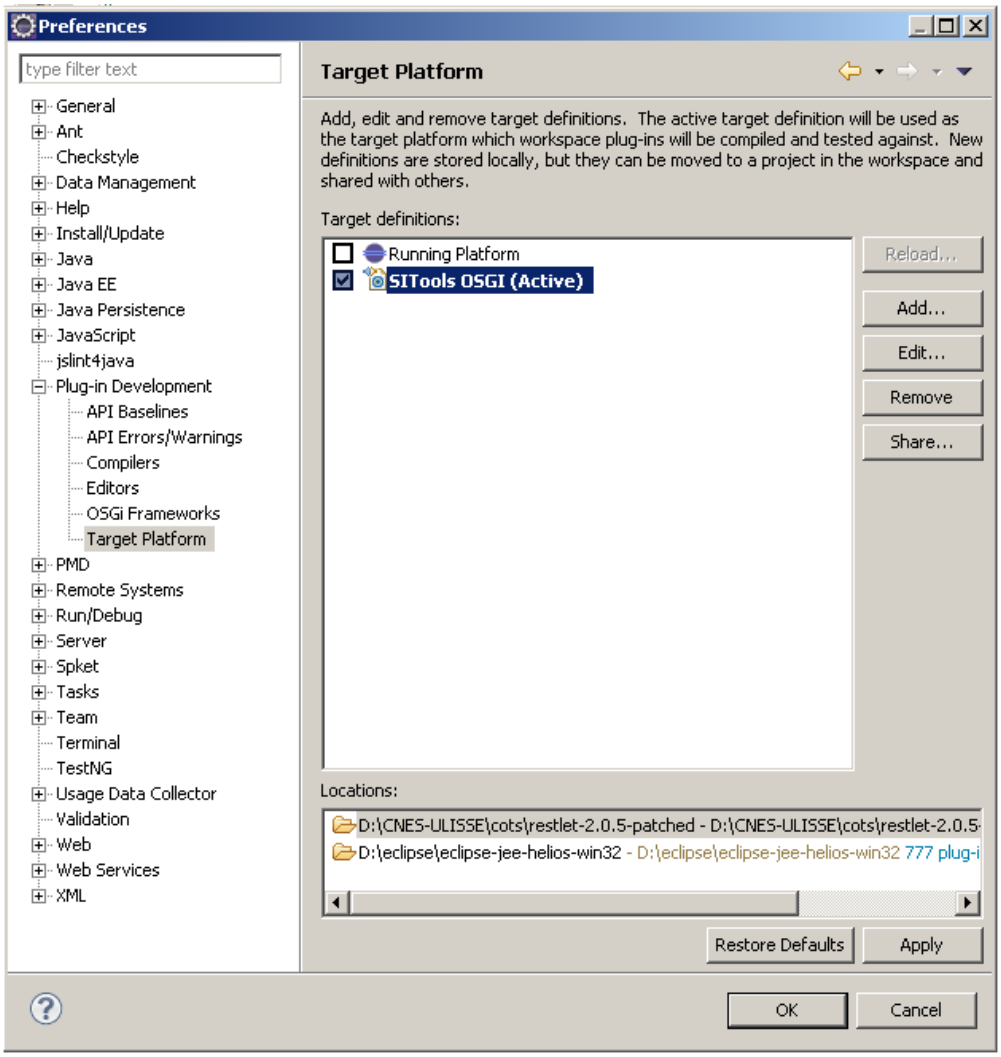


Figure 14 : Sélection de la nouvelle plateforme

Notes additionnelles sur les projets

SITools2 fait appel à plusieurs bibliothèques qui sont listées ci-dessous :

Composant	Bibliothèques	Commentaires
Framework	org.restlet.jar	Version courante : [version=2.0.5]. L'application doit fonctionner en mode standalone avec Jetty. Le mode de fonctionnement J2EE, d'une

		application déployée dans un container de servlets Tomcat ou Jetty n'a pas été maintenu comme exigence au profit de l'étude d'un déploiement dans une plateforme OSGi. Pour répondre notamment à cette dernière contrainte, nous avons été contraints de modifier quelques fichiers de Restlet pour constituer une version « restlet-2.0.5-patched ».
Extension xstream	org.restlet.ext.xstream.jar	
	com.thoughtworks.xstream.jar	L'intérêt de cette extension est de pouvoir réaliser indifféremment le mapping classe Java <> flux xml/json. Mais l'inconvénient est que la définition du schéma xsd doit être réalisée « à la main »
Extension pool de connexions JDBC	org.apache.commons.dbcp.jar	
	org.apache.commons.pool.jar	
Extension fileupload	org.restlet.ext.fileupload.jar	
	org.apache.commons.fileupload.jar	
	org.apache.commons.httpclient.jar	
	org.apache.commons.io.jar	
	org.apache.commons.logging.jar	
Extension freemarker	org.restlet.ext.freemarker.jar	
	org.freemarker.jar	
Extension JSON	org.restlet.ext.json.jar	
	org.json.jar	
Extension RSS et syndication	org.restlet.ext.atom	
	org.restlet.ext.rome	
	com.sun.syndication	
	org.jdom	
Extension email	org.restlet.ext.javamail	
	javax.mail	
Extension Solr	org.apache.solr_3.1.0	
	org.restlet.ext.solr (dossier librairies)	
Extension HTTP	org.restlet.ext.httpclient.jar	
Extension XML	org.restlet.ext.xml.jar	
Extension JDBC	org.restlet.ext.jdbc.jar	Cette extension permet de produire des représentations de contenus de base de données. Notre besoin :

		• métadonnées : représentation dans un format standard de description des tables/colonnes • données : représentation dans un format standard de contenu de base de données, jointures, avec streaming, pagination … Une première exploration de ce composant a été réalisée dans l'étude REST-Restlet.
Extension WADL	org.restlet.ext.wadl.jar	Le besoin est de produire la documentation de l'API d'un serveur Restlet, et de chacune de ses applications au format standard WADL et en HTML (feuille de style Yahoo « WADL2HTML.xslt »). Nous n'utiliserons pas le fichier WADL pour instancier les serveurs / applications car les possibilités semblent insuffisantes à ce jour.
Extension personnalisée	Solr – Lucene	Restlet dispose d'une extension Lucene. Pour les besoins de Sitools2, nous avons créé une librairie (librairies/org.apache.solr_3.1.0) dans le dossier librairies contenant l'ensemble des jar de Solr. Nous avons également mis à jour l'extensions Lucene (org.restlet.ext.solr) de Restlet afin de la rendre compatible avec la version 3.1 de Solr.

***IMPORTANT** : Cette liste n'est pas exhaustive compte tenu des modifications assez fréquentes sur les dépendances du projet. La liste exacte des dépendances est disponible dans le fichier MANIFEST.MF du projet fr.cnes.sitools.core (fr.cnes.sitools.core/META-INF/MANIFEST.MF)*

Bases de données référentes

Base de données utilisateurs

L'enregistrement et la gestion d'utilisateurs de Sitools2 nécessite une base de données de type PostgreSQL ou MySQL, incluse, mais qui doit être correctement configurée.

Pour PostgreSQL, il faut créer une base de données sur le serveur et ajouter un utilisateur « sitools ». Il faut ensuite exécuter le script « pgsql_sitools.sql » (présent dans le dossier « database/PGSQL » à la racine du projet).

Pour MySQL il faut faire de même en exécutant l'ensemble des scripts présent dans le dossier « database/ MYSQL_CNES » sur une base de données créée précédemment.

Une fois la base créée et remplie, il faut configurer Sitools2 pour qu'il pointe vers cette base (propriété « Starter.DATABASE_URL » du fichier sitools.properties ou directement dans l'installateur izPack lors de l'installation).

Base de données FUSE du CNES (tests unitaires, optionnelle)

Les tests unitaires du projet s'appuient sur une table de base de donnée fournie et doit être correctement configurée, notamment en terme de droits d'accès.

Pour configurer la base de données pour les tests, il faut exécuter l'ensemble des scripts présents dans le dossier « database » à la racine du projet sur des bases de données précédemment créées.

Fonctions PostGis dans la base de données PostgreSQL

A noter que pour bénéficier des fonctions PostGis (manipulation d'informations géographiques) quel que soit le schéma utilisé, l'extension PostGis doit être installée dans le schéma « public » de la base de données.

Installation d'autres plugins Eclipse (optionnels)

- Plugin PMD (mesure du code, optionnel) : http://pmd.sf.net/eclipse
- Plugin CHECKSTYLE (vérifications syntaxiques, optionnel) : http://eclipse-cs.sf.net/update/

Les principaux projets de SITools2

SITools2 contient un ensemble de projets qui sont listés ci-dessous :

Projet	Description
client-admin	Projet JavaScript contenant les informations IHM de Sitools2 Admin
client-audit	
client-user	Projet JavaScript contenant les informations IHM de Sitools2 (Portail et Desktop)
client-public	Projet qui contient les sources communes aux 2 applications clientes, utilisateurs et administrateur. /js : Librairies /res : fichiers d'internationalisation, icônes, css, licenses
fr.cnes.sitools.core	Projet Java principal (Serveur Restlet, Classes d'objets, Bibliothèques, procédures Ant, tests unitaires etc.)
fr.cnes.sitools.extensions	Projet lié à fr.cnes.sitools.core, permet de créer des objets Java qui vont être instanciées par le serveur (convertisseurs, filtres)
fr.cnes.sitools.ext.tests	Projet lié aux tests de fr.cnes.sitools.extensions
org.restlet.ext.solr	Projet patchant l'extension Restlet de la librairie SOLR
org.restlet.ext.wadl	Projet patchant l'extension Restlet du WADL
sitools-tests-testnG	Tests d'IHM, automatisés ou non
sitools-tests-jmeter	Tests de performances
sitools-build	Package de distribution propre, minimal et autonome
sitools-install-izpack	Construction de l'installer izpack

sitools-test-izpack	Test de l'installation izpack
sitools-update-classpath	Projet mettant à jour le classpath au redémarrage du serveur, permettant ainsi de prendre en compte de nouvelles extensions à déployer

A ces projets, se rajoutent les projets pour plateforme OSGi suivants :

- **fr.cnes.sitools.service.admin** : service Sitools2 pour l'administration de la plateforme OSGi et des autres services Sitools2.
- **fr.cnes.sitools.service.access** : service d'accès de Sitools2 correspondant notamment à la majorité des applications actuelles exposées par le serveur core.
- **fr.cnes.sitools.service.storage** : service des datastorage correspondant au service d'accès de la fonction ArchivalStorage de l'OAIS

2.4.3 Build du projet

L'utilisation du projet nécessite, au-delà de la compilation des classes Java (automatique sous Eclipse), un ensemble de ressources à répartir selon la configuration de l'utilisateur du projet de développement. Cette fonction est assurée par deux tâches Ant qu'il convient d'exécuter pour pouvoir développer, tester ou utiliser Sitools2. L'organisation en « mode OSGI » des projets demande également de définir une plateforme d'exécution ou « target platform » afin d'utiliser Sitools2 au sein d'Eclipse

Création et configuration d'un build.properties personnalisé

Le projet *fr.cnes.sitools.core* contient une série de fichiers conf/build/properties/build-*.properties, chacun d'eux étant associé à la configuration de l'environnement du développeur. Celui que la tâche Ant va utiliser est désigné par le fichier maître build.properties.

Un fichier de configuration, placé dans conf/build/properties est nommé build-yourID.properties est de la forme :

```
#-------------------------------------------------------------
#  build.properties
#  dependent platform properties
#                          version 1.0
#-------------------------------------------------------------
ROOT_DIRECTORY=<HOME>/sitools-distribution/
HOST_DOMAIN=<ADRESSE LOCALE>|localhost|127.0.0.1
DATABASE_URL=<URL BASE DE DONNEE UTILISATEURS>
DATABASE_USER=<NOM POUR ACCES DB>
DATABASE_PASSWORD=<PASSW POUR ACCES DB>
PUBLIC_HOST_DOMAIN=http://localhost:8182
# Exemple :
ROOT_DIRECTORY=/home/mmarseil/sitools-distribution/
HOST_DOMAIN=silo68.silogic.fr|localhost|127.0.0.1
DATABASE_URL=jdbc:postgresql://odysseus4.fr.akka.corp:5432/CNES-hudson
DATABASE_USER=sitools
DATABASE_PASSWORD=sitools
PUBLIC_HOST_DOMAIN=http://localhost:8182
```

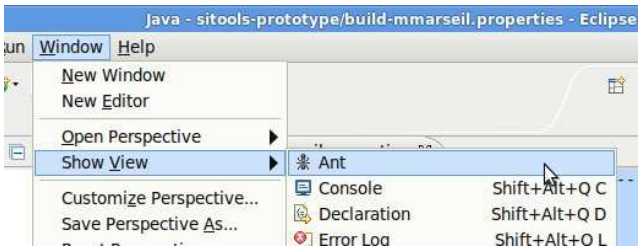
Il suffit alors de modifier le build.properties comme ceci :

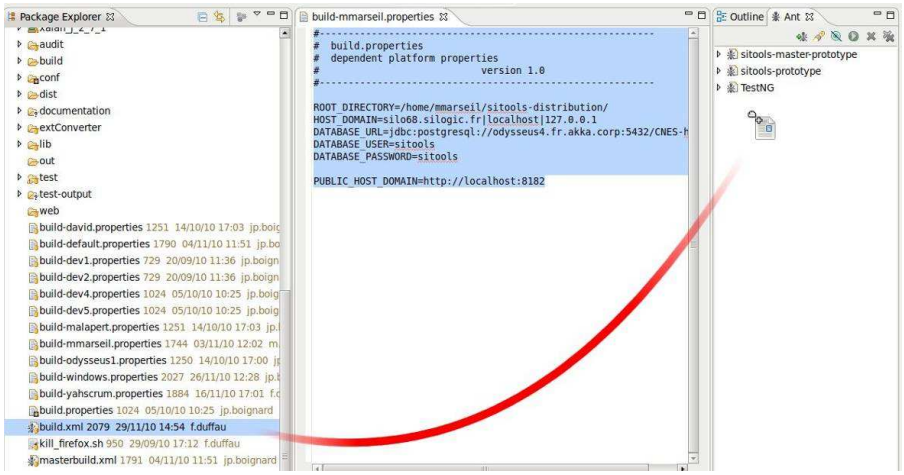
```
#-------------------------------------------------------------
#  build.properties
#  local platform properties
#-------------------------------------------------------------
HOST = yourID
```

Lancement des tâches Ant

Sous Eclipse

Le lancement de tâches Ant est possible sous Eclipse via la vue spéciale disponible :





Un glissé-déposer permet alors de rajouter les fichiers de tâches Ant (build.xml) du projet pour pouvoir ensuite les exécuter :

Dans la vue Ant, dans l'ordre, exécuter :

- refresh-properties
- refresh-userlibraries

Le refresh-properties permet de reconstruire toutes le propriétés du projet en incluant les spécificités indiquées par le développeur dans son build-devname.properties. Il crée un nouveau fichier sitools.properties dans le projet fr.cnes.sitools.core.

Le refresh-userlibrairies recense l'ensemble des librairies du projet sous formes de chemins absolus (qui dépendent donc du build-devname.properties) et les places dans sitools.userlibraries.eclipse.xml. Ces librairies sont indispensables à la bonne compilation du projet doivent donc être, après cette étape, incorporés aux propriétés du projet.

En console

Ant peut également être installé sur tout type de système d'exploitation (Windows, Linux, MacOS X) afin de l'utiliser directement en ligne de commande. Ainsi, pour lancer Ant et reconstruire le projet SITools2, il suffit de se placer dans le répertoire fr.cnes.sitools.core et d'entrer la ligne de commande :

ant « nom_de_la-cible »

où le nom de la cible peut-être vide pour le build par défaut, mais aussi une des cibles définies dans le build.xml.

Fichiers de sortie attendus

En fin de compilation, plusieurs fichiers doivent être présents à la racine du projet :

- *fr.cnes.sitools.core.jar* : JAR exécutable contenant le classpath relatif aux autres projets, dans le fichier MANIFEST.MF. Le classpath contient également les JAR éventuellement déposés dans le dossier ext du projet fr.cnes.sitools.core

- *fr.cnes.sitools.tests.jar* : JAR exécutable permettant de lancer la suite de tests. Attention ! Cette suite de tests nécessite l'initialisation et la configuration de la base de données de tests.

- startSitools.bat : script BATCH de lancement sous Windows. Ces scripts sont générés à partir d'une transformation XSLT effectuées lors du lancement de la cible Ant refresh-userlibraries. Les squelettes de ces scripts (pour Windows et Linux, cf. point suivant) sont donc disponibles dans le fichier conf/build/userlibraries/sitools.userlibraries.xsl.

- startSitools.sh : script BASH de lancement sous Linux, généré par transformation XSLT (cf. point précédent).

- sitools : script de start/stop/monitoring pour Linux. Ce script est une copie directe du fichier conf/bin/sitools

Projet extensions

Afin de développer de nouvelles classes de convertisseurs ou de filtres, les futurs développeurs doivent se placer dans le projet fr.cnes.sitools.extensions. Pour que celui fonctionne correctement sous Eclipse, le projet. Une fois les sources Java créées, la tâche Ant principale du projet (build-extensions) permet de reconstruire le fichier fr.cnes.sitools.extensions.jar. Ce ficher doit se trouver dans le classpath de l'exécution de Sitools2 afin que les classes plugins puissent être découvertes et utilisées.

Projet build-sitools

Le projet build-sitools permet de modifier rapidement le paramètre ROOT_DIRECTORY dans les différents projets du workspace. Ce projet contient un ensemble de fichiers de conf/build/build-*.properties, un pour chaque configuration (comme pour le projet fr.cnes.sitools.core). Il faut donc renseigner le ROOT_DIRECTORY dans le fichier correspondant à votre configuration et renseigner le paramètre HOST dans le fichier build.properties. Il faut ensuite lancer la tache ant par défaut du projet sitools-build. Cette étape doit être effectuée avant toute compilation des projets sitools-install-izpack, sitools-tests-jmeter ou client-audit.

Le projet izPack-Install

Ce projet permet de générer un installeur automatisé Java grâce au projet IzPack. Avant toute chose, il faut mettre à jour le fichier build.properties grâce au projet build-sitools et aux instructions du paragraphe 2.3.6. Ensuite il suffit de lancer la tâche ant par défaut du fichier build-install.xml pour déclencher la création de l'installeur. Il faut également que les sources aient été compilées avant la génération de l'installeur, en effet la tache ant ne compile pas les sources.

2.5 Développement d'un plugin à partir d'une installation IZPack

Il est possible de développer un plugin dans Eclipse à partir d'une installation IzPack. Cela permet de débuger facilement son extension avant de la déployer.

2.5.1 Installation avec IzPack

Sitools2 peut être installé avec un installeur IzPack. Pour ce faire, il suffit de lancer le jar « SITools2-<version>-install.jar » avec la commande « java –jar SITools2-<version>-install.jar » ou en double cliquant dessus. Ensuite il faut suivre les instructions et saisir les informations demandées.

Note : sous windows, il faut choisir un chemin d'installation court et sans espace. (ex : D:/sitools2)

Il faut configurer Sitools2 lors de l'installation avec Izpack. On peut donc choisir les paramètres du virtual host de Sitools2, de base de données, d'email ainsi que du proxy. Les paramètres de proxy sont utiles si le serveur qui héberge Sitools2 à besoin d'accéder à des ressources situés derrière un proxy.

Les paramètres du virtual host permettent de saisir les adresses derrière lesquelles Sitools2 est accessibles via le paramètre « host domain » (liste de nom de domaine ou d'adresse IP séparé par des |).

Dans le cas où l'on configure Sitools2 derrière un proxy, il faut le configurer lors de l'installation. Si un proxy est configuré, tous les appels effectués par le serveur Sitools2 seront envoyés au proxy. Pour éviter que les appels internes ne soient envoyés au proxy, il est nécessaire de configurer le paramètre « non proxy host » (liste de nom de domaine ou d'adresse IP séparé par des |). Ce paramètre doit contenir au moins les adresses définies comme « host domain », ainsi que toutes autres adresses disponibles sans proxy.

Lors de l'installation, bien noter l'url d'installation de Sitools2. Dans notre exemple : D:\CNES_ULISSE\develop-plugins

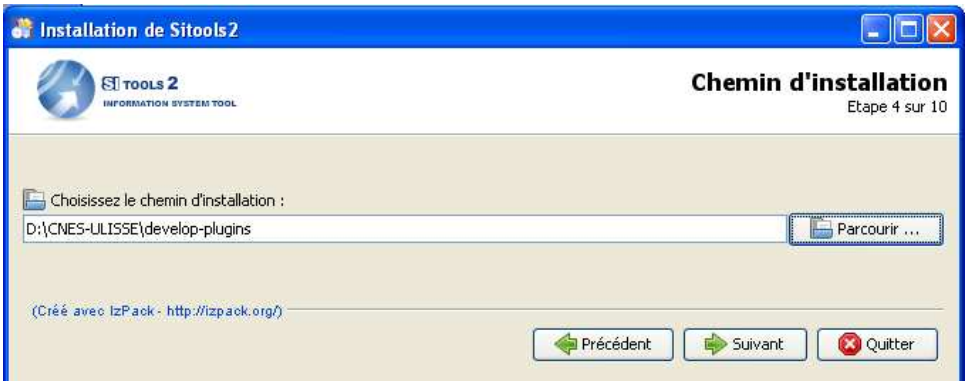


Figure 15 : Choix du chemin d'installation IzPack

Suite à l'installation, la configuration par défaut de la sécurité bloque les appels sur les applications administrateurs et systèmes. Il faut donc modifier le paramétrage de la sécurité dans le fichier ***sitools.properties***

2 solutions sont possibles :

- Configurer les adresses IP autorisés pour les applications d'administrations
    - Configurer les paramètres ***Security.Intranet.net*** et ***Security.Intranet.mask*** avec les sous réseau autorisés à accéder aux applications d'administrations (cf : Sécurisation de l'interface d'administration 3.7)

- Ne pas mettre de filtrage IP pour les applications d'administrations
    - Configurer les paramètres ***Security.Intranet.ADMIN*** et ***Security.Intranet.SYSTEM*** avec la valeur false

2.5.2 Création du plugin

Lancement de Eclipse

Lors du lancement de Eclipse, il faut choisir le dossier « workspace » créé lors de l'installation de Sitools2 comme workspace.

Dans notre exemple : D:\CNES_ULISSE\develop-plugins\workspace

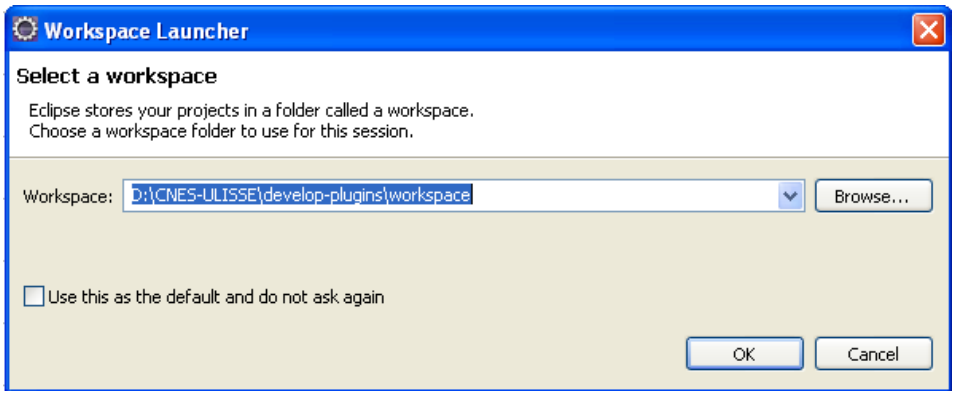


Figure 16 : Choix du workspace

Créer le projet

Une fois que Eclipse est lancé, créer un projet Java. Dans notre exemple on l'appellera « cnes_plugins ».

Il faut ensuite ajouter au classpath du projet le Jar du core de Sitools2, « fr.cnes.sitools.core.jar » présent dans le dossier fr.cnes.sitools.core.

Pour ajouter ce jar :

Click droit sur le projet, « properties », onglet « librairies », « Add External Jars » et choisir « fr.cnes.sitools.core.jar ».

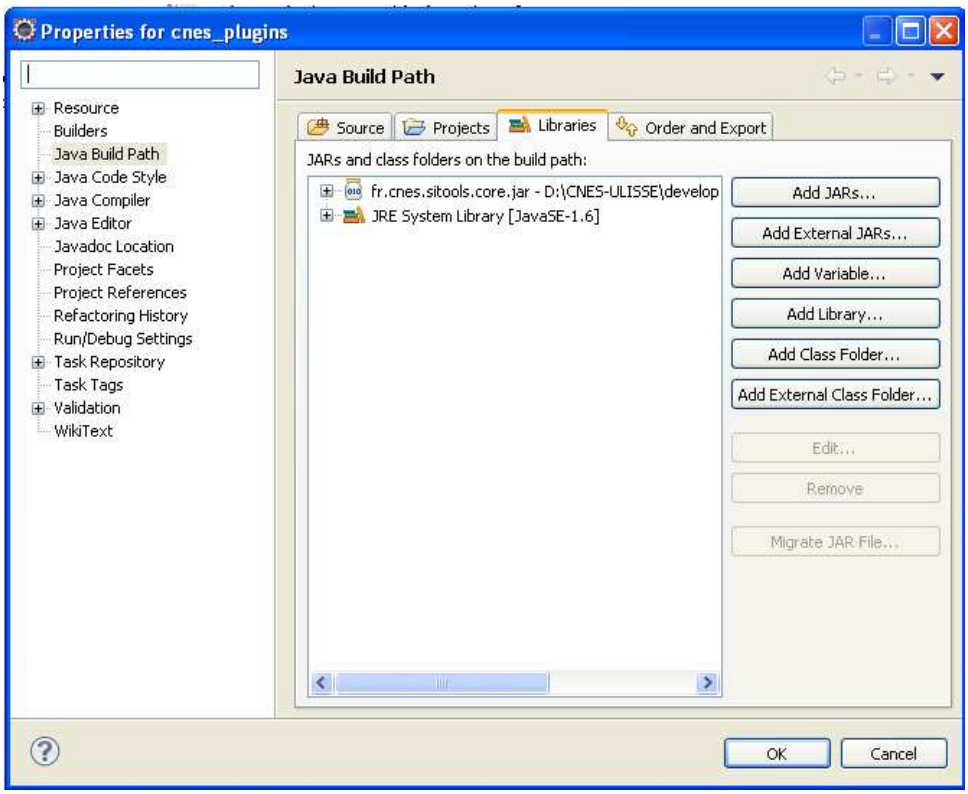


Figure 17 : Ajout d'une dépendance

Normalement toutes les librairies nécessaires seront mises en dépendances. On peut les voir en dépliant le nœud « Reference Librairies » du projet

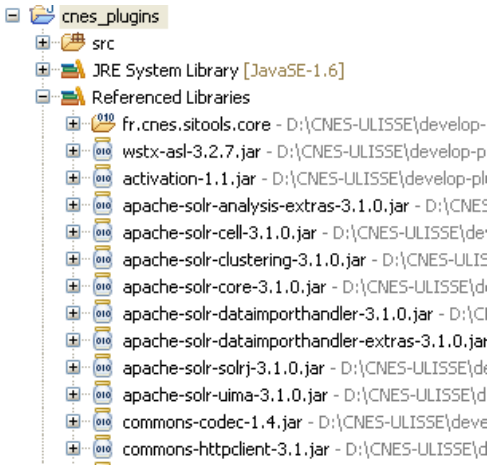


Figure 18 : Liste des dépendances du projet, liste réduite

Développement du plugin

Ensuite il faut ajouter les différentes classes au projet, par exemple si on veut ajouter une ressource et un convertisseur on rajoute 3 classes ainsi qu'un dossier META-INF/services dans src. Ce dossier contient les fichiers Helper pour lister les convertisseurs et les ressources ajouté.

On a donc le projet suivant :

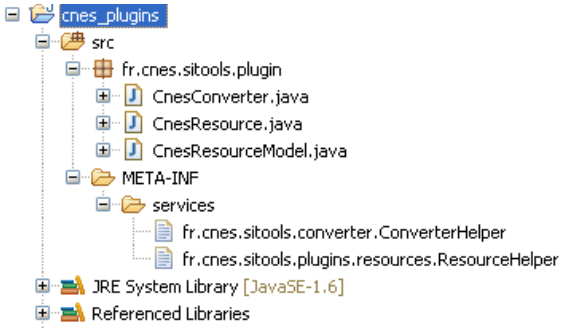


Figure 19 : Hiérarchie Eclipse du projet

Il faut ensuite démarrer le projet, pour ce faire, on click droit sur le projet, puis « Run As » et « Java Application ». Eclipse va rechercher une classe Main à démarrer. Il faut choisir la classe Starter du package « fr.cnes.sitools.server ».

Sitools2 va donc se lancer avec les extensions et les classes que vous venez d'ajouter.

Pour debugger votre extension, il suffit de cliquer sur « Debug As » au lieu de « Run As ».

2.5.3 Export du projet

Pour exporter le projet dans un Jar, il faut cliquer sur le projet, puis « export », choisir « java », « Jar file ».

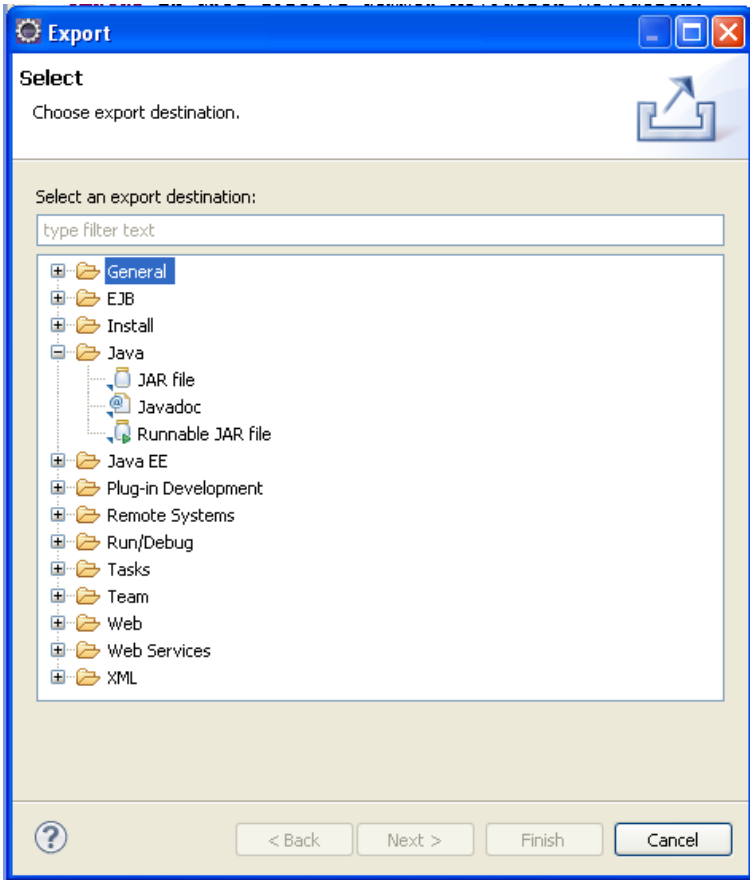


Figure 20 : Choix du type d'export

On laisse les sources à exporter. Bien vérifier que le dossier META-INF est inclus.

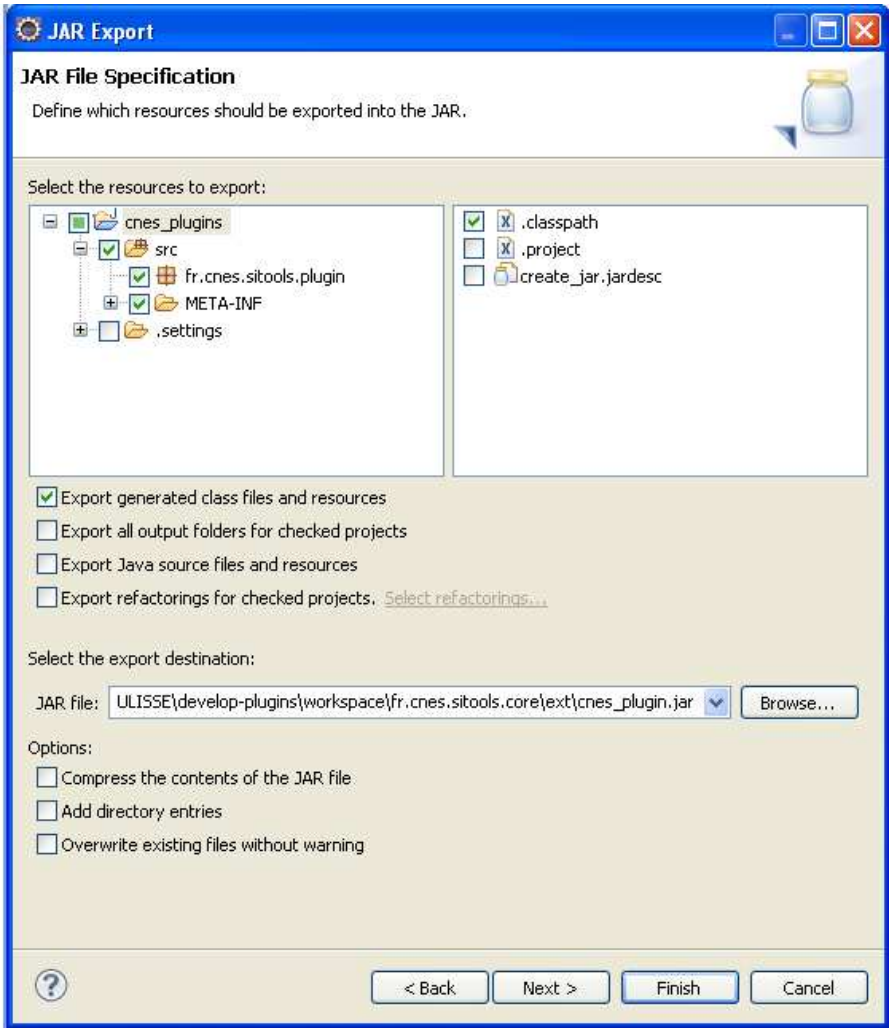


Figure 21 : Paramètres de l'export

On peut également sauvegarder la configuration du Jar dans le projet afin de refaire un Jar sans avoir à refaire les étapes précédentes :

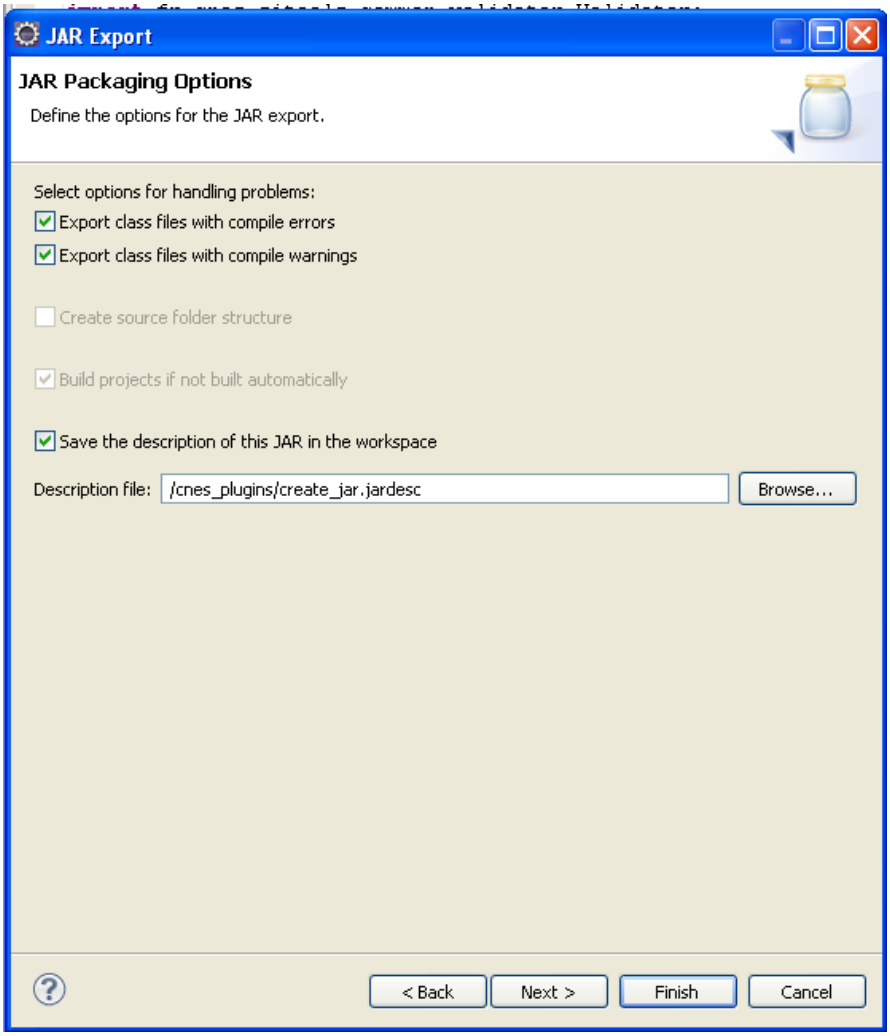


Figure 22 : Sauvegarde des paramètres de l'export

Il suffit ensuite de copier le Jar généré dans le dossier ext du projet fr.cnes.sitools.core pour démarrer Sitools2 avec notre nouveau plugin.

2.6 Développement, lancement et tests

2.6.1 Développement Sitools2

Quelques règles préalables concernant le développement de Sitools2 :

- Le développement dans les projets JavaScript (client-admin et client-user) ne nécessite pas, à priori, d'ajustements du côté d'Eclipse.
- Le développement JAVA du serveur nécessite de relancer le serveur pour prendre effet.
- Un changement d'architecture critique (ajout d'un dossier de source par exemple) nécessite des modifications au niveau des tâches Ant, et donc un re-build global du projet.
- Les classes JAVA pouvant être considérée comme « extérieures » à Sitools2 devraient être introduites via le projet d'extensions fr.cnes.sitools.extensions, ou via un projet équivalent en termes de structure. Pour un nouveau projet, penser à modifier les classpath de compilation et d'exécution (tâche Ant + xsl).

2.6.2 Lancement de Sitools2 sous Eclipse

Afin de lancer Sitools sous Eclipse, il faut créer un « build »

- Aller dans la fenêtre « run configuration »
- Ajouter une nouvelle « Java application »
  - Choisir le projet : fr.cnes.sitools.core
  - Choisir la classe Main : fr.cnes.sitools.server.Starter
- Si l'on veut utiliser les extensions, Ajouter le projet fr.cnes.sitools.extensions au classpath
- Ajouter les jars du serveur Jetty pour pouvoir lancer Sitools2 avec Jetty:
  - Sur l'onglet Classpath, sélectionner « User Entries » dans l'arbre
  - Cliquer sur « Add External JARs »
  - Ajouter les jars suivants :
    - Tous les JARs présents dans cots/restlet-2.0.5-patched/org.eclipse.jetty
    - Le JAR « org.restlet.ext.jetty.jar » présent dans cots/restlet-2.0.5-patched
    - Le JAR « javax.servlet.jar » présent dans cots/ restlet-2.0.5-patched/javax.servlet_2.5

Il est désormais possible de lancer Sitools2 en exécutant le « build » défini ci-dessus.

2.6.3 Lancement de Sitools2 sans Eclipse

Afin de lancer Sitools2 en dehors de l'environnement Eclipse, il faut construire le projet « fr.cnes.sitools.core » comme expliqué dans 2.3. Il faut également faire de même avec le projet « fr.cnes.sitools.extensions ».

Ensuite ; il suffit d'exécuter la commande :

./sitools start

Dans le dossier « fr.cnes.sitools.core ».

NB : au lancement, les JARs présents dans le dossier « ext » seront ajoutés au MANIFEST.MF du JAR « fr.cnes.sitools.core.jar » grâce à l'exécution du jar « sitools-update-classpath.jar ». Cela à pour effet d'ajouter les JARs au classpath et de pouvoir les utiliser.

2.6.4 Tests unitaires

Un ensemble de tests unitaires sont intégrés à la distribution de Sitools2, couvrant l'ensemble des applications de WEB service intégrés et de leurs principales fonctionnalités. L'exécution de ces tests permet:

- de vérifier la non-régression de Sitools2 après le développement d'un nouveau composant,
- de générer la documentation API personnalisée de Sitools2,
- de générer la documentation WADL (format XML et HTML) pour chacune des applications.

Les tests unitaires sont placés dans le répertoire test/src de la distribution.

Pré-requis

Une grande partie des tests unitaires reposent sur la création d'une source de donnée, et de la manipulation de cette source à travers l'API de Sitools2. Les tests unitaires nécessitent donc la présence d'une base de données préexistante et immuable. Il s'agit dans ce cas de la table Fuse de la base de données Sitools2 livrée avec la distribution.

Exécution

Une tâche ant permet l'exécution de l'ensemble des tests unitaires de Sitools2 :

```
# compilation de Sitools2
ant –f build.xml
# execution des tests
ant –f build.xml execute-test
```

Attention *! L'exécution des tests peut prendre plusieurs minutes. Si un seul test doit être joué, il faut construire une nouvelle tâche ant associée.*

Pour plus de facilité, les tests unitaires peuvent évidemment être lancés depuis Eclipse, par un clic droit sur le package de test (répertoire test/src) fr.cnes.sitools -> run As … -> JUnit test. Ils peuvent aussi être lancés individuellement de la même manière. On notera cependant que, dans des cas très rares, des tests unitaires peuvent paraître défectueux lors d'une exécution en groupe sous Eclipse. Dans le doute, un test individuel puis l'exécution de la tâche ant d'exécution de tous les tests permet d'infirmer ou de confirmer le problème.

Rapport d'exécution

Dans le cas de l'utilisation de la tâche ant pour jouer l'ensemble des tests, un rapport final au format HTML est disponible dans le dossier test/reports/index.html.

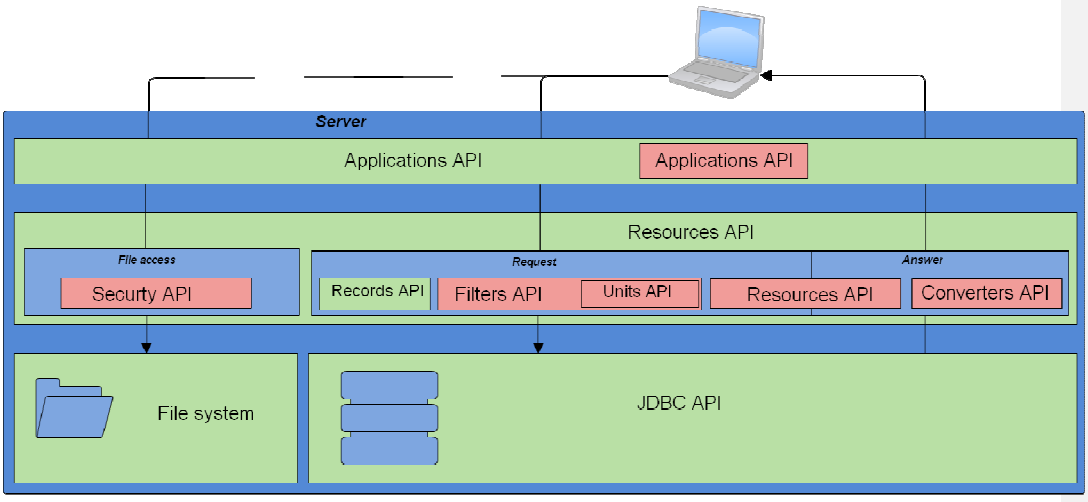
Exécution avec Eclipse

Les tests unitaires peuvent être lancés via, dans « test/src », le paquetage fr.cnes.sitools (clique-droit + Run As … + Junit Test).

3 Développement des extensions du serveur

SITools2 possède un ensemble d'API permettant de construire différents plugins. Ces plugins permettent d'ajouter de nouvelles fonctionnalités ou bien de modifier certains comportements du serveur :

- Le plugin « **Filter** » permet d'ajouter **un filtre de requête** qui va lire une signature dans une URL et la convertir en une partie de la requête SQL interrogeant le jeu de données,
- Le plugin « **Converter** » permet d'appliquer **une fonction de transfert** sur des données requêtées et de fournir une réponse qui sera affichée dans la réponse du serveur,
- Le plugin « **Application** » permet d'ajouter un nouveau comportement au serveur qui peut être indépendant d'un jeu de données ou d'un projet,
- Le plugin « **Resource** » est une ressource synchrone ou asynchrone liée à un jeu de données. Elle permet de récupérer les données requêtées et d'effectuer diverses opérations ainsi que de retourner une réponse qui sera affichée par le serveur,
- Le plugin « **Security Filter** » permet d'ajouter son propre mécanisme d'autorisation pour l'accès à la fonction « stockage » de l'OAIS.
- Le plugin « **Units** » permet d'ajouter ses propres mécanismes de **conversion d'unités**. Ces mécanismes de conversions d'unités peuvent être ensuite utilisés dans les formulaires de requête de SITools2.



3.1 Développement d'un filtre de requête en entrée

Les filtres de requête permettent d'ajouter des contraintes lors d'une requête sur un jeu de données. Chaque filtre de requête peut ajouter des prédicats à la requête SQL qui va ensuite être exécuté par la base de données. Un filtre dispose d'un ensemble de paramètres en entrée, de paramètres internes (constantes) et de paramètres en sortie.

3.1.1 Introduction

La création d'un filtre se résume à l'implémentation d'une classe Java retournant un ensemble de prédicats. Cette classe doit hériter de la classe *AbstractFilter* et implémenter la méthode *createPredicats(Request request, ArrayList<Predicat> predicats)* ainsi que *getValidator()*. Le squelette d'une telle classe est le suivant :

```
public class MyFilter extends AbstractFilter {
public BasicFilter() {
// méta données du filtre
// déclaration des paramètres de configuration
}
@Override
public final ArrayList<Predicat> createPredicats(Request request, ArrayList<Predicat> predicats) throws Exception {
        // récupération des paramètres de requêtes et de configuration
        // création des prédicats
       // retour de la liste des prédicats
}
@Override
public Validator<AbstractFilter> getValidator() {
  return new Validator<AbstractFilter>() {
    @Override
    public Set<ConstraintViolation> validate(AbstractFilter item) {
      // validation des paramètres de configuration
    }
  }
}
}
```

3.1.2 Le constructeur

- **Métadonnées du filtre**

Le Constructeur possède un ensemble de métadonnées relatif au filtre, à savoir :

Métadonnées	Description	Méthodes à employer
L'auteur du filtre	L'auteur du filtre apparaît dans l'interface d'administration	setClassAuthor
Le propriétaire du filtre	Le propriétaire du filtre apparaît dans l'interface d'administration	setClassOwner
Le nom du filtre	Le nom du filtre apparaît dans l'interface d'administration	setName
La description du filtre	La description du filtre apparaît dans l'interface d'administration	setDescription
La version du filtre	La version du filtre apparaît dans l'interface d'administration. Une vérification entre la version du filtre configuré et la version du filtre provenant du JAR est effectuée. Un **warning** est affichée si les versions diffèrent.	setClassVersion

Paramétrage de configuration du filtre

Lors du développement d'un filtre, le développeur peut créer des paramètres de configuration qui seront saisis par l'administrateur du système. Pour cela il suffit d'instancier des objets *FilterParameter(String name, String description, FilterParameterType type)* et de les ajouter au filtre par l'utilisation de la méthode *addParam(final FilterParameter param)*.

Lors de l'instanciation d'un objet *FilterParameter*, les paramètres suivants doivent être spécifiés :

Paramètre	Description
name	Nom du paramètre qui sera affiché dans l'interface d'administration pour la configuration du filtre
description	Description du paramètre qui sera affiché dans l'interface d'administration
filterParameterType	Type du paramètre entrainant un comportement différent dans l'IHM. Le type peut être soit **INTERN** ou **IN**. Un type INTERN est une constante que l'administrateur devra saisir dans l'IHM d'administration. Un type IN est un attribut du jeu de données que l'utilisateur devra choisir parmi la liste exposée dans l'IHM d'administration.

Notion de filtres par défaut

Les filtres sont actuellement le seul moyen d'enrichir l'API de recherche sur un jeu de données, et donc, de générer des prédicats à partir de paramètres d'une requête.

Les composants de formulaires, le filtrage sur les données dans le composant graphique tabulaire (« liveGrid ») utilisent des filtres pour générer des prédicats.

Afin de ne pas alourdir la gestion de l'administrateur, on a ajouté à un filtre la notion « par défaut ». Lors de la création d'un jeu de données, tous les filtres par défaut seront attachés automatiquement au nouveau jeu de données via un mécanisme de Trigger.

Un Filtre peut avoir deux constructeurs. Un par défaut, sans paramètre et obligatoire. Un autre avec un objet de type Context *en paramètre. Dans cet objet* Context*, on peut retrouver le jeu de données en utilisant l'attribut « **DATASET** ». Ce constructeur permet d'avoir une liste de paramètres qui dépend du jeu de données et rend le filtre encore plus souple.*

3.1.3 La méthode createPredicats

Cette méthode est exécutée lors de l'exécution du filtre.

Il faut tout d'abord récupérer l'ensemble des paramètres du filtre. Pour ce faire il faut utiliser les méthodes *getParameterMap()*, *getInternParam(final String name)*.

Une fois les paramètres récupérés, il suffit de créer les prédicats voulus en fonction des attributs du jeu de données paramétrés et des constantes.

Dans cette méthode, on peut également récupérer certaines variables depuis le contexte. Ces variables sont **SitoolsSettings** (l'ensemble des propriétés du fichier sitools.properties) et la **DataSetApplication**.

Pour ce faire, il suffit de les récupérer dans les paramètres du contexte avec les clés : « DataSetApplication» et « ContextAttributes.SETTINGS » :

```
datasetApp = (DataSetApplication) getContext().getAttributes().get("DataSetApplication") ;
sitoolsSettings = (SitoolsSettings) getContext().getAttributes().get(ContextAttributes.SETTINGS ) ;
```

Depuis la DataSetApplication, il est possible d'obtenir une connexion à la base de données :

```
// recuperation complete du dataset
DataSet ds = datasetApp.getDataSet();
// recuperation d'une connexion à la source de données
Connection con = SitoolsDataSourceFactory.getInstance()
.getDataSource(ds.getDatasource().getName()).getConnection();
```

Où datasetApp est la DataSetApplication récupéré depuis le contexte.

▪ Prévention de l'injection SQL

Chaque filtre doit se prémunir contre l'injection SQL. Si les valeurs attendues sont numériques, il faut « caster » les valeurs attendues en double. Si les valeurs attendues sont des chaines alphanumériques, utiliser la méthode suivante :

```
String value = SQLUtils.quote(parameters[VALUES])
```

Chaque développeur peut donc dans l'implémentation de son filtre définir le caractère par défaut de celui ci.

3.1.4 Utilisation de la conversion d'unité

Les classes qui héritent de la classe abstraite AbstractFilter ont à leur disposition la méthode :

```
Double convert(String unitFromName, String unitToName, String valueFrom, String dimension)
```

qui renvoie la conversion d'une valeur (valueFrom) donnée dans une certaine unité (unitFromName) vers une autre unité (unitToName) en utilisant une dimension physique Sitools2 dont le nom est indiqué (dimension). Cette méthode peut donc être utilisée dans la méthode *createPredicats(Request request, ArrayList<Predicat> predicats)* afin de créer une requête cohérente en fonction de l'unité associée à un attribut d'un jeu de données. Cette méthode renvoie une exception *ResourceException* lorsque l'unité ou la dimension physique envoyée n'est pas reconnue. Un exemple de l'utilisation de cette méthode est visible dans la méthode checkValues du NumericBetweenFilter.

3.1.5 Validation de la configuration

Les paramètres que l'administrateur a saisis peuvent être vérifiés par ce mécanisme de validation. Le squelette de la méthode est la suivante :

```
/**
* Gets the validator for this Filter
*
* @return the validator for the filter
*/
@Override
public Validator<AbstractFilter> getValidator() {
return new Validator<AbstractFilter>() {
@Override
public Set<ConstraintViolation> validate(AbstractFilter item) {
// TODO Auto-generated method stub
return null;
}
};
}
```

Deux états sont disponibles quand un problème de validation est détecté :

- **WARNING** : un warning apparaît lors de la configuration du filtre par l'administrateur,

- **CRITICAL** : un erreur critique apparaît lors de la configuration du filtre par l'administration. Dans ce cas, la configuration du filtre n'est pas sauvegardée

3.1.6 Description de l'API

Il est important que l'API du filtre soit décrite pour permettre à des clients, autre que l'IHM de SITools2, de comprendre comment utiliser les filtres. Voici un exemple montrant comme décrire les paramètres du filtre pour un WADL. Ce code est à utiliser dans le constructeur :

```
HashMap<String, ParameterInfo> rpd = new HashMap<String, ParameterInfo>();

ParameterInfo paramInfo

paramInfo = new ParameterInfo("p[#]", false, "xs:string", ParameterStyle.QUERY, "CONE_SEARCH_CARTESIEN|columnAlias1,columnAlias2,columnAlias3|RA_value|DEC_value|SR_Value");

rpd.put("0", paramInfo);

paramInfo = new ParameterInfo("c[#]", false, "xs:string", ParameterStyle.QUERY, "CONE_SEARCH_CARTESIEN|dictionaryName,conceptName1,conceptName2,conceptName3|RA_value|DEC_value|SR_Value");

rpd.put("1", paramInfo);

this.setRequestParamsDescription(rpd);
```

NOTE : Il est aussi possible de décrire l'API en surchargeant la méthode getRequestParamsDescription().

3.1.7 Ajout d'un Filtre dans Sitools2

Méthode avec Eclipse

Si vous utilisez Eclipse, et que vous avez récupérer l'ensemble des sources, l'ajout d'un convertisseur est très simple.

En effet, il suffit d'ajouter cette classe dans le projet extensions et d'ajouter le nom complet de la classe dans le fichier **fr.cnes.sitools.converter.FilterHelper**.

Il faut ensuite exécuter la tache Ant présente dans le projet extensions afin d'inclure la nouvelle classe dans la liste des extensions. Il faut également redémarrer Sitools2 pour prendre en compte le nouveau Jar.

Méthode sans Eclipse

Dans le cas ou vous n'utilisez pas Eclipse, et que le projet extensions n'existe pas, il faut créer la bonne arborescence « à la main ».

Cette arborescence est tout de même assez simple. Il suffit d'avoir à la racine les dossiers contenant les sources ainsi qu'un dossier META-INF/services qui contient un fichier nommé fr.cnes.sitools.converter.FilterHelper dans lequel on place le nom des classes.

Il faut ensuite

- Compiler les sources en ayant le jar de Sitools2 et de Restlet dans le classpath.
- Créer un jar en incluant les classes compilées et le dossier META-INF/services.
- Ajouter ce jar au classpath de Sitools2

3.1.8 Gestion des logs

Il existe plusieurs méthodes pour récupérer un Logger dans Sitools2.

Via la classe Application :

```
Application application = getApplication()
Logger logger = application.getLogger();
```

Directement via la classe Logger

```
Logger logger = Logger.getLogger(« nom de la classe courante »);
```

Via le Context :

```
Logger logger = Context.getLogger() ;
```

Ensuite pour utiliser le Logger :

```
logger.log(Level.INFO, "Message to log");
```

3.2 Développement d'un convertisseur en sortie

Les convertisseurs peuvent être utilisés pour effectuer une conversion après l'exécution de la requête par la base de données. Il dispose d'un ensemble de paramètres en entrée, de paramètres internes (constantes) et de paramètres en sortie. Les attributs du jeu de données qui possèdent des convertisseurs doivent être sélectionnés comme non filtrable et non ordonnable puisque le convertisseur ne propose pas la conversion inverse.

3.2.1 Introduction

La création d'un convertisseur en sortie se résume à l'implémentation d'une classe Java effectuant un traitement sur les données en sortie.

Cette classe doit hériter de la classe *AbstractConverter* et implémenter la méthode *getConversionOf*

Le squelette d'une telle classe est le suivant :

```
public class MyConverter extends AbstractConverter {
public LinearConverter() {
//initialisation des détails du convertisseur
//déclaration des paramètres
}
        @Override
public final Record getConversionOf(final Record rec) throws Exception {
        //récupération des paramètres
        //exécution du convertisseur
        return rec.
}
@Override
public Validator<AbstractConverter> getValidator() {
  return new Validator<AbstractConverter>() {
    @Override
    public Set<ConstraintViolation> validate(AbstractConverter item) {
      // validation des paramètres
    }
  }
}
}
```

3.2.2 Constructeur

Métadonnées du convertisseur

Le Constructeur possède un ensemble de métadonnées relatif au convertisseur, à savoir :

Métadonnées	Description	Méthodes à employer
L'auteur du convertisseur	L'auteur du convertisseur apparaît dans l'interface d'administration	setClassAuthor
Le propriétaire du convertisseur	Le propriétaire du convertisseur apparaît dans l'interface d'administration	setClassOwner
Le nom du convertisseur	Le nom du convertisseur apparaît dans l'interface d'administration	setName
La description du convertisseur	La description du convertisseur apparaît dans l'interface d'administration	setDescription
La version du	La version du convertisseur apparaît	setClassVersion

convertisseur	dans l'interface d'administration. Une vérification entre la version du filtre configuré et la version du filtre provenant du JAR est effectuée. Un **warning** est affichée si les versions diffèrent.	

Paramétrage de configuration du convertisseur

Lors du développement d'un convertisseur, le développeur peut créer des paramètres de configuration qui seront saisis par l'administrateur du système. Pour cela il suffit d'instancier des objets *ConverterParameter(String name, String description, ConverterParameterType type)* et de les ajouter au convertisseur par l'utilisation de la méthode *addParam(final ConverterParameter param)*.

Lors de l'instanciation d'un objet *ConverterParameter*, les paramètres suivants doivent être spécifiés :

Paramètre	Description
name	Nom du paramètre qui sera affiché dans l'interface d'administration pour la configuration du filtre
description	Description du paramètre qui sera affiché dans l'interface d'administration
converterParameterType	Type du paramètre entrainant un comportement différent dans l'IHM. Le type peut être soit **INTERN**, **IN**, **OUT** ou **IN_OUT**. Un type INTERN est une constante que l'administrateur devra saisir dans l'IHM d'administration. Un type IN est un attribut du jeu de données que l'utilisateur devra choisir parmi la liste exposée dans l'IHM d'administration. Un type OUT est un attribut du jeu de données (l'attribut pouvant être réel ou virtuel) dans lequel la réponse sera écrite. Un type IN_OUT est une fonction de transfert qui sera appliquée sur un attribut du jeu de données.

Un Convertisseur peut avoir deux constructeurs :

- Un par défaut, sans paramètre et obligatoire
- Un autre avec un objet de type *Context* en paramètre. Dans cet objet *Context*, on peut retrouver le jeu de données en utilisant l'attribut « **DATASET** » dans l'objet *Context*. Ce constructeur permet d'avoir une liste de paramètres qui dépend du jeu de données et rend le filtre encore plus souple.

3.2.3 La méthode getConvertionOf

Cette méthode est exécutée lors de l'exécution du convertisseur.

Il faut tout d'abord récupérer l'ensemble des paramètres du convertisseur. Pour ce faire il faut utiliser les méthodes *getInParam*, *getInternParam*, *getInOutParam* et *getOutParam* de la classe *AbstractConverter*.

Une fois les paramètres récupérés il suffit d'effectuer les traitements voulus sur les données

Dans cette méthode, on peut également récupérer certaines variables depuis le contexte. Ces variables sont SitoolsSettings, DataSetApplication ainsi que la requête HTTP effectuée précédemment.

Pour ce faire, il suffit de les récupérer dans les paramètres du contexte avec les clés : « **DataSetApplication**» et « **ContextAttributes.SETTINGS** » et « **REQUEST** ».

Depuis la DataSetApplication, il est possible d'obtenir une connexion à la base de données :

```
// recuperation complete du dataset
DataSet ds = datasetApp.getDataSet();
// recuperation d'une connexion à la source de données
Connection con = SitoolsDataSourceFactory.getInstance()
.getDataSource(ds.getDatasource().getName()).getConnection();
```

Où datasetApp est la DataSetApplication récupéré depuis le contexte.

Dans le cas où une valeur nulle est retournée par le convertisseur, aucune conversion n'est réalisée.

3.2.4 Validation de la configuration

Les paramètres que l'administrateur a saisis peuvent être vérifiés par ce mécanisme de validation. Le squelette de la méthode est la suivante :

```
/**
* Gets the validator for this Converter
*
* @return the validator for the converter
*/
@Override
public Validator<AbstractConverter> getValidator() {
return new Validator<AbstractConverter>() {
@Override
public Set<ConstraintViolation> validate(AbstractConverter item) {
// TODO Auto-generated method stub
return null;
}
};
}
```

Deux états sont disponibles quand un problème de validation est détecté :

- **WARNING** : un warning apparaît lors de la configuration du filtre par l'administrateur,

- **CRITICAL** : une erreur critique apparaît lors de la configuration du filtre par l'administration. Dans ce cas, la configuration du filtre n'est pas sauvegardée

3.2.5 Ajout d'un convertisseur dans Sitools2

Méthode avec Eclipse

Si vous utilisez Eclipse, et que vous avez récupéré l'ensemble des sources, l'ajout d'un convertisseur est très simple.

En effet, il suffit d'ajouter cette classe dans le projet extensions et d'ajouter le nom complet de la classe dans le fichier **fr.cnes.sitools.converter.ConverterHelper**.

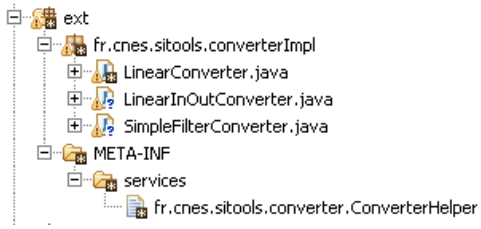
Il faut ensuite exécuter la tache Ant présente dans le projet extensions afin d'inclure la nouvelle classe dans la liste des extensions. Il faut également redémarrer Sitools2 pour prendre en compte le nouveau Jar.

Méthode sans Eclipse

Dans le cas où vous n'utilisez pas Eclipse, et que le projet extensions n'existe pas, il faut créer la bonne arborescence « à la main ».

Cette arborescence est tout de même assez simple. Il suffit d'avoir à la racine les dossiers contenant les sources ainsi qu'un dossier META-INF/services qui contient un fichier nommé fr.cnes.sitools.converter.ConverterHelper dans lequel on place le nom des classes.

L'arborescence ressemble donc à ceci :



Il faut ensuite

- Compiler les sources en ayant le jar de Sitools2 et de Restlet dans le classpath.
- Créer un jar en incluant les classes compilées et le dossier META-INF/services.
- Ajouter ce jar au classpath de Sitools2

3.2.6 Gestion des logs

Il existe plusieurs méthodes pour récupérer un Logger dans Sitools2.

Via la classe Application :

```
Application application = getApplication()
Logger logger = application.getLogger();
```

Directement via la classe Logger

```
Logger logger = Logger.getLogger(« nom de la classe courante »);
```

Via le Context :

```
Logger logger = Context.getLogger() ;
```

Ensuite pour utiliser le Logger :

```
logger.log(Level.INFO, "Message to log");
```

3.3 Développement d'une application plugin

Une application permet d'exposer des ressources relatives à une même entité métier (gestion des projets, datasets…). Chaque application à une définition de sécurité qui lui est propre.

3.3.1 Introduction

Le système permet l'ajout d'une application à la manière d'un plugin en utilisant le même mécanisme que les convertisseurs.

Pour ajouter une application if suffit :

- D'écrire sa classe, elle doit hériter de « *AbstractApplicationPlugin* »
- De l'ajouter dans le projet « extensions »
- D'ajouter le nom complet de la classe dans le fichier « fr.cnes.sitools.application.ApplicationPluginHelper» dans le dossier META-INF
- De compiler le projet « extensions » et de redémarrer le Sitools2.

Il y a deux étapes importantes dans le codage d'une application, le constructeur qui permet de définir les paramètres et la méthode *createInboundRoot* qui définit le comportement de l'application et attache les ressources. Le codage n'est pas différent de celui d'une application Restlet classique.

3.3.2 Le constructeur

Métadonnées de l'application

Le Constructeur possède un ensemble de métadonnées relatif à l'application, à savoir :

Métadonnées	Description	Méthodes à employer

L'auteur de l'application	L'auteur du convertisseur apparaît dans l'interface d'administration	getModel().setClassAuthor
Le propriétaire de l'application	Le propriétaire du convertisseur apparaît dans l'interface d'administration	getModel().setClassOwner
La version du filtre	La version du convertisseur apparaît dans l'interface d'administration. Une vérification entre la version du filtre configuré et la version du filtre provenant du JAR est effectuée. Un **warning** est affichée si les versions diffèrent.	getModel().setClassVersion
La catégorie	Catégorie dans laquelle l'application doit être classée (ADMIN, USER, ….)	getModel().setCategory

Paramétrage de configuration de l'application

Lors du développement d'une application, le développeur peut créer des paramètres de configuration qui seront saisis par l'administrateur du système. Pour cela il suffit d'instancier des objets *ConverterParameter()*, d'utiliser ses setteurs et de les ajouter à l'application par la méthode *addParameter(final ApplicationPluginrParameter param)*.

```
//création du paramètre
ApplicationPluginParameter param1 = new ApplicationPluginParameter();
param1.setName("param1");
param1.setDescription("Description de param1");
//ajout du paramètre
this.addParameter(param1);
```

Lors de l'instanciation d'un objet ApplicationPlugin*Parameter*, les paramètres suivants doivent être spécifiés :

Paramètre	Description
name	Nom du paramètre qui sera affiché dans l'interface d'administration pour la configuration du filtre
description	Description du paramètre qui sera affiché dans l'interface d'administration

Une application peut avoir deux constructeurs :

- Un par défaut, sans paramètre et obligatoire
- Un autre avec un objet de type *Context* en paramètre ainsi que le modèle de l'application. Dans cet objet *Context*, on peut retrouver le jeu de données en utilisant l'attribut « **DATASET** » dans l'objet *Context*. Ce constructeur permet d'avoir une liste de paramètres qui dépend du jeu de données et rend le filtre encore plus souple.

3.3.3 La méthode createInboundRoot

Cette méthode définit le fonctionnement de l'application et permet d'attacher différentes ressources. C'est dans cette méthode que l'on peut utiliser les valeurs des paramètres définit dans le constructeur. On récupère un paramètre de la façon suivante :

```
//récupération du paramètre, « param1 » est le nom du paramètre
ApplicationPluginParameter param = this.getParameter("param1");
```

3.3.4 Validation de la configuration

Les paramètres que l'administrateur a saisis peuvent être vérifiés par ce mécanisme de validation. Le squelette de la méthode est la suivante :

```
/**
* Gets the validator for this Converter
*
* @return the validator for the converter
*/
@Override
public Validator<AbstractApplicationPlugin> getValidator() {
return new Validator<AbstractApplicationPlugin>() {
@Override
public Set<ConstraintViolation> validate(AbstractApplicationPlugin item) {
// TODO Auto-generated method stub
return null;
}
};
}
```

Deux états sont disponibles quand un problème de validation est détecté :

- **WARNING** : un warning apparaît lors de la configuration du filtre par l'administrateur,
- **CRITICAL** : un erreur critique apparaît lors de la configuration du filtre par l'administration. Dans ce cas, la configuration du filtre n'est pas sauvegardée

3.3.5 Description de l'API

Il est important que l'API de l'application soit décrite pour permettre de la documenter. Voici un exemple montrant comme décrire une application.

```
@override
Public void sitoolsDescribe() {
    this.setName("ProxyApp");
    this.setAuthor("AKKA Technologies");
    this.setOwner("CNES");
    this.setDescription("Proxy Application Plugin");
}
```

3.3.6 Attachement des ressources plugins

Il est possible d'attacher des ressources plugin à une application plugin. Pour que cela soit possible il faut ajouter une ligne dans le constructeur de l'application

```
attachParameterizedResources(router);
```

3.3.7 Ajout d'une application dans SITools2

Méthode avec Eclipse

Si vous utilisez Eclipse, et que vous avez récupérer l'ensemble des sources, l'ajout d'une application est très simple.

En effet, il suffit d'ajouter cette classe dans le projet extensions et d'ajouter le nom complet de la classe dans le fichier **fr.cnes.sitools.applications.ApplicationHelper**.

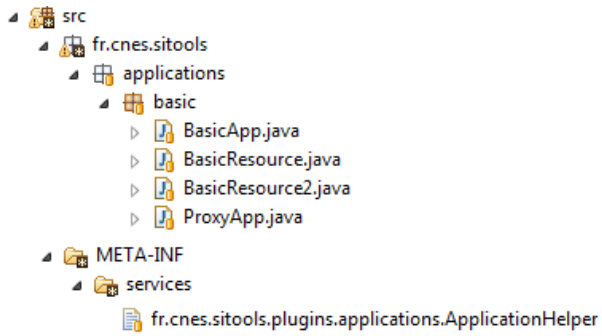
Il faut ensuite exécuter la tache Ant présente dans le projet extensions afin d'inclure la nouvelle classe dans la liste des extensions. Il faut également redémarrer Sitools2 pour prendre en compte le nouveau Jar.

Méthode sans Eclipse

Dans le cas ou vous n'utilisez pas Eclipse, et que le projet extensions n'existe pas, il faut créer la bonne arborescence « à la main ».

Cette arborescence est tout de même assez simple. Il suffit d'avoir à la racine les dossiers contenant les sources ainsi qu'un dossier META-INF/services qui contient un fichier nommé fr.cnes.sitools.applications.ApplicationHelper dans lequel on place le nom des classes.

L'arborescence ressemble donc à ceci :



Il faut ensuite

- Compiler les sources en ayant le jar de Sitools2 et de Restlet dans le classpath.
- Créer un jar en incluant les classes compilées et le dossier META-INF/services.
- Ajouter ce jar au classpath de Sitools2

3.3.8 Gestion des logs

Il existe plusieurs méthodes pour récupérer un Logger dans Sitools2.

Via la classe Application :

```
Application application = getApplication()
Logger logger = application.getLogger();
```

Directement via la classe Logger

```
Logger logger = Logger.getLogger(« nom de la classe courante »);
```

Via le Context :

```
Logger logger = Context.getLogger() ;
```

Ensuite pour utiliser le Logger :

```
logger.log(Level.INFO, "Message to log");
```

3.4 Développement d'une ressource plugin

Une ressource plugin est un service qui permet d'être attachée à une application. Dans la plupart des cas, cette ressource est attachée à un jeu de données et permet donc de le requêter afin de réaliser un traitement sur l'information contenue dans celui-ci.

***NOTE** : à l'heure actuelle, les applications d'administration et d'exposition de projets et de jeux de données, les applications plugins et l'administratonApplication sont les seules à pouvoir accueillir de tels plugins. Le design du système autorise cependant l'extension de ce comportement aux autres applications de SITools2 si le besoin se présentait.*

3.4.1 Introduction

Il existe deux types de « resource plugin » :

- L'un est simple et s'exécute d'une façon synchrone, deux classes sont nécessaires : une classe représentant le modèle et une autre représentant la ressource (le service à affectuer)

- L'autre est plus complexe et peut s'exécuter d'une façon synchrone ou asynchrone. Une classe supplémentaire est nécessaire : la façade

3.4.2 Création du modèle

La classe modèle, interagissant avec l'IHM d'administration, doit hériter de la classe *ResourceModel*.Le squelette d'une telle classe est le suivant :

```
public class BasicParametrizedResourceModel extends ResourceModel {
  /**
   * Constructor
   */
  public BasicParametrizedResourceModel() {
    super();
    setClassAuthor("AKKA");
    setClassVersion("0.1");
    setName("BasicParametrizedResourceModel");
    setDescription("Resource model");
    setClassOwner("CNES");
    setResourceClassName(fr.cnes.sitools.resources.basic.BasicParametrizedResource.class.getName());
        ResourceParameter textToSend = new ResourceParameter("text", "text to send",
ResourcesParameterType.PARAMETER_INTERN);
    textToSend.setValue("FooBar"); // default value
    addParam(textToSend);
    this.completeAttachUrlWith("basicresource/{yourtext}");
  }
```

Métadonnées de la ressource

Le Constructeur possède un ensemble de métadonnées relatif à la ressource, à savoir :

Métadonnées	Description	Méthodes à employer
L'auteur de la ressource	L'auteur de la ressource apparaît dans l'interface d'administration	setClassAuthor
Le propriétaire de la ressource	Le propriétaire de la ressource apparaît dans l'interface d'administration	setClassOwner
Le nom de la ressource	Le nom de la ressource apparaît dans l'interface d'administration	setName
La description de la ressource	La description de la ressource apparaît dans l'interface d'administration	setDescription

La version de la ressource	La version de la ressource apparaît dans l'interface d'administration. Une vérification entre la version du filtre configuré et la version du filtre provenant du JAR est effectuée. Un **warning** est affichée si les versions diffèrent.	setClassVersion

Paramétrage de configuration de la ressource

Lors du développement d'une ressource, le développeur peut créer des paramètres de configuration qui seront saisis par l'administrateur du système. Pour cela il suffit d'instancier des objets *ResourceParameter(String name, String description, ResourceParameterType type)* et de les ajouter au convertisseur par l'utilisation de la méthode *addParam(final ResourceParameter param)*.

Lors de l'instanciation d'un objet *ResourceParameter*, les paramètres suivants doivent être spécifiés :

Paramètre	Description
name	Nom du paramètre qui sera affiché dans l'interface d'administration pour la configuration du filtre
description	Description du paramètre qui sera affiché dans l'interface d'administration
ResourceParameterType	Type du paramètre entrainant un comportement spécifique dans l'IHM. Le type peut être soit **INTERN**, **USER_INPUT** ou **USER_GUI**. Un type INTERN est une constante que l'administrateur devra saisir dans l'IHM d'administration. Un type USER_INPUT est un attribut qui apparait à l'utilisateur dans l'IHM cliente lorsque la ressource est appelée. Un type USER_GUI est utilisé pour afficher une information à l'utilisateur (ex : souhaitez-vous continuer l'opération ?).

Une fois l'objet *ResourceParameter* créé, il est possible d'ajouter des comportements plus fins au niveau de l'IHM. Pour cela, il suffit d'utiliser la méthode *setValueType* de l'objet ResourceParameter. Le tableau ci-dessous définit les valeurs du setteur qui sont supportées

Valeur du setteur	Description
xs:dataset.columnAlias	Liste les colonnes visibles du jeu de données (uniquement compatible si la ressource est attachée à un jeu de données) dans l'IHM.
xs:enum[val1,val2,val3]	Liste dans l'IHM les choix parmi les valeurs val1, val2 et val3 (une seule valeur possible). La méthode setValue permet ensuite de positionner une valeur par défaut dans l'IHM.
xs:enum-mulitple[val1,val2,val3]	Liste dans l'IHM les choix parmi les valeurs val1, val2 et val3 (plusieurs valeurs possibles). La méthode setValue permet ensuite de positionner une valeur par défaut dans l'IHM
xs:enum-editable[val1,val2,val3]	Liste dans l'IHM les choix parmi les valeurs val1, val2 et val3 (éditable par l'administrateur). La méthode setValue permet ensuite de positionner une valeur par défaut dans l'IHM
xs:enum-editable-multiple[val1,val2,val3]	Liste dans l'IHM les choix parmi les valeurs val1, val2 et val3 (éditable par l'administrateur, et plusieurs choix possibles). La méthode setValue permet ensuite de positionner une valeur par

	défaut dans l'IHM
xs:boolean	Liste dans l'IHM les choix parmi true ou false. La méthode setValue permet ensuite de positionner une valeur par défaut dans l'IHM
xs:image	Liste dans l'IHM les choix parmi la liste des images uploadées sur le serveur, possibilité de choisir une image.
xs :dictionary	Liste dans l'IHM les choix parmi la liste des dictionnaires (l'attribut « name » du dictionnaire est utilisé lors de la sauvegarde du paramètre).

***Note** : Il est tout à fait possible d'ajouter d'autres « xs » (ex : « xs :interger ») qui ne sont pas définis dans le tableau ci-dessous. Cependant ces « xs » n'étant pas implémentés dans l'IHM, ces « xs » auront le même comportement que un paramètre INTERN.*

Une ressource peut avoir deux constructeurs :

- Un par défaut, sans paramètre et obligatoire
- Un autre avec un objet de type *Context* en paramètre. Dans cet objet *Context*, on peut retrouver le jeu de données en utilisant l'attribut « **DATASET** » dans l'objet *Context*. Ce constructeur permet d'avoir une liste de paramètres qui dépend du jeu de données et rend le filtre encore plus souple.

Spécification d'une partie de l'URI de la ressource

La ressource étant attachée à la DataSetApplication, l'URI de la ressource contient donc l'URI de la dataSetApplication. Pour configurer la partie de l'URI située après celle de la DataSetApplication, il suffit d'appeler la méthode completeAttachUrlWith.

L'implémentation du service

Le modèle fournit également la classe d'implémentation dans lequel est contenu le code métier : setResourceClassName(fr.cnes.sitools.resources.basic.BasicParametrizedResource.class.getName())

Pour que le client Sitools2 sache quelles sont les méthodes qu'il peut invoquer, il faut renseigner la liste des méthodes autorisées. Il s'agit de la liste des méthodes en majuscule séparée par le caractère « | » :

```
this.getParameterByName("methods").setValue("POST|GET")
```

Spécifier le type de sélection dans l'interface utilisateur

Pour que le client Sitools2 sache le type de sélection qu'il peut appliquer, il convient de renseigner le paramètre « **dataSetSelection** » :

```
this.setDataSetSelection(DataSetSelectionType.ALL)
```

L'énumération ***DataSetSelectionType*** contient les différentes possibilités :

```
/** No selection authorized */
  NONE,
  /** Single selection */
  SINGLE,
  /** Muliple selection */
  MULTIPLE,
  /** All the dataset can be selected */
  ALL
```

Ajouts de paramètres de façon dynamique

On peut ajouter dans un modèle de ressources des paramètres de façon dynamique en fonction du contexte de l'application. Cela permet par exemple d'avoir un paramètre par colonne de dataset ou de proposer la liste des dictionnaires en xs :enum à l'administrateur.

Il s'agit uniquement de rajouter des paramètres qui seront ensuite rempli par l'administrateur. Une fois le modèle sauvegardé, il n'est plus possible d'en ajouter avec cette méthode.

Il faut surcharger la méthode « initParametersForAdmin » dans l'implémentation du modèle. Cette méthode prend en paramètre un objet « Context » ce qui permet de récupérer certains paramètres :

- appClassName : le nom de la classe d'application parent
- parent : l'identifiant le la classe d'application parent

Compatibilité des ressources avec des applications

Il est également possible de spécifier le type d'application avec laquelle **cette ressource est compatible**. Par exemple, cela permet d'être sûr qu'une application qui gère des jeux de données ne sera pas attachée à une application de gestion des projets.

Pour spécifier le type de l'application, il suffit d'appeler, dans le modèle de la ressource, la méthode *setApplicationClassName(className)* avec en paramètre le nom complet de la classe d'application :

```
this.setApplicationClassName("fr.cnes.sitools.dataset.DataSetApplication")
```

On peut également saisir le nom d'une super classe, toutes les classes qui hérite de celle-ci seront donc compatibles. Dans le cas ou rien n'est spécifié, toutes les applications sont compatibles.

Au niveau de l'interface d'administration, seules les ressources compatibles avec une application donnée sont affichées ce qui permet d'éviter les erreurs.

Dans le cas ou une ressource serait attachée à une application non compatible, cette ressource est tout de même attachée mais un Warning est affiché dans le LOG. Lorsque l'on appelle cette ressource, elle retourne une erreur 503 pour spécifier que le service n'est pas disponible.

Chaque ressource peut être rattachée à une application qui étend ParamterizedApplication. Aujourd'hui, seule les ¨ProjectAdministration, ProjectApplication, DatasetAdministration et DatasetApplication sont concernées.

3.4.3 Notions communes aux « resource plugin »

Dans un Resource plugin, il est possible de faire quasiment n'importe quel traitement. Elle peut également répondre à l'API de consultation des données d'un jeu de données. On peut donc :

- **Accéder à la requête sur les données du jeu de données**

```
// On récupère le context
Context context = getContext();
// On récupère la DataSetApplication
DataSetApplication datasetApp = (DataSetApplication) getApplication();
// On génère un nouveau DataSetExplorerUtil
DataSetExplorerUtil dsExplorerUtil = new DataSetExplorerUtil(datasetApp, getRequest(), getContext());
//On récupère les paramètres de connexion
DatabaseRequestParameters params = dsExplorerUtil.getDatabaseParams();
//On instancie une requête à l'aide d'une factory
DatabaseRequest databaseRequest = DatabaseRequestFactory .getDatabaseRequest(params);
//On crée une requête en fonction du type de la requête (distinct ou normale)
if (params.getDistinct()) {
        databaseRequest.createDistinctRequest();
} else {
        databaseRequest.createRequest();
}
```

On peut ensuite boucler sur les enregistrements avec la méthode boolean :nextResult() et récupérer un enregistrement avec la méthode Record :getRecord().

```
try {
        while (databaseRequest.nextResult()){
                Record rec = databaseRequest.getRecord();
```

A la fin de l'exécution, il faut fermer la connexion à la source de données :

```
} finally {
        if (databaseRequest != null) {
                databaseRequest.close();
        } }
```

Attention à bien fermer la connexion à la base de données. De plus, dans le cas ou la connexion est utilisée dans le code d'une représentation, il faut créer la requête et la relâcher dans la méthode write (ne pas prendre la connexion dans le constructeur et la relâcher dans le write).

Appliquer les convertisseurs

On peut également appliquer les convertisseurs définis sur le jeu de données. Pour ce faire, on doit récupérer la DatasetApplication et appliquer les convertisseurs.

```
//On récupère la DatasetApplication
DataSetApplication datasetApp = (DataSetApplication)getApplication();
//récupération des convertisseurs
ConverterChained converterChained = datasetApp.getConverterChained();
```

Pour appliquer un convertisseur, on appelle la méthode Record :getConversionOf(Record) de la classe ConverterChained sur chacun des enregistrements. Dans le cas ou le convertisseur utilise la requête HTTP, il convient de l'ajouter dans le contexte du convertisseur avant son exécution. Il s'agit d'un traitement assez simple :

```
if (converterChained != null) {
   converterChained.getContext().getAttributes().put("REQUEST", request);
}
```

Ensuite on applique les convertisseurs sur chacun des enregistrements :

```
if (Util.isSet(converterChained)) {
    rec = converterChained.getConversionOf(rec);
}
```

On peut également accéder au jeu de données en le récupérant dans la DatasetApplication.

Obtenir une connexion directe à la source de données

```
// recuperation complete du dataset
DataSet ds = datasetApp.getDataSet();
// recuperation d'une connexion à la source de données
Connection con = SitoolsDataSourceFactory.getInstance()
.getDataSource(ds.getDatasource().getName()).getConnection();
```

Accès aux paramètres de la requête HTTP

Une resource plugin peut enrichir l'API de consultation des enregistrements d'un jeu de données en récupérant d'autres paramètres dans la requête HTTP. Il suffit de récupérer leur valeur dans l'objet Request.

```
getRequest().getResourceRef().getQueryAsForm().getFirst("limit")
```

Ajouter un prédicat

Il est possible d'ajouter un prédicat pour enrichir la requête à la base de données avant son exécution. Il suffit d'utiliser le code suivant :

```
Predicat pred = new Predicat()
pred.setLogicOperator("AND")
pred.setLeftAttribute(primaryKey)
//primaryKey est un objet de type Column
pred.setCompareOperator("=")
pred.setRightValue("?")
// params est un Objet de type DatabaseRequest
params.getPredicats().add(pred)
```

Créer un fichier

Il est possible de créer des fichiers lors de l'exécution d'une ressource. Pour ce faire, il existe une méthode dans la classe OrderResourceUtils (disponible dans le projet des extensions):

public static String addFile(Representation repr, String urlDest, ClientInfo clientInfo, Context context) throws SitoolsException

Cette méthode prend une Representation en paramètre, ainsi qu'une Url où sera créé le fichier. Cette Url doit contenir le nom du fichier. Elle prend également un ClientInfo pour prendre en compte les droits sur le répertoire de destination ainsi que le Context pour récupérer certaines variables.

Il faut spécifier le média type de la représentation en fonction du type de fichier. Il faut également que l'extension du fichier soit compatible avec ce média type. (Par exemple si on veut créer un fichier nommé « test.html », il faut que le media type de la représentation soit « MediaType.TEXT_HTML »).

Afin d'obtenir une Url de destination disponible, il existe une méthode :

public static String getUserAvailableFolderPath(SvaTask svaTask, Context context)

Cette Url de destination pointe dans l'espace utilisateur de l'utilisateur ayant demandé l'exécution de la ressource ou dans le dossier temporaire si la ressource est demandé sans utilisateur.

Copier un fichier existant

De la même manière que précédemment, il est possible de copier un fichier présent à une Url donnée dans une autre Url.

public static String copyFile(String fileUrl, String destUrl, ClientInfo clientInfo, Context context) throws SitoolsException

Zipper un ensemble de fichier

Il est possible de faire un fichier zip contenant un ensemble de fichier. Il suffit de passer une liste de nom de fichier accessible et le chemin local d'un fichier zip sur le serveur.

```
public final void zipFiles(ArrayList<String> listOfFiles, String destFilePath, ClientInfo clientInfo, Context context) throws SitoolsException
```

Le fichier zip ne peut être créé qu'en local sur le serveur. Il peut ensuite être copié dans un autre répertoire puis supprimé. Par exemple, dans le cas de la ressource de commande (OrderResource), le fichier zip est créé dans le dossier temporaire (SitoolsSettings.getInstance().getTmpFolderUrl()) puis recopié dans l'espace utilisateur.

Initialiser et utiliser un objet partagé par plusieurs ressources

Il peut être nécessaire dans certains cas d'initialiser un objet et de vouloir l'utiliser dans des ressources différentes.

Pour ce faire, il faut stocker cet objet dans le « Context » du Composant (Component) Restlet. Sitools2 est construit autour d'un seul composant auquel on attache toutes les applications, il est donc connu de toutes les applications et ressources.

Lors de l'exécution d'une ressource, dans la méthode doInit, on vérifie que l'objet existe dans le Context du composant. S'il existe on le récupère sinon on l'initialise et on le place dans le Context. On peut utiliser ce code comme exemple :

```
//Récupération du SitoolsSettings, il contient un pointeur vers le composant
SitoolsSettings settings = ((SitoolsSettings) getContext().getAttributes().get(ContextAttributes.SETTINGS));
/Récupération du Context du composant
Context contextComposant = settings.getComponent().getContext();
Object object = contextComposant.getAttributes().get(<SOME KEY>);
if (object == null) {
   //Initialisation de l'objet
   object = new <SOME OBJECT>();
   //Ajout de l'objet initialisé
   contextComposant.getAttributes().put(<SOME KEY>,object);
}
```

Accès à des fichiers distants

Pour que les ressources puissent accéder à des fichiers ou des services distants, il peut être nécessaire de configurer un proxy. Cette configuration s'effectue dans le fichier « sitools.properties ». Il faut donc saisir les propriétés suivantes :

- Starter.WITH_PROXY=true ou false
- Starter.PROXY_HOST=Proxy host
- Starter.PROXY_PORT=Proxy port
- Starter.PROXY_USER=Proxy user
- Starter.PROXY_PASSWORD=Proxy Password
- Starter.NONPROXY_HOST=Liste des domaines (séparés par un |) qui ne seront pas envoyé via le proxy pour les appels de Sitools2 en tant que client.

Pour accéder à un fichier, il suffit d'utiliser la méthode getFile défini précédemment. Pour accéder à un service distant en utilisant le proxy, il faut ajouter quelques instructions pour prendre en compte l'authentification du proxy :

```
//création de la requête
request = new Request(Method.GET, url);
//application de l'authentification pour le proxy
if((ProxySettings.getProxyAuthentication()!=null)&&reqGET.getProxyChallengeResponse() == null){
      request.setProxyChallengeResponse(ProxySettings.getProxyAuthentication());
}
//execution de la requête
org.restlet.Response r = context.getClientDispatcher().handle(request);
On peut également utiliser une classe utilitaire ajouté récemment, l'authentification pour le proxy sera ajouté automatiquement. :
ClientResourceProxy clientResourceProxy = new ClientResourceProxy(url, Method.GET);
ClientResource clientResource = clientResourceProxy.getClientResource();
Representation representation = clientResource.handle();
```

3.4.4 L'implémentation d'une ressource synchrone

Un plugin de ressource hérite obligatoirement de la classe abstraite SitoolsParametrizedResource.

Elle peut implémenter l'ensemble des méthodes HTTP, par exemple pour GET :

```
package fr.cnes.sitools.resources.basic;
import org.restlet.representation.Representation;
import org.restlet.representation.StringRepresentation;
import fr.cnes.sitools.common.resource.SitoolsParametrizedResource;
public class BasicParametrizedResource extends SitoolsParametrizedResource {
  @Override
  public void doInit() {
    super.doInit();
    this.textSent = (String) this.getRequestAttributes().get("yourtext");
  }
  @Override
  public Representation get() {
    return new StringRepresentation("This is a dynamic resource sending the text : "
        + getModel().getParametersMap().get("text").getValue());
  }
  @Override
  public void sitoolsDescribe() {
    setName(this.getClass().getName());
    setDescription("Test class for dynamic resource");
  }
}
```

Dans cet exemple, il est possible de récupérer la valeur de « yourtext » à partir de l'url, via la méthode :

```
(String) this.getRequestAttributes().get("yourtext")
```

Pour récupérer la valeur des autres paramètres on utilise les méthodes classiques :

```
getModel().getParametersMap().get("text").getValue()
```

Gestion des logs

Il existe plusieurs méthodes pour récupérer un Logger dans Sitools2.

Via la classe Application :

```
Application application = getApplication()
Logger logger = application.getLogger();
```

Directement via la classe Logger

```
Logger logger = Logger.getLogger(« nom de la classe courante »);
```

Via le Context :

```
Logger logger = Context.getLogger() ;
```

Ensuite pour utiliser le Logger :

```
logger.log(Level.INFO, "Message to log");
```

3.4.5 Implémentation d'une « ressource plugin » avec gestion des tâches

Il est désormais possible d'exécuter la ressource de manière synchrone ou asynchrone avec une gestion des tâches d'exécutions. Il est donc possible d'utiliser les ressources de la même manière que les ressources synchrones.

Les filtres définis sur un jeu de données peuvent être appliqué à cette ressource ce qui permet d'exécuter une requête similaire à l'API records dans une ressource.

3.4.5.1 Introduction

Il existe deux fonctionnements pour une ressource, le développeur devra choisir l'un ou l'autre en fonction de ses besoins :

- Le résultat est un objet « Representation » : dans ce fonctionnement, il faut retourner une représentation qui sera stockée sur le serveur. Elle pourra être consommée une fois l'exécution terminée mais ne pourra l'être qu'une seule fois.

- Le résultat est un fichier : dans ce fonctionnement, le résultat de l'exécution de la ressource doit être stocké dans un fichier et on doit retourner « null ». Il faut également renseigner l'url du fichier résultat dans la tâche courante pour pouvoir y accéder plusieurs fois à la fin de l'exécution.

Il existe deux moyens de passer des paramètres à une ressource :

- En utilisant l'API de consultation des enregistrements d'un jeu de données

- En passant l'url d'un fichier dans le corps de la requête. Ce fichier doit contenir une liste d'enregistrements contenant au moins la clé de chaque enregistrement. Ce fichier doit être en format JSON et être de la forme : {"orderRecord":{"records":[liste des records]}}. L'utilisation de ce dernier moyen demande de traiter le corps de la requête dans la « façade » dans le cas d'un appel POST, PUT ou DELETE

Il faut implémenter 3 classes pour mettre en place une ressource plugin avec gestion des tâches. Comme pour une ressource plugin classique il faut un modèle et une ressource. Il faut également une ressource « façade » qui servira d'entrée dans la ressource et définira l'interface du service.

3.4.5.2 Création du modèle

Le modèle s'implémente comme un modèle de ressource classique, cependant il faut que ce modèle hérite de la classe TaskResourceModel. Cette classe met en place 4 nouveaux paramètres : l'url d'une image (image), le nom de la classe d'implémentation de la ressource (resourceImplClassName) , le type d'exécution que pourra spécifier l'administrateur (runTypeAdministration), et le type d'exécution choisi par le client (runTypeClient). .

On peut spécifier ces paramètres avec les méthodes setResourceImplClassName(), setRunTypeAdministration(), setRunTypeClient() et setImage().

Exemple pour une ressource de commande de fichiers :

```
public class OrderResourceModel extends TaskResourceModel {
  /**
   * Constructor
   */
  public OrderResourceModel() {
    super();
    setClassAuthor("AKKA");
    setClassVersion("0.1");
    setName("OrderResourceModel");
    setDescription("Resource model for Order Resource");
    /** Resource facade */
    setResourceClassName("fr.cnes.sitools.resources.tasks.order.OrderResourceFacade");
    /** Resource d'implémentation */
    setResourceImplClassName("fr.cnes.sitools.resources.tasks.order.OrderResource");
    setRunTypeAdministration(TaskRunType.TASK_DEFAULT_RUN_SYNC);
    ResourceParameter paramColUrl = new ResourceParameter("colUrl", "Colum containing data url for order",
        ResourceParameterType.PARAMETER_INTERN);
    /** Type de paramètre pour lister les colonnes du dataset */
    paramColUrl.setValueType("xs:dataset.columnAlias");
    ResourceParameter param2 = new ResourceParameter("zip",
        "(true or false) If the data needs to be zipped at the end", ResourceParameterType.PARAMETER_USER_INPUT);
    param2.setValue("false");
    /** Type de colonne booléen */
    param2.setValueType("xs:boolean");
    this.addParam(paramColUrl);
    this.addParam(param2);
    this.setApplicationClassName(DataSetApplication.class.getName());
    this.getParameterByName("methods").setValue("POST");
    this.setDataSetSelection(DataSetSelectionType.ALL);
```

```
}
```

L'implémentation du service

Le model fournit également :

- la classe d'implémentation dans lequel est contenu la façade : setResourceClassName(fr.cnes.sitools.resources.basic.BasicParametrizedResource.class.getName())
- la classe d'implémentation dans lequel est contenu le code métier: setResourceImplClassName("fr.cnes.sitools.resources.tasks.order.OrderResource")
- le mode par défaut de la classe d'implémentation : setRunTypeAdministration(TaskRunType.TASK_DEFAULT_RUN_SYNC)

3.4.5.3 La façade

Il s'agit de la ressource qui sera attachée à l'application, c'est donc celle qui servira d'entrée. Elle doit donc définir les différentes méthodes autorisées. Elle doit ensuite faire appel à la classe TaskUtils qui gère les appels synchrones ou asynchrones et les tâches d'exécutions.

Exemple pour une ressource de commande de fichiers :

```
public class OrderResourceFacade extends SitoolsParameterizedResource {
 /**
  * Description de la ressource
  */
 @Override
 public void sitoolsDescribe() {
  setName("OrderResourceFacade");
  setDescription("Resource to order data");
 }
 /**
  * Description WADL de la methode POST
  *
  * @param info
  *        The method description to update.
  */
 @Override
 public void describePost(MethodInfo info) {
  info.setDocumentation("Method to order data from a dataset");
  info.setIdentifier("order");
  addStandardPostOrPutRequestInfo(info);
  DataSetExplorerUtil.addDatasetExplorerGetRequestInfo(info);
```

```
    DataSetApplication application = (DataSetApplication) getApplication();
    DataSetExplorerUtil.addDatasetExplorerGetFilterInfo(info, application.getFilterChained());
    addStandardResponseInfo(info);
    addStandardInternalServerErrorInfo(info);
  }
  /**
   * Create the order
   *
   * @param represent
   *          the {@link Representation} entity
   * @param variant
   *          The {@link Variant} needed
   * @return a representation
   */
  @Post
  public Representation createOrder(Representation represent, Variant variant) {
    processBody();
    return TaskUtils.execute(this, variant);
  }
  /**
   * process the body and save the request entity {@link Representation}
   */
  public void processBody() {
    Representation body = this.getRequest().getEntity();
    if (body != null && body.isAvailable() && body.getSize() > 0) {
      Form bodyForm = new Form(body);
      getContext().getAttributes().put(TaskUtils.BODY_CONTENT, bodyForm);
    }
  }
}
```

Cette façade doit également traiter le corps de la requête dans le cas ou la ressource serait appelée de manière asynchrone. En effet lors d'un appel asynchrone, la réponse est retournée avant la fin de l'exécution effective de la requête (dans la classe OrderResource). Lorsque la réponse est retourné, le corps de la requête est supprimé par Restlet et ne peut plus être utilisé, il faut donc le traiter avant l'appel au TaskUtils, par exemple en utilisant la méthode processBody.

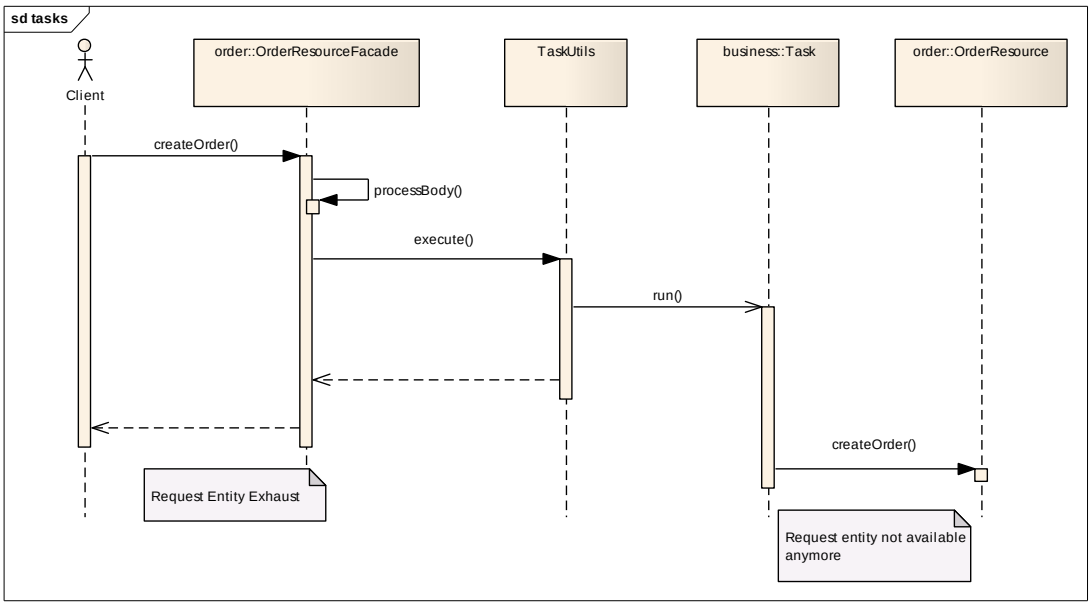


Figure 23 : Diagramme de séquence très simplifié illustrant le déroulement d'un appel à OrderResource

Attention : Dans le cas d'une ressource acceptant une sélection d'enregistrement de dataset, il est impératif de recopier et d'appeler la méthode processBody comme dans l'exemple ci-dessus sinon lors d'un appel POST ou PUT, les sélections ne seront pas appliquées.

3.4.5.4 La ressource implémentation

Liste des objets passés dans le contexte

Certains objets sont passés dans le contexte aux ressources, pour chacun d'entre eux il existe une constante définie dans la classe TaskUtils :

- PARENT_APPLICATION : l'application parente de la ressource
- LOG_FOLDER : le répertoire de log de la ressource
- BODY_CONTENT : le contenu du corps de la requête (doit précédemment avoir été rempli au niveau de la façade et mis dans le contexte de la ressource avec la même clé)

Précaution d'usage

- Il faut éviter de prendre les connexions à la base de données à la création du Resource (dans le code d'implémentation de la ressource, ou le doInit) et il faut les relâcher lors de la récupération du résultat (dans le code de la Representation associée).

- Il faut éviter les traitements trop longs ou bien prévenir l'administrateur de forcer le mode d'exécution à asynchrone.
- Dans une ressource, il est possible de faire quasiment n'importe quel traitement il convient donc de faire attention aux traitements effectués.

Gestion des logs

Chaque exécution de ressource (tâche) possède son propre fichier de log. Ce log est présent dans le dossier data/resources_logs. (Configurable dans le fichier de properties avec la clé : Starter.APP_RESOURCE_LOGS_DIR)

Pour ajouter quelque chose au log, il faut récupérer le logger et utiliser la méthode log :

```
// on récupère le logger de la tache
java.util.logging.Logger logger = task.getLogger();
// on log
logger.log(Level.INFO, "log");
```

A noter que le système d'analyse de log (paragraphe 10.3) n'utilise pas ces fichiers pour l'analyse.

3.4.6 Création d'une ressource de commande

Une des ressources particulières offerte par Sitools2 est la ressource de commande. Elle permet d'effectuer une commande de fichiers, d'en garder une trace au niveau du serveur et de notifier l'administrateur à chaque commande utilisateur.

Architecture et modèle de classe

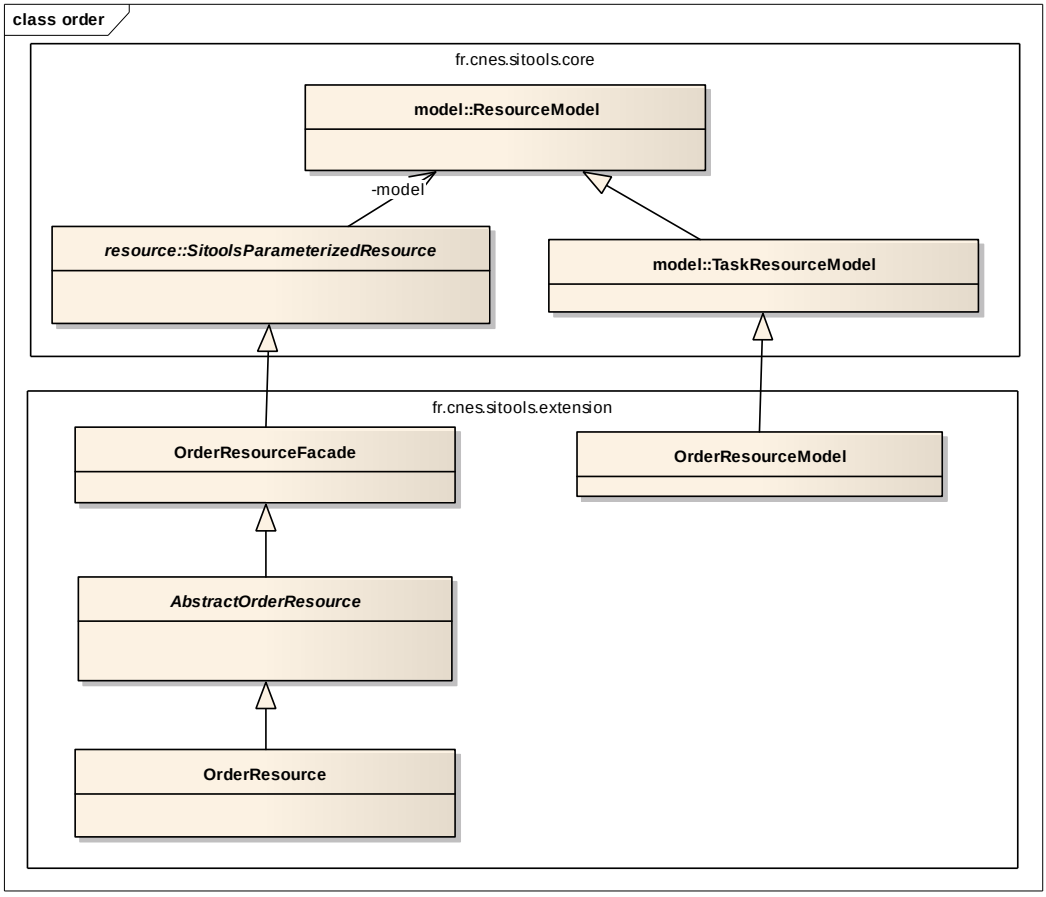


Figure 24 : Modèle de classe général OrderResource

Ce diagramme de classe très simplifié (pas d'affichage des méthodes ni des attributs) présente les différentes classes utilisées (hors classes utilitaires) pour la ressource de commande.

OrderResourceFacade est la façade de cette ressource,

OrderResourceModel est le modèle de cette ressource. On peut surcharger cette classe pour ajouter d'autres paramètres et effectuer des validations supplémentaires.

OrderResource est l'implémentation de cette ressource, on peut, comme on peut le voir dans le schéma suivant, hériter de cette classe ou directement de AbstractOrderResource pour spécialiser le processus de commande.

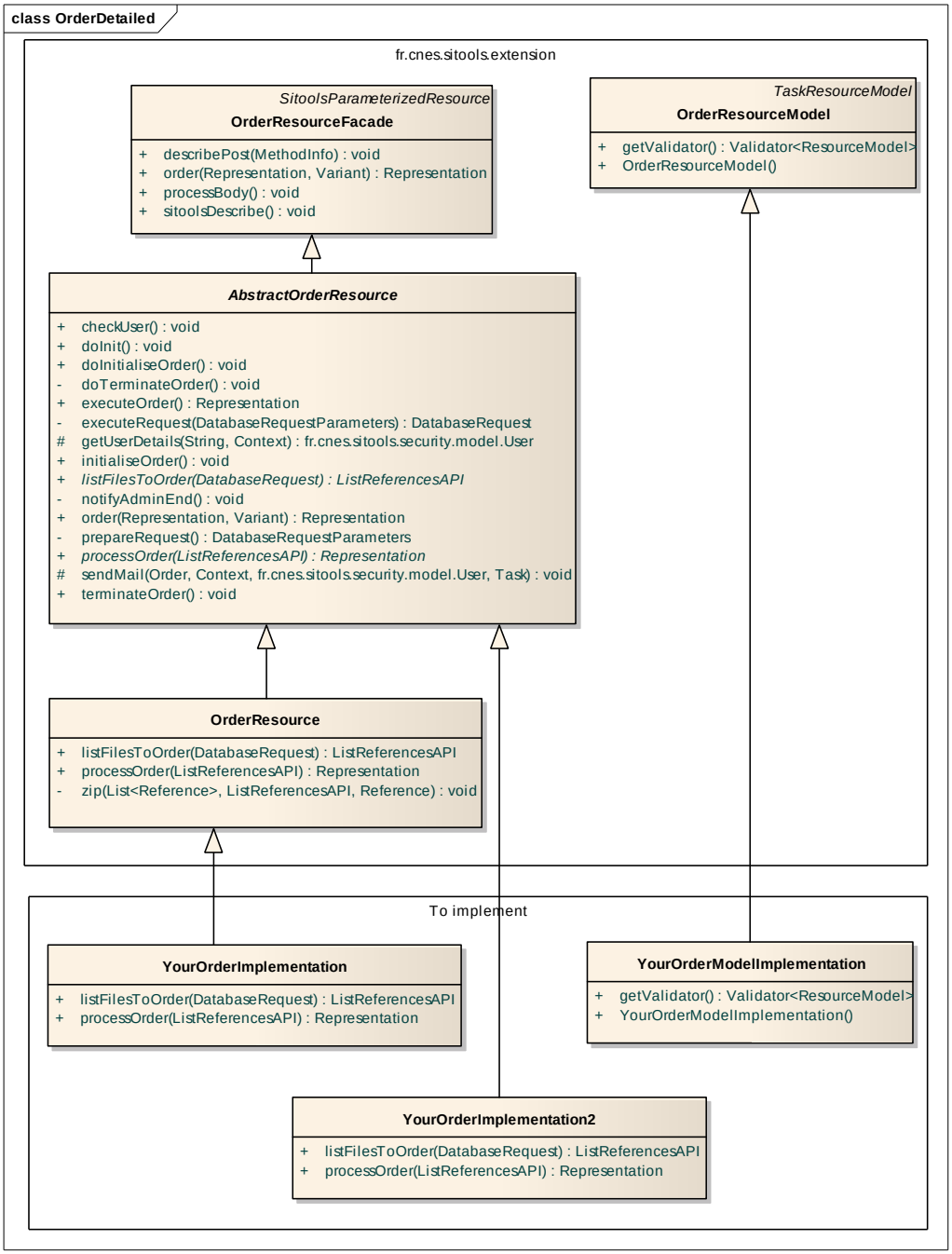


Figure 25 : Modèle de classe détaillé ResourceOrder

Sitools2 fournit une classe abstraite qui réalise le processus de commande moins certaines actions à implémenter (AbstractOrderResource).

Il fournit également une implémentation par défaut, qui réalise la commande de fichier disponible à une URL et les copie ou les Zip dans l'espace utilisateur de l'utilisateur qui a demandé la commande (OrderResource). L'URL de chaque fichier est récupérée dans une colonne du Dataset.

Il est donc possible de réaliser sa propre implémentation de commande en implémentant la classe AbstractOrderResource ou en surchargeant OrderResource.

Processus de commande

Cette partie présente les différentes étapes du processus de commande avec le nom des méthodes réalisant chacune des étapes :

- Initialisation : initialiseOrder()
    - Vérification de l'utilisateur : checkUser()
    - Préparation de la requête à la base de données : prepareRequest()
    - Initialisation de la commande Sitools : doInitialiseOrder()
- Exécution de la commande : executeOrder()
    - Activation de la commande Sitools
    - Exécution de la requête à la base de données : executeRequest()
    - Création de la liste de fichier à commander : listFilesToOrder() (A surcharger)
    - Copie de la liste des fichiers à commander dans l'espace administrateur
    - Exécution de la commande : processOrder() (A surcharger)
- Finalisation de la commande : terminateOrder()
    - Finalisation de la commande Sitools : doTerminateOrder()
    - Notification de l'administrateur : notifyAdminEnd()

Pour simplifier, les 2 méthodes principales à surcharger sont listFilesToOrder() et processOrder().

Les classes utilitaires disponibles

Sitools2 fournit un ensemble de classes utilitaires pour simplifier l'implémentation d'un processus de commande.

Elles sont situées dans le package fr.cnes.sitools.resources.order.utils et sont au nombre de 3 :

ListReferencesAPI

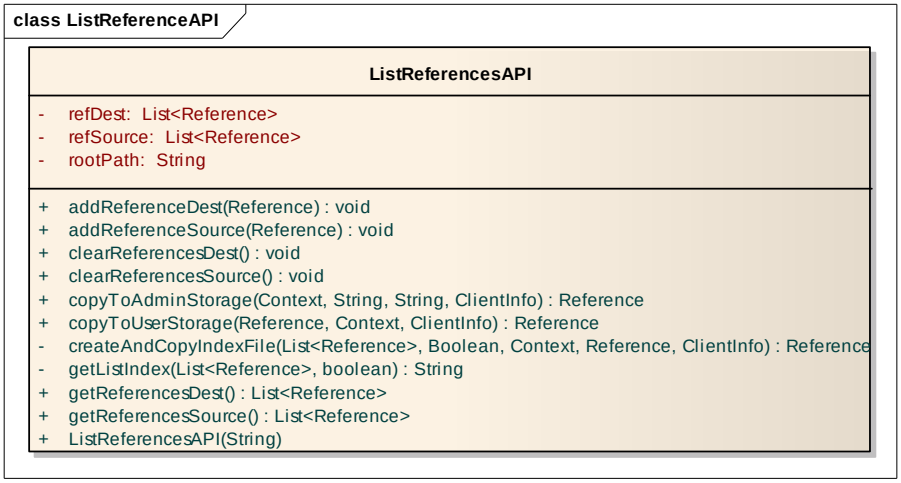


Figure 26 : Classe ListReferenceAPI

Cette classe permet de gérer des listes de fichier à commander et de générer des fichiers d'index avec des liens vers les fichiers commandés.

Plus précisément elle permet de gérer 2 listes de Reference (Objet Restlet qui représente un lien vers une URI), une pour les fichiers sources (liste des fichiers à commander avec leurs URI sources) et une pour les fichiers destinations (liste des fichiers commandé avec leurs URI cibles).

Elle permet également de générer et de copier des fichiers d'index à partir de chacune des listes de Reference.

La méthode copyToAdminStorage génère la liste des fichiers sources et la copie dans l'espace administrateur des commandes (permet d'avoir une trace des fichiers commandés).

La méthode copyToUserStorage génère la liste des fichiers cible et la copie à une Reference donnée, en principe dans l'espace utilisateur où la commande est copiée. L'appel à cette méthode est à la charge du développeur.

OrderAPI

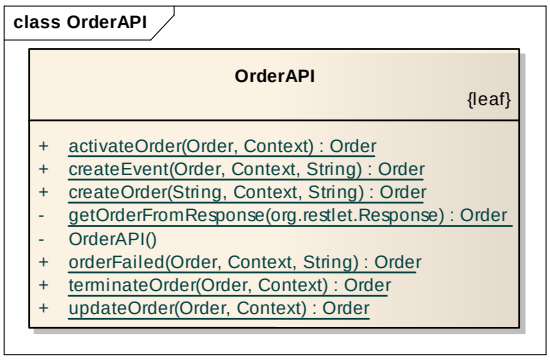


Figure 27 : Classe OrderAPI

Cette classe permet de gérer un objet commande au sens de Sitools2 afin de garder une trace visible depuis l'interface administrateur et utilisateur de chacune des commandes. Les différentes méthodes de cette classe permettent de créer, de modifier et de modifier le statut d'une commande. Elles permettent également d'ajouter un événement à cette commande pour permettre à l'utilisateur de suivre l'avancement de sa commande.

OrderResourceUtils

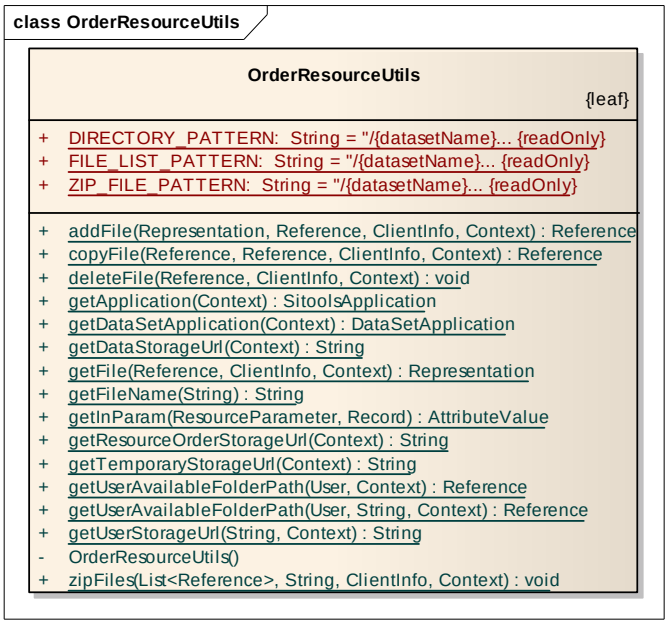


Figure 28 : Classe OrderResourceUtils

Cette classe offre un ensemble de méthodes utiles pour l'implémentation d'une ressource de commande. Elle permet de gérer des fichiers, d'en créer, d'en modifier, d'en copier ou d'en supprimer. Elle permet également de créer un ZIP à partir d'un ensemble de Reference.

Exemple d'implémentation

La classe OrderResource du projet fr.cnes.sitools.extension peut servir d'exemple pour une implémentation de ressource de commande.

3.4.7 Validation de la configuration

Les paramètres que l'administrateur a saisis peuvent être vérifiés par ce mécanisme de validation. Le squelette de la méthode est la suivante :

```
/**
* Gets the validator for this Converter
*
* @return the validator for the converter
*/
@Override
public Validator<ReosurceModel> getValidator() {
return new Validator<ResourceModel>() {
@Override
public Set<ConstraintViolation> validate(ResourceModel item) {
// TODO Auto-generated method stub
return null;
}
};
}
```

Deux états sont disponibles quand un problème de validation est détecté :

- **WARNING** : un warning apparaît lors de la configuration du filtre par l'administrateur,
- **CRITICAL** : une erreur critique apparaît lors de la configuration du filtre par l'administration. Dans ce cas, la configuration du filtre n'est pas sauvegardée

3.4.8 Ajout des ressources dans SITools2

Il faut enfin modifier le fichier fr.cnes.sitools.plugins.ResourceHelper en y ajoutant la classe Model précédemment créée.

3.5 Développement de filtres de sécurité

Le système permet l'ajout de filtres de sécurité. La finalité de ces filtres est de personnaliser l'accès à certaines ressources de SITools2.

3.5.1 Introduction

L'implémentation d'un filtre de sécurité se résume à l'implémentation :

- D'un modèle héritant de FilterModel
- D'une ressource héritant de org.restlet.routing.Filter

3.5.2 Création du modèle

Le squelette d'une telle classe est le suivant :

```
public class DataStorageAuthorizerModel extends FilterModel {
public DataStorageAuthorizerModel() {
        super();
        setName("DataStorageAuthorizer");
        setDescription("Customizable datastorage directory authorizer");
        setClassAuthor("AKKA Technologies");
        setClassOwner("CNES");
        setClassVersion("0.1");
        setClassName("fr.cnes.sitools.filter.authorizer.DataStorageAuthorizerModel");

        setFilterClassName("fr.cnes.sitools.filter.authorizer.DataStorageAuthorizer");

        /**
        * Parameter for the log directory
        */
        FilterParameter logDir = new FilterParameter("logdir", "Storage logging directory",
FilterParameterType.PARAMETER_INTERN);
        addParam(logDir);

        /**
        * Parameter for authorize or block
        */
        FilterParameter authorize = new FilterParameter("authorize", "Authorize true|false",
FilterParameterType.PARAMETER_INTERN);
        authorize.setValue("true"); // default true
        authorize.setValueType("xs:boolean");
        addParam(authorize);
        }

        @Override
```

```
public Validator<FilterModel> getValidator() {
// TODO validator for DataStorageAuthorizerModel
return null;
}
```

Métadonnées du filtre de sécurité

Le Constructeur possède un ensemble de métadonnées relatif au filtre de sécurité, à savoir :

Métadonnées	Description	Méthodes à employer
L'auteur du filtre de sécurité	L'auteur de la ressource apparaît dans l'interface d'administration	setClassAuthor
Le propriétaire du filtre de sécurité	Le propriétaire de la ressource apparaît dans l'interface d'administration	setClassOwner
Le nom du filtre de sécurité	Le nom de la ressource apparaît dans l'interface d'administration	setName
La description du filtre de sécurité	La description de la ressource apparaît dans l'interface d'administration	setDescription
La version du filtre de sécurité	La version de la ressource apparaît dans l'interface d'administration. Une vérification entre la version du filtre configuré et la version du filtre provenant du JAR est effectuée. Un **warning** est affichée si les versions diffèrent.	setClassVersion

Paramétrage de configuration du filtre de sécurité

Lors du développement d'une ressource, le développeur peut créer des paramètres de configuration qui seront saisis par l'administrateur du système. Pour cela il suffit d'instancier des objets *FilterParameter(String name, String description, FilterParameterType type)* et de les ajouter au convertisseur par l'utilisation de la méthode *addParam(final FilterParameter param)*.

Lors de l'instanciation d'un objet Filter*Parameter*, les paramètres suivants doivent être spécifiés :

Paramètre	Description
name	Nom du paramètre qui sera affiché dans l'interface d'administration pour la configuration du filtre
description	Description du paramètre qui sera affiché dans l'interface d'administration
filterParameterType	Type du paramètre entrainant un comportement différent dans l'IHM. Le type peut être soit **INTERN** ou **IN**. Un type INTERN est une constante que l'administrateur devra saisir dans l'IHM d'administration. Un type IN est un attribut du jeu de données que l'utilisateur devra choisir parmi la liste exposée dans l'IHM d'administration.

Cette classe doit étendre de FilterModel.

Dans le constructeur, on peut ajouter des paramètres, comme pour d'autres plugins, qui seront utilisés ultérieurement. Dans ce modèle on définit également le nom de la classe Modèle et le nom de la classe de filtre associée.

```
setFilterClassName("fr.cnes.sitools.filter.authorizer.DataStorageAuthorizer")
```

3.5.3 Création du filtre

Filtres génériques

Il s'agit d'un filtre Restlet, cette classe doit donc étendre la classe org.restlet.routing.Filter.

Elle doit définir son propre constructeur avec le contexte en paramètre :

```
public DataStorageAuthorizer(Context context) {
  this.context = context;
  filterId = (String) this.context.getAttributes().get("FILTER_ID");
  filterModel = (FilterModel) this.context.getAttributes().get("FILTER_MODEL");
  FilterParameter authorizeParameter = filterModel.getParameterByName("authorize");
  if (authorizeParameter != null) {
    try {
      bauthorize = Boolean.parseBoolean(authorizeParameter.getValue());
    }
    catch (Exception e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
  }
}
```

Cas particulier : filtre sur les datastorages

Le filtre DataStorageAuthorizer est une implémentation de filtre qui étend org.restlet.security.Authorizer . Il faut donc surcharger la méthode suivante :

```
public boolean authorize(Request arg0, Response arg1) {
}
```

3.5.4 Validation de la configuration

Les paramètres que l'administrateur a saisis peuvent être vérifiés par ce mécanisme de validation. Le squelette de la méthode est la suivante :

```
/**
* Gets the validator for this Converter
*
* @return the validator for the converter
*/
@Override
public Validator<FilterModel> getValidator() {
return new Validator<FilterModel>() {
@Override
public Set<ConstraintViolation> validate(FilterModel item) {
// TODO Auto-generated method stub
return null;
}
};
}
```

Deux états sont disponibles quand un problème de validation est détecté :

- **WARNING** : un warning apparaît lors de la configuration du filtre par l'administrateur,
- **CRITICAL** : une erreur critique apparaît lors de la configuration du filtre par l'administration. Dans ce cas, la configuration du filtre n'est pas sauvegardée

3.5.5 Ajout d'un filtre de sécurité dans SITools2

Il faut enfin modifier le fichier fr.cnes.sitools.plugins.FilterPluginHelper en y ajoutant la classe Modèle précédemment créée.

3.5.6 Gestion des logs

Il existe plusieurs méthodes pour récupérer un Logger dans Sitools2.

Via la classe Application :

```
Application application = getApplication()
Logger logger = application.getLogger();
```

Directement via la classe Logger

```
Logger logger = Logger.getLogger(« nom de la classe courante »);
```

Via le Context :

```
Logger logger = Context.getLogger() ;
```

Ensuite pour utiliser le Logger :

```
logger.log(Level.INFO, "Message to log");
```

3.6 Développement de convertisseurs d'unité

Le système permet le développement de systèmes d'unité et de convertisseurs spécifiques entre unités n'ayant pas les mêmes dimensions physiques.

3.6.1 Création d'un nouveau système d'unités

Le développement majeur consiste en la création d'un nouveau système d'unité par l'extension de la classe SystemOfUnits du framework javax.measure (intégré au jar fr.cnes.sitools.core.jar).

Cette classe, qui doit suivre le design pattern d'un singleton, sera alors constituée de plusieurs attributs de classe (mot-clé « static » en Java) :

- un champ général UNITS (type Set<Unit< ?>>) qui contient l'ensemble des unités du système
- d'un champ pour chaque unité créée (cf. exemple ci-après).

De plus, pour assigner un symbole à l'unité, chacune des unités instanciées doit s'enregistrer au près du singleton du framework javax.measure LocalFormat, qui gère le mapping entre symbole et unités. Cet enregistrement est possible via la méthode LocalFormat.getInstance().getSymbolMap().label(unit, symbol).

Lors de l'instanciation de chaque unité, celle-ci doit également être ajouté à l'attribut statique UNITS décrit précédemment. Comme chaque unité est elle-même statique, cette instanciation doit donc s'effectuer via une méthode intermédiaire (méthode addUnit dans l'exemple donné ci-après).

```
public final class AstronomicSystem extends SystemOfUnits {
  /** Set of units in the system */
  public static final Set<Unit<?>> UNITS = new HashSet<Unit<?>>();
  /** Add Angstrom unit, equivalent to a tenth of a nanometer */
  public static final Unit<Length> ANGSTROM = addUnit(new AstronomicUnit<Length>("" + (char) 143, METRE, new MultiplyConverter(1e-10)));
… others units defined the same way …
  /** Instance for singleton */
  private static AstronomicSystem instance = null;
  /** Singleton private constructor */
  private AstronomicSystem() {}
```

```
/** Get singleton
 * @return the instance
 */
public static synchronized AstronomicSystem getInstance() {
  if (instance == null) {
    instance = new AstronomicSystem();
  }
  return instance;
}
@Override
public Set<Unit<?>> getUnits() {
  return Collections.unmodifiableSet(UNITS);
}
/** Add a unit to the system and return it
 * @param <U> the unit type to instantiate
 * @param unit the unit to add
 * @return the unit added
 */
private static <U extends Unit<?>> U addUnit(U unit) {
  UNITS.add(unit);
  return unit;
}
```

Voici également un exemple de spécialisation des unités :

```
public class AstronomicUnit<Q extends Quantity<Q>> extends TransformedUnit<Q>  {

  /** Long for serialization */
  private static final long serialVersionUID = 1L;

  /** Holds the symbol. */
  private final transient String symbol;
  /**
   * Constructor with symbol
   * @param symbol the symbol used for the unit
   * @param parent the base unit definition
   * @param converter the converter to the parent
   */
  public AstronomicUnit(String symbol, Unit<Q> parent, UnitConverter converter) {
    super(parent, converter);
    this.symbol = symbol;
    LocalFormat.getInstance().getSymbolMap().label(this, symbol);
  }
}
```

3.6.2 Création d'un convertisseur d'unité spécifique

La conversion entre deux unités n'ayant pas la même dimension physique (d'une longueur vers une fréquence par exemple, en utilisant la constante universelle c), n'est pas automatique dans le framework javax.measure.

Afin de rendre possible la conversion, SITools2 permet la création de convertisseurs spécialisés à partir de la classe mère SitoolsUnitConverter.

Ce convertisseur spécialisé a deux attributs principaux : l'unité de départ et celle d'arrivée, initialisées dans le constructeur du convertisseur. Ces deux unités peuvent être simple, le framework étant par ailleurs capable de gérer la conversion entre les multiples de ces deux unités. Dans l'exemple donné ci après, on se contente donc de prendre comme unité de départ l'Hertz (fréquence) et comme unité d'arrivée le mètre (longueur). Il est à noter que ce convertisseur sera également capable d'effectuer automatiquement la conversion inverse (arrivée vers départ) sans développement supplémentaire.

La suite du développement consiste à implémenter cinq méthodes :

- getBaseToTargetConverter : doit retourner un convertisseur de l'unité de base vers l'unité cible, en passant généralement vers le système métrique (les convertisseurs vers le système métrique sont automatiquement générés par le framework javax.measure)
- inverse : doit retourner le convertisseur inverse du précédent,
- convert(double x) : renvoie la valeur x convertie, de l'unité de base vers l'unité cible,
- convert(Number x, MathContext ctx) : renvoie la valeur convertie dans un certain contexte mathématique,
- isLinear() : renvoie true si la conversion est linéaire.

L'héritage nécessite l'implémentation d'autres méthodes, mais qui n'ont aucun impact sur la conversion (equals et hashCode). Ci-après nous montrons l'exemple d'un convertisseur permettant la conversion entre le domaine des fréquences et celui des longueurs d'onde :

```
public class FrequencyWavelengthConverter extends SitoolsUnitConverter {
 /** Serial number */
 private static final long serialVersionUID = 1L;
 /** Speed of light */
 private static final Velocity C = QuantityFactory.getInstance(Velocity.class).create(2.99792e8, MetricSystem.METRES_PER_SECOND);
 /** Constructor */
 public FrequencyWavelengthConverter() {
  super();
  this.setStartUnit(MetricSystem.HERTZ);
  this.setTargetUnit(MetricSystem.METRE);
 }

 @Override
 public UnitConverter getBaseToTargetConverter() {
  UnitConverter startConverter = this.getStartUnit().getConverterToMetric();
  UnitConverter targetConverter = this.getTargetUnit().getConverterToMetric().inverse();
  return targetConverter.concatenate(this).concatenate(startConverter);
 }
 @Override
 public UnitConverter inverse() {
  UnitConverter startConverter = this.getStartUnit().getConverterToMetric().inverse();
  UnitConverter targetConverter = this.getTargetUnit().getConverterToMetric();
  return targetConverter.concatenate(this).concatenate(startConverter);
 }
 @Override
 public double convert(double value) {
```

```
    return C.doubleValue(MetricSystem.METRES_PER_SECOND) / value;
  }
  @Override
  public Number convert(Number value, MathContext ctx) {
    return convert(value.doubleValue());
  }
  @Override
  public boolean equals(Object cvtr) {
    return (cvtr instanceof FrequencyWavelengthConverter);
  }
  @Override
  public int hashCode() {
    return 0;
  }
  @Override
  public boolean isLinear() {
    return true;
  }
}
```

3.6.3 Découverte des systèmes d'unité et des convertisseurs par SITOOLS2

Afin que les systèmes d'unité et les convertisseurs spécifiques soient découverts par SITools2, une dernière classe doit être crée puis déclarée dans un fichier Helper correspondant. Cette classe a donc pour unique but d'initialiser les nouveaux systèmes d'unités et les convertisseurs afin de les rendre disponibles.

Cette classe hérite de DimensionHelper, et consiste donc en l'enregistrement des convertisseurs d'unité (méthode registerUnitConverter(SitoolsUnitConverter)) et l'enregistrement des systèmes d'unité disponibles (méthode registerSystem(SystemOfUnit)).

Nous montrons ci-après un exemple d'Helper associé :

- au convertisseur dont on a montré l'exemple précédemment,
- au système d'unité dont on a montré l'exemple précédemment,
- aux systèmes d'unité présents dans le framework.

```
public class SitoolsUnitConverterHelper extends DimensionHelper {

  /**
   * Constructor
   */
  public SitoolsUnitConverterHelper() {
    super();

    /**
     * Registering all converters and all systems here
     */
    this.registerUnitConverter(new FrequencyWavelengthConverter());
    this.registerSystem(AstronomicSystem.getInstance());
    this.registerSystem(MetricSystem.getInstance());
    this.registerSystem(USCustomarySystem.getInstance());
  }
}
```

Pour finir, cette classe doit être répertoriée – comme pour les autres plugins – dans un fichier nommé fr.cnes.sitools.units.UnitsHelper, présent dans le répertoire META-INF/services. Ce référencement consiste uniquement à la présente du nom de la classe dans le fichier :

```
# Contenu de fr.cnes.sitools.units.UnitsHelper #
fr.cnes.sitools.units.helper.SitoolsUnitConverterHelper
```

3.6.4 Gestion des logs

Il existe plusieurs méthodes pour récupérer un Logger dans Sitools2.

Via la classe Application :

```
Application application = getApplication()
Logger logger = application.getLogger();
```

Directement via la classe Logger

```
Logger logger = Logger.getLogger(« nom de la classe courante »);
```

Via le Context :

```
Logger logger = Context.getLogger() ;
```

Ensuite pour utiliser le Logger :

```
logger.log(Level.INFO, "Message to log");
```

3.7 Sécurisation de l'interface d'administration

Sitools2 permet de configurer un Filtre pour filtrer de façon globale les requêtes envoyées au serveur. Le filtre utilisé est configurable dans le fichier sitools.properties en modifiant la propriété : Security.filter.class. Il convient de mettre le nom complet d'une classe Java surchargeant la classe SecurityFilter.

Pour l'instant il existe 2 filtres disponible, un qui permet un filtrage IP Intranet/Extranet et un autre qui permet en plus un filtrage de type BlackList IP.

Filtrage-IP Intranet/Extranet

Dans Sitools2, on distingue 2 domaines réseaux que l'on appelle Intranet et Extranet. Intranet correspond à un ensemble de sous réseaux restreints. Extranet correspond à tous les réseaux possibles (aucun filtrage).

Cela permet de bloquer les accès à certaines applications sur l'adresse IP d'origine de l'appel, avant toutes les autres vérifications de sécurité (autorisation, authentification …).

Il s'agit donc d'un filtre Restlet (classe SecurityFilter du package fr.cnes.sitools.security.filter) que l'on attache avant chaque application. La catégorie de l'application permet de savoir si une application est en Intranet ou en Extranet.

La configuration du filtrage s'effectue dans le fichier sitools.properties grâce aux propriétés suivantes :

- Security.Intranet.net : Contient la liste des adresses réseaux du domaine Intranet séparées par des | (pipe)
- Security.Intranet.mask : Contient le masque de sous réseaux commun aux adresses réseaux
- Security.Intranet.<catégorie d'application> : (true ou false) pour spécifier si une catégorie d'application appartient au domaine Intranet ou non. (<catégorie d'application> correspond à une entrée de l'énumération fr.cnes.sitools.common.model.Category)

Le nom de la classe correspondant à ce filtre est : fr.cnes.sitools.security.filter.SecurityFilter

Configuration sans sécurité

Afin de configurer Sitools2 sans aucune séparation Intranet/Extranet, il suffit de configurer l'ensemble des variables Security.Intranet.<catégorie d'application> à false.

Configuration conseillée

Il est conseillé de configurer l'ensemble des applications d'administration et système dans le domaine Intranet. Les autres peuvent être laissées dans le domaine extranet.

Filtrage BlackList IP

Ce filtre permet, en plus du filtrage IP Intranet/Extranet de refuser les requêtes provenant d'une liste finie d'adresse IP. Cette liste est définie dans le fichier « sitools.properties » avec la propriété « Security.filter.blacklist ».Il s'agit simplement d'une liste d'adresse IP séparée par des « | ».

Exemple : Security.filter.blacklist=192.168.0.1|192.168.0.2

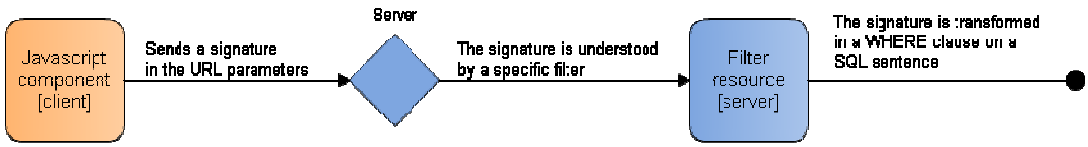
Pour ne configurer aucun filtrage, il faut configurer la propriété Security.filter.blacklist avec la valeur « 0.0.0.0 ».

Le nom de la classe correspondant à ce filtre est : fr.cnes.sitools.security.filter.IPBlackListFilter

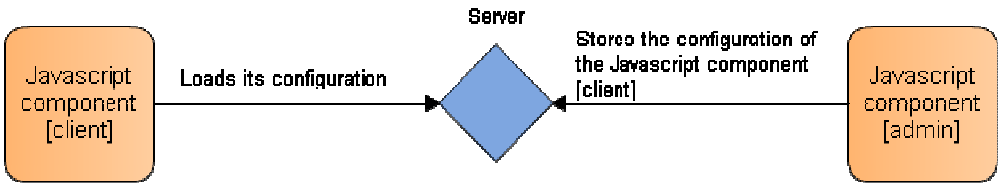
4 Développement des extensions de l'IHM

4.1 Développement d'un type de composant de formulaire

Un composant de formulaire est un morceau de code JavaScript qui va permettre d'offrir à l'utilisateur une information ou une interaction sous forme graphique. Cela peut aller d'un simple texte à afficher jusqu'à un ensemble d'éléments d'un formulaire (combo box, …). Ce code source se veut réutilisable et personnalisable par une CSS.



Chaque composant de formulaire possède sa propre signature qui est envoyé par une requête AJAX au serveur de SITools2. Cette signature est ensuite interceptée par un « filtre de requête » en JAVA permettant de transformer la signature du composant en une clause d'une requête SQL. Chaque composant est donc associé à une classe JAVA au niveau serveur.



Afin de personnaliser le composant de formulaire JavaScript client, on utilise un composant JavaScript d'administration. Le composant d'administration stocke les variables dans un modèle de données sur le serveur. Cette configuration est ensuite chargée par le composant de formulaire JavaScript client.

De nouveaux composants de formulaires clients ainsi que leur composant d'administration et leur « filtre de requête » associé peuvent être ajoutés par configuration par l'administrateur. L'administrateur peut ensuite utiliser ces composants graphiques pour créer son formulaire de requête sur chaque « jeu de données » par l'intermédiaire d'une interface WYSIWYG.

4.1.1 Création d'un type de composant

4.1.1.1 Au niveau de l'IHM administrateur

En tant qu'administrateur, on peut ajouter de nouveaux types de composants de formulaires réalisés par les développeurs.

Un type de composant est défini par :

Paramètre	Description
Type	Type représente une clé unique sur la liste des types de composants. Ce type fait parti de la signature du composant qui est envoyé au serveur. Le serveur intercepte la signature et utilise le filtre qui implémente la signature du composant
componentDefaultHeight	la hauteur par défaut dans l'interface d'administration
componentDefaultWidth	largeur par défaut dans l'interface d'administration
Javascript Admin Object	Le nom de l'objet à instancier pour que l'administrateur puisse paramétrer ce type de composant
Javascritp User Object	Le nom de l'objet à instancier pour que l'utilisateur puisse visualiser ce type de composant
File Url Admin	Le chemin pour accéder à la description de l'objet js permettant à l'administrateur de configurer ce type de composant
File Url User	Le chemin pour accéder à la description de l'objet js permettant à l'utilisateur de visualiser ce type de composant
Image	Le chemin vers l'url de l'image du rendu de ce type de composant
Priority	La priorité dans laquelle seront inclus dans la page HTML les scripts. Ceci permet de développer des objets étendant des objets déjà créés en étant sûr que cela ne génèrera pas d'erreurs. Exemple : 1 développeur souhaitant créer un textField particulier pourra étendre sitools.common.forms.components.TextField et devra mettre une priorité supérieure à 2.

4.1.1.2 Au niveau Java

Chaque composant de formulaire utilisera l'API de recherche du jeu de données. Pour enrichir cette API de recherche, on peut attacher différents filtres à un jeu de données. Ces filtres permettent de contraindre le jeu de données défini par l'administrateur. Dans le cas d'une table SQL, les filtres ajoutent des contraintes dans une clause WHERE.

Il est donc primordial, pour que les composants de formulaires fonctionnent correctement, que des filtres soient implémentés et rattachés au jeu de données. Il faut également, lors de la définition d'un jeu de données, que les attributs du jeu de données (colonnes d'une table SQL pour une SGBD) soient configurés avec l'attribut « Filter » pour pouvoir appliquer un filtre sur ces attributs.

Chaque développeur pourra développer ses propres filtres pour enrichir l'API de recherche d'un jeu de données.

4.1.1.3 Au niveau JavaScript

Les objets JavaScripts utilisateur et administrateur doivent être définis dans des fichiers et déposés sur le serveur à un endroit accessible. Le paramétrage du type de composant permettra de faire le lien vers ces fichiers, et de les inclure dynamiquement.

Objet administrateur

Le développeur devra créer un objet JavaScript qui sera instancié lorsque l'administrateur souhaitera ajouter ce nouveau type de composant dans l'un de ses formulaires.

Cet objet doit étendre de la classe sitools.component.forms.oneParam.abstractForm, et implémenter une méthode _onValidate().

Cette méthode décrit le comportement lorsque l'administrateur valide son paramétrage pour ce type de composant : ajouter ou modifier un des composants du formulaire en cours d'édition.

Elle accepte comme paramètres la grid qui contient tous les composants du formulaire, et l'action en cours.

Un exemple : client-admin/js/forms/componentsAdminDef/OneParam/TextField.js

```
sitools.component.forms.oneParam.TextField                                    =
Ext.extend(sitools.component.forms.oneParam.withoutValues, {


    _onValidate : function (action, gridFormComponents) {
        //méthode appelée lorsque l'administrateur valide son formulaire.
        var f = this.getForm();

        if (action == 'modify') {
            var rec = gridFormComponents.getSelectionModel().getSelected();
            rec.set("nom de l'attribut", "valeur dans le formulaire") ;
        } else {
            //En insertion on ajoute une nouvelle entrée au store :
            var formComponentsStore = gridFormComponents.getStore();
            formComponentsStore.add(new Ext.data.Record(attributs));
        }
        return true;
    }
});
```

Mis en forme : Anglais (États Unis)

Objet Utilisateur

Le développeur devra également créer un objet JavaScript qui sera instancié lorsque l'utilisateur affichera le formulaire.

Cet objet doit étendre de la classe Ext.form.Field et implémenter une méthode getParameterValue() qui retournera la signature du composant (qui sera lue par l'API de recherche). Cet objet doit également contenir une propriété stype : « sitoolsFormContainer »

Cet objet sera instancié avec les paramètres décrient dans le paragraphe suivant.

Client-serveur

Le modèle d'échange de chaque composant de formulaire est défini par l'objet ParameterDTO, Chaque composant de formulaire contient donc une liste de ParameterDTO.

Pour l'affichage du formulaire il faut donc convertir le paramètre en composant ExtJs :

Dans le fichier formParameterToComponent.js, en fonction du type, on instancie l'objet défini dans le type de composant en lui passant les paramètres tels que définis par l'administrateur.

Aujourd'hui, ce paramétrage est limité à :

Paramètre	Description
ParameterId	id du paramètre (unique dans un formulaire)
Values	tableau de valeurs possibles pour le composant (optionnel)
Code	un tableau d'alias d'attributs d'un jeu de données (dans le cas d'une SGBD, les colonnes d'une table)
Type	le type de composant
Label	le label du composant
Height	la hauteur du composant
WidthBox	la largeur du composant
ValueSelection	pour certains composants, la façon dont on charge les valeurs (S : valeurs spécifiques, D valeurs provenant d'une table de données)
ParentParam	le paramètre parent pour les composants liés entre eux
DataUrl	l'url à laquelle on peut charger des données
AutoComplete	paramétrage pour le textfield (true or false)
FormId	l'identifiant du formulaire
Unit	l'unité relative au composant
Css	une classe Css additionnelle qui est référencée dans main.css
DefaultValues	les valeurs par défaut éventuelles

Il sera fait appel à la méthode getParameterValue() du composant pour effectuer la recherche sur le jeu de données. Ainsi, lors de la génération de la requête, chaque composant renseigné créera un paramètre sous la forme :

P[n] =getParameterValue()

Chaque paramètre de non p[n] sera consommé par les filtres rattachés au jeu de données et génèrera des prédicats sous les conditions prévues dans chaque filtre.

4.1.1.4 Création des prédicats au niveau serveur

Voir paragraphe 3.1 sur le développement d'un filtre de requête en entrée.

4.1.2 Création de composants avec unités

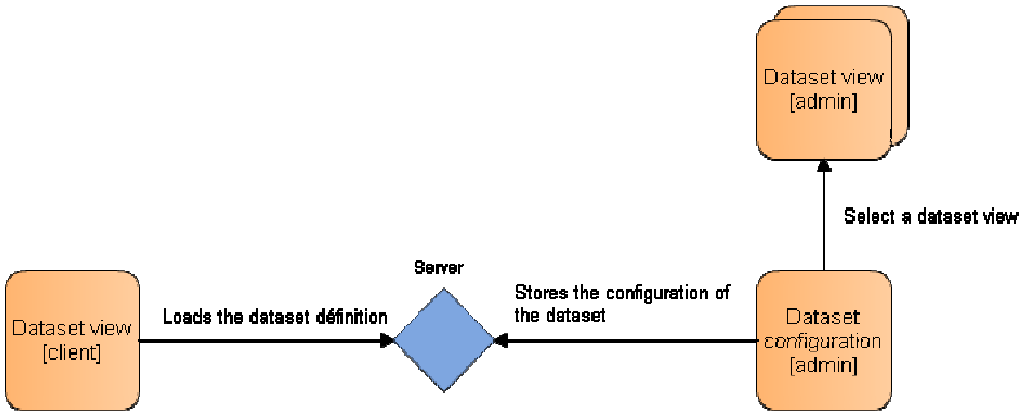
Les unités sont gérées au niveau de la description JavaScript d'un composant sous la forme d'un Ext.Button dont le comportement actuel a été défini dans chacun des composants. Le comportement de ce bouton côté client consiste principalement à :

- Afficher une unité si l'attribut du jeu de données à laquelle le composant est lié en défini une (rien dans le cas contraire)
- Afficher une liste d'unité disponible en fonction de la dimension qui a été associée avec le composant graphique du formulaire

Remarque : Dans le cas des composants de formulaire de type cone-search, l'unité utilisée au niveau de la requête serveur est le degré. Si une unité est saisie au niveau du formulaire, une conversion sera effectuée entre cette unité et le degré, l'unité de référence des attributs dans le jeu de données pour le composant graphique « cone search » n'est pas pris en compte.

4.2 Développement de dataset views

Un « dataset view » est un composant JavaScript permettant de présenter les données provenant d'un jeu de données dans l'IHM cliente. Il existe actuellement deux « dataset Views » disponibles dans SITools2 : le premier permet de présenter les données sous un format tabulaire pour les données homogènes alors que le deuxième permet de présenter les données hétérogènes.



Afin d'utiliser un « dataset view » pour un jeu de données, il convient d'en sélectionner un parmi une liste définie lors de la configuration du jeu de données. La configuration du jeu de données est ensuite sauvegardée sur le serveur. Lors de l'affichage du jeu de données, le client charge tout d'abord la configuration du jeu de données depuis le serveur. A partir de cette configuration, les données sont visualisées dans le « dataset view » qui a été sélectionné lors de la phase d'administration.

4.2.1 Création d'une Dataset View

En tant qu'administrateur, on peut ajouter de nouvelles vues graphiques (ex : live-grid) pour la visualisation des jeux de données

Une vue est définie par :

Paramètre	Description
Nom	Nom de la dataview (affiché ultérieurement dans l'ihm administration de datasets)
Description	Description succinte de la dataview (également affiché dans l'ihm administration)
JavaScript Object	Objet javascript qui sera instancié lorsque ce dataset sera visualisé
File Url	Url du fichier où est implémenté l'objet en question.
La priorité	Ordre dans lequel les fichiers JavaScripts seront inclus dans les pages HTML Egalement utilisé pour définir la vue positionné par défaut lors de la création d'un dataset (la valeur minimale est positionnée par défaut)

Une fois créé dans l'interface d'administration, il faudra placer à l'endroit spécifié le fichier JavaScript qui décrira l'objet afin que l'utilisateur puisse visualiser ses données.

Chaque « dataset view » doit implémenter un ensemble de fonction pour pouvoir être utilisé dans SITools2.

4.2.2 Paramétrage du jeu de données

Lors de la définition d'un jeu de données, il faut définir quelle « dataset view » sera utilisé pour afficher les données.

4.2.3 Services sur les « dataset views »

Chaque « dataset view » doit implémenter ses propres évènements (création de menus contextuels, boutons, double click). Si l'on peut s'inspirer des vues existantes, chaque vue doit être autonome, et tous les objets qu'elle utilise doivent être décrit dans le fichier JavaScript.

Chaque « dataset view » doit également implémenter une méthode statique getParameters :

```
/**
 * @static
 * Implementation of the method getParameters to be able to load view Config panel.
 * @return {Array} the parameters to display into administration view.
 */
sitools.user.component.dataviews.livegrid.LiveGrid.getParameters = function () {
	return [{
		jsObj : "Ext.slider.SingleSlider",
		config : {
			minValue : 20,
			maxValue : 250,
			increment : 5,
			width : 400,
			value : 25,
			fieldLabel : i18n.get("label.lineHeight"),
			plugins: [new Ext.ux.plugins.SliderRange(), new Ext.slider.Tip()],
			parameterName : "lineHeight"
		}
	}];
};
```

Cette méthode doit retourner un tableau de paramètres.

Chaque paramètre est un objet décrit ainsi :

Attribut	Description
jsObj	Nom de la classe à implémenter lorsque l'administrateur configurera la vue
Config	La configuration initiale qui sera transmise lors de la création de l'objet

4.3 Gestion et développement d'un module

Un module est un composant JavaScript permettant de créer une application configurable par l'administrateur dans l'IHM cliente.

4.3.1 Mise en place d'un module par l'interface d'administration

En tant qu'administrateur, on peut ajouter de nouveaux modules (ex : dataset-explorer) développés par les développeurs.

Un module est défini par :

Paramètre	Description
Nom	Le nom du module
Description	La description du module
Label	Label internationalisé par défaut (peut être surchargé dans la configuration du projet)
Auteur	L'auteur du module
Version	La version du module
Title	Il est possible d'internationaliser le titre des modules en rajoutant une entrée dans le dictionnaire.
Width	la largeur par défaut de la fenêtre associée au module
Height	la hauteur par défaut de la fenêtre associée au module
Icon Class	Une classe CSS permettant notamment de définir l'icône du module.
X	l'abscisse par défaut de la fenêtre associée au module
Y	l'ordonnée par défaut de la fenêtre associée au module
Xtype	Le xtype détermine l'objet javascript qui sera chargé lors de l'ouverture de la fenêtre associée
Priority	L'ordre dans lequel les fichiers doivent être inclus (en premier les petites priorité)
Des dépendances	Une liste d'url de fichiers javascripts ou css. Ces fichiers seront chargés dynamiquement si le module est rattaché à un projet. Il faudra notamment qu'un des javascripts définisse un objet dont le xtype est spécifié dans le module

4.3.2 Paramétrage du projet

Lors de la définition d'un projet, l'administrateur peut rattacher les modules.

Pour chaque module, il peut :

- Rattacher ou non un module au projet

- Déterminer l'ordre dans le menu de navigation,
- Définir des catégories ou des emplacements réservés :
    - Catégorie : le module sera présent dans l'IHM sous le menu prinicpal portant le label de la catégorie. Ceci permet de regrouper plusieurs modules dans un seul menu.
    - Emplacements réservés : Définir un id de div présent dans le template HTML rattaché au projet permet d'ouvrir la représentation du module à l'intérieur de la div au chargement de SITools2. Le module ne sera pas présent au niveau du menu.
- Définir des propriétés spécifiques au module : chaque module peut spécifier un certain paramétrage à renseigner par l'administrateur.

4.3.3 Développement d'un nouveau module

Chaque module doit donc spécifier un xtype, qui correspond à un objet JavaScript. Cet objet JavaScript doit étendre de l'objet Ext.Panel. Les dépendances sont inclues dans l'ordre déterminé par l'administrateur lors de la définition du module.

4.3.4 Paramétrages des modules

Chaque module peut également implémenter une méthode statique getParameters afin de donner à l'administrateur la possibilité de paramétrer le module spécifiquement pour un projet donné :

```
/**
 * @static
 * Implementation of the method getParameters to be able to load view Config
Module panel.
 * @return {Array} the parameters to display into administration view.
 */
sitools.user.modules.datastorageExplorer.getParameters = function () {

    return [{
        jsObj : "Ext.form.TextField",
        config : {
            fieldLabel : i18n.get("label.urlDatastorage"),
            allowBlank : false,
            width : 200,
            listeners: {
                render: function(c) {
                  Ext.QuickTips.register({
                    target: c,
                    text: "the datastorage url (cf. Storage)"
                  });
                }
            },
            name : "dynamicUrlDatastorage",
            value : undefined
        }
    },
    {
        jsObj : "Ext.form.TextField",
        config : {
            fieldLabel : i18n.get("label.nameDatastorage"),
            allowBlank : false,
            width : 200,
            listeners: {
                render: function(c) {
                  Ext.QuickTips.register({
                    target: c,
                    text: "the label NAME of the datastorage to display (cf.
Storage)"
                  });
                }
            },
            name : "nameDatastorage",
            value : undefined
        }
    }];
};
```

Cette méthode doit retourner un tableau de paramètres.

Chaque paramètre est un objet décrit ainsi :

Attribut	Description
jsObj	Nom de la classe à implémenter lorsque l'administrateur configurera le module
Config	La configuration initiale qui sera transmise lors de la création de l'objet

4.4 Développement de services IHM sur un dataset

Un service IHM ou GUIService, est une fonctionnalité disponible sur une vue de jeu de données comme un tri, un flitre ou l'affichage d'un graph. Cette fonctionnalité peut avoir accès à la description du jeu de données, aux données affichées ainsi qu'aux critères de recherche.

4.4.1.1 Au niveau de l'IHM administrateur

En tant qu'administrateur, on peut ajouter de nouveaux services IHM réalisés par les développeurs.

Un service IHM est défini par :

Paramètre	Description
Nom	Le nom du service IHM
Description	La description du service IHM
Label	Label internationalisé par défaut (peut être surchargé dans la configuration du service IHM sur un dataset)
Auteur	L'auteur du service IHM
Version	La version du service IHM
Icon	L'url d'une image pour définir l'icône du service IHM. (peut être surchargé dans la configuration du service IHM sur un dataset). La taille de l'image conseillée est 16*16 pixels
Xtype	Le xtype détermine l'objet javascript qui sera chargé lors de l'appel du service
Priority	L'ordre dans lequel les fichiers doivent être inclus (en premier les petites priorités)
Des dépendances	Une liste d'url de fichiers javascripts ou css. Ces fichiers seront chargés dynamiquement. Il faudra notamment qu'un des javascripts définisse un objet dont le xtype est spécifié dans le service IHM

4.4.2 Paramétrage des services sur un dataset

Suite à la définition d'un dataset, l'administrateur peut configurer des services IHM et serveur.

Pour chaque service, il peut :

- Déterminer l'ordre dans le menu de la vue de jeu de données,
- Définir des catégories: le service sera présent dans l'IHM sous le menu de la vue de jeu de données portant le label de la catégorie. Ceci permet de regrouper plusieurs services dans un seul menu.
- Définir des propriétés spécifiques au service : chaque service peut spécifier un certain paramétrage à renseigner par l'administrateur.
- Définir le label et la visibilité du service.

4.4.3 Développement d'un nouveau service IHM

Chaque module doit donc spécifier un xtype, qui correspond à un objet JavaScript. Cet objet JavaScript est responsable de son affichage et peut se présenter sous la forme d'une fenêtre.

Un module doit également implémenter la fonction « executeAsService », par exemple :

```
/**
 * @static
 * Implementation of the method executeAsService to be able to launch this window as a
service.
 * @param {Object} config contains all the service configuration
 */sitools.user.component.dataviews.services.addSelectionService.executeAsService =
function (config) {
    var selections = config.dataview.getSelections();
    var rec = selections[0];
    if (Ext.isEmpty(rec)) {
        Ext.Msg.alert(i18n.get('label.warning'),
i18n.get('warning.noRecordsSelected'));
        return;
    }
    var addSelectionService = new
sitools.user.component.dataviews.services.addSelectionService(config);
    addSelectionService.show();
};
```

Les dépendances sont inclues dans l'ordre déterminé par l'administrateur lors de la définition du module.

Le paramètre config contient tous les paramètres du service ainsi que les objets suivants :

Object	Description
columnModel	L'objet columModel de la vue de jeu de données
store	L'objet store de la vue de vue de jeu de données
dataview	L'objet vue de jeu de données
origin	Le nom de vue de jeu de données

4.4.4 Paramétrages des services IHM

Chaque service IHM peut également implémenter une méthode statique getParameters afin de donner à l'administrateur la possibilité de paramétrer le service IHM spécifiquement pour un dataset donnée ;

```
/**
 * @static
 * Implementation of the method getParameters to be able to load view Config
service panel.
 * @return {Array} the parameters to display into administration view.
 */
sitools.user.component.dataviews.services.addSelectionService.getParameters =
function () {

    return [{
        jsObj : "Ext.form.TextField",
        config : {
            fieldLabel : i18n.get("label.urlDatastorage"),
            allowBlank : false,
            width : 200,
            listeners: {
                render: function(c) {
                  Ext.QuickTips.register({
                    target: c,
                    text: "the datastorage url (cf. Storage)"
                  });
                }
            },
            name : "dynamicUrlDatastorage",
            value : undefined
        }
    },
    {
        jsObj : "Ext.form.TextField",
        config : {
            fieldLabel : i18n.get("label.nameDatastorage"),
            allowBlank : false,
            width : 200,
            listeners: {
                render: function(c) {
                  Ext.QuickTips.register({
                    target: c,
                    text: "the label NAME of the datastorage to display (cf.
Storage)"
                  });
                }
            },
            name : "nameDatastorage",
            value : undefined
        }
    }];
};
```

Cette méthode doit retourner un tableau de paramètres.

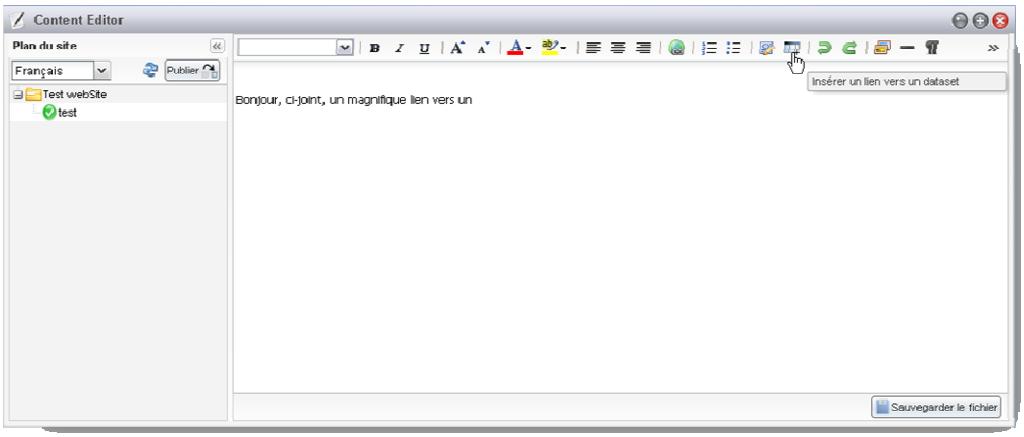
Chaque paramètre est un objet décrit ainsi :

Attribut	Description
jsObj	Nom de la classe à implémenter lorsque l'administrateur configurera le module
Config	La configuration initiale qui sera transmise lors de la création de l'objet

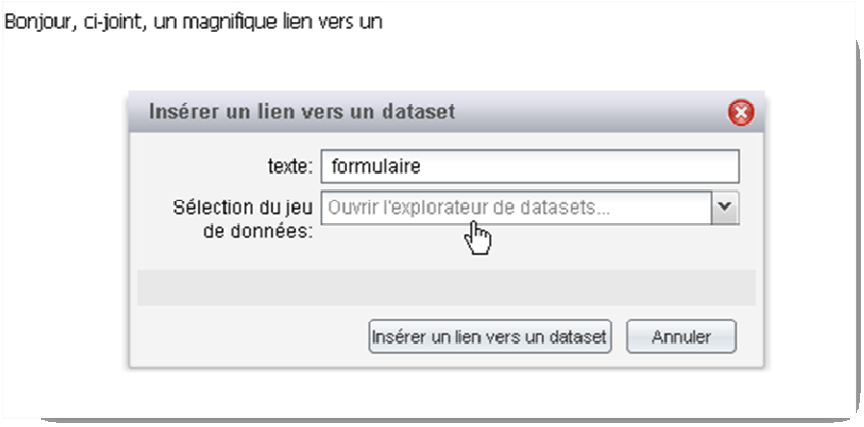
4.5 Tutoriels

4.5.1 Comment insérer un lien vers un dataset ou un formulaire depuis l'éditeur de contenu ?

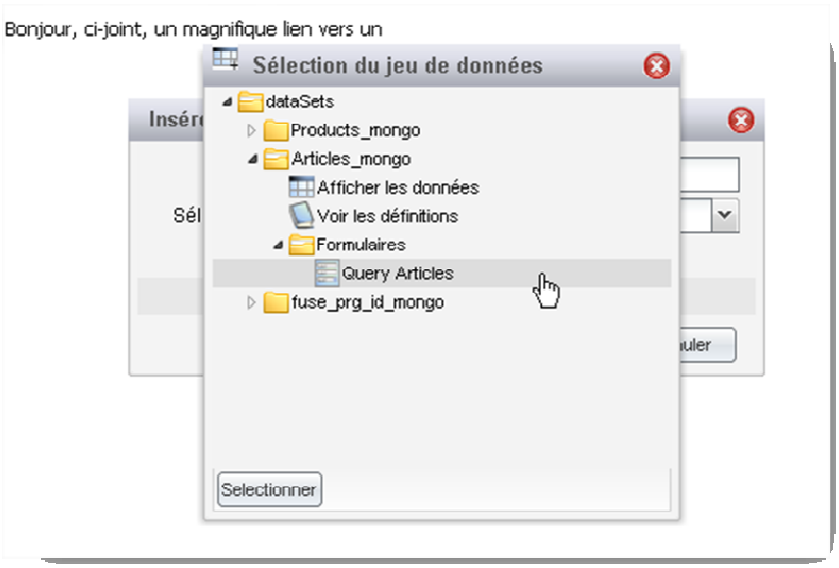
1. **Une fois le module Editeur de Contenu ouvert (Content Editor), cliquez sur le bouton ci-dessus.**

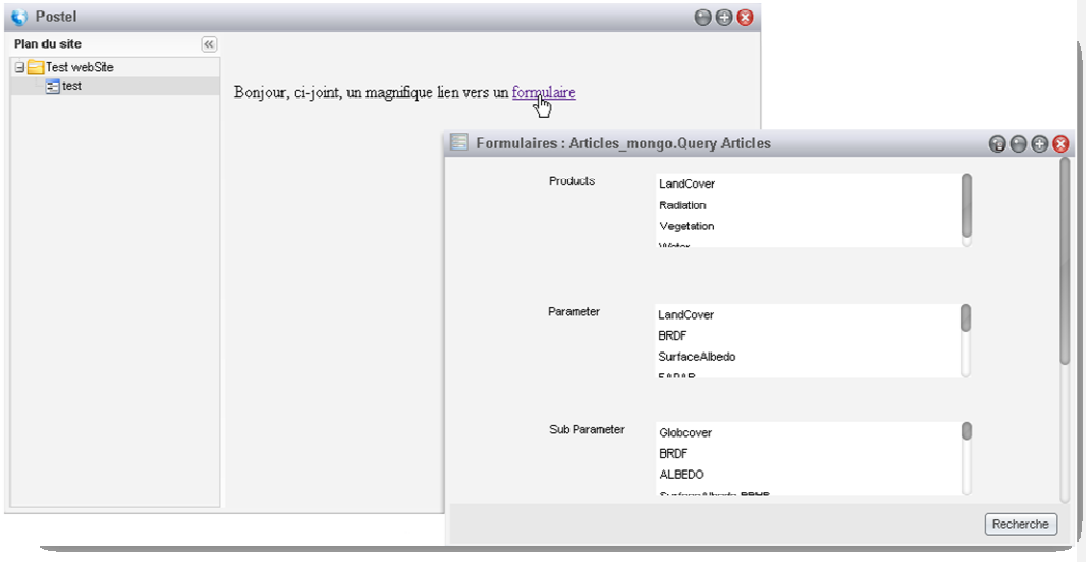


2. **Entrez le texte à rendre cliquable dans la zone *texte* puis cliquez sur *ouvrir l'explorateur de datasets...***

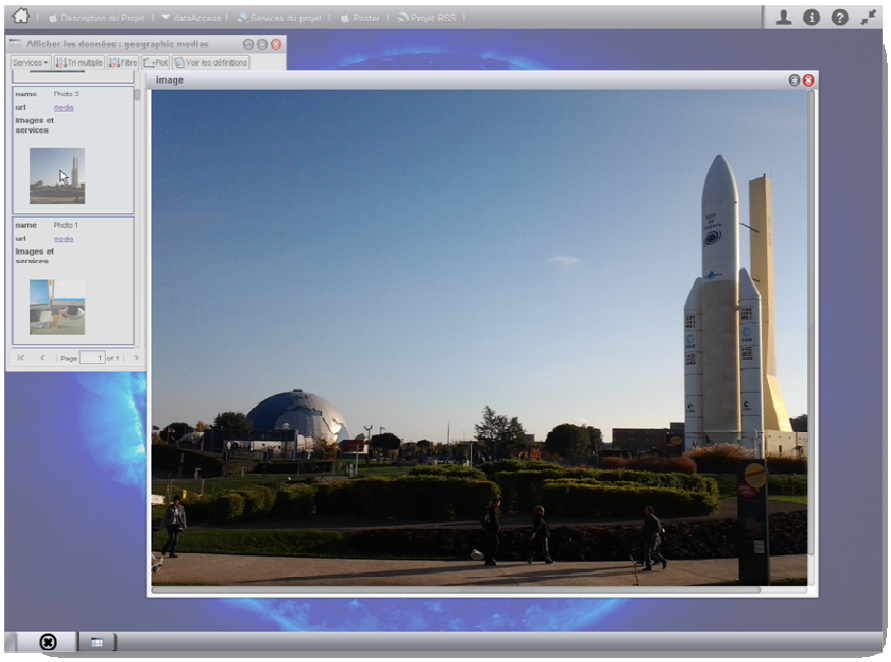


3. **Sélectionnez soit un jeu de données soit un formulaire.**





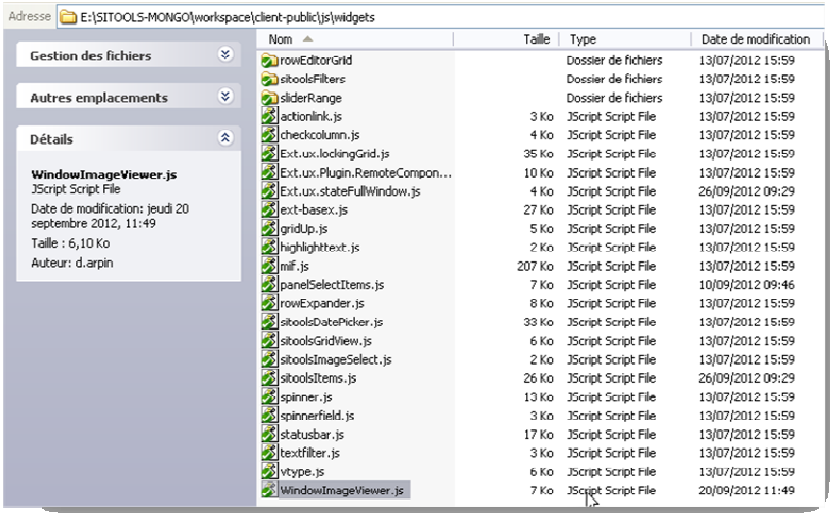
4.5.2 Comment modifier le comportement de la fenêtre preview ?



- Par défaut, la *fenêtre image preview* s'ouvre en mode modal, c'est-à-dire qu'aucune action ne sera possible sur d'autres éléments du bureau (fond légèrement grisé). Il existe un moyen pour modifier ce comportement, pou par exemple, pouvoir comparer deux images côte à côte.

1. **Rendez-vous dans le répertoire d'installation de SITools puis allez dans :**

{$root}/workspace/client-public/js/widgets/ **et ouvrez le fichier** ***WindowImageViewer.js***



1. **Commentez la ligne** ***modal : true***

```
sitools.widget.WindowImageViewer = Ext.extend(Ext.Window, {
    resizeImage : false,
    maximizable : true,
    modal : true,
    minHeight : 0,
    minWidth : 0,
```

```
sitools.widget.WindowImageViewer = Ext.extend(Ext.Window, {
    resizeImage : false,
    maximizable : true,
//    modal : true,
    minHeight : 0,
    minWidth : 0,
```

4.5.3 Comment modifier le comportement d'une fenêtre en général ?

Nous savons que sitools possède deux modes de navigation : Desktop (fenêtre volante) et Fixed (fenêtre inscrustée).

Chacun de ces modes est associé à un objet javaScript qui est chargé d'afficher les composants en fonction du type de navigation. De plus, ces objets javaScript (à savoir sitools.user.desktop.navProfile.desktop et sitools.user.desktop.navProfile.fixed) peuvent déléguer l'ouverture du composant à lui-même.

C'est-à-dire que si le composant en passe d'être ouvert implémente les méthodes ***showMeInDesktopNav ()*** et ***showMeInFixedNav ()***, ces dernières seront exécutées.

fixed.js

```
/*************************************
/*global Ext, sitools, ID, i18n, document, showResponse, alertFailure,
Ext.namespace('sitools.user.desktop.navProfile');

/**
sitools.user.desktop.navProfile.fixed = {
    /**
    context : "fixed",
    /**
    createComponent : function (config) {
        var component = new config.JsObj(config.componentCfg);
        var windowSettings = config.windowSettings;

        //déléguer au composant l'ouverture
        if (Ext.isFunction(component.showMeInFixedNav)) {
            component.showMeInFixedNav(component, config);
            return;
        }
```

desktop.js

```
/*************************************
/*global Ext, sitools, ID, i18n, document, showResponse, alertFailure,
Ext.namespace('sitools.user.desktop.navProfile');

/**
sitools.user.desktop.navProfile.desktop = {
    /**
    context : "desktop",
    /**
    createComponent : function (config) {
        var desktop = getDesktop();
        var componentCfg  = config.componentCfg;
        var component = new config.JsObj(componentCfg);
        var windowSettings = config.windowSettings;
        //déléguer au composant l'ouverture
        if (Ext.isFunction(component.showMeInDesktopNav)) {
            component.showMeInDesktopNav(component, config);
            return;
        }
```

Prenons un exemple concret :

Le composant ***sitools.widget.feedItemDetails*** est un objet contenant le détail d'une actualité (flux RSS). On veut que ce composant s'ouvre toujours de façon modale, c'est pourquoi on va personnaliser les méthodes showMeInDesktopNav() et showMeInFixedNav() :

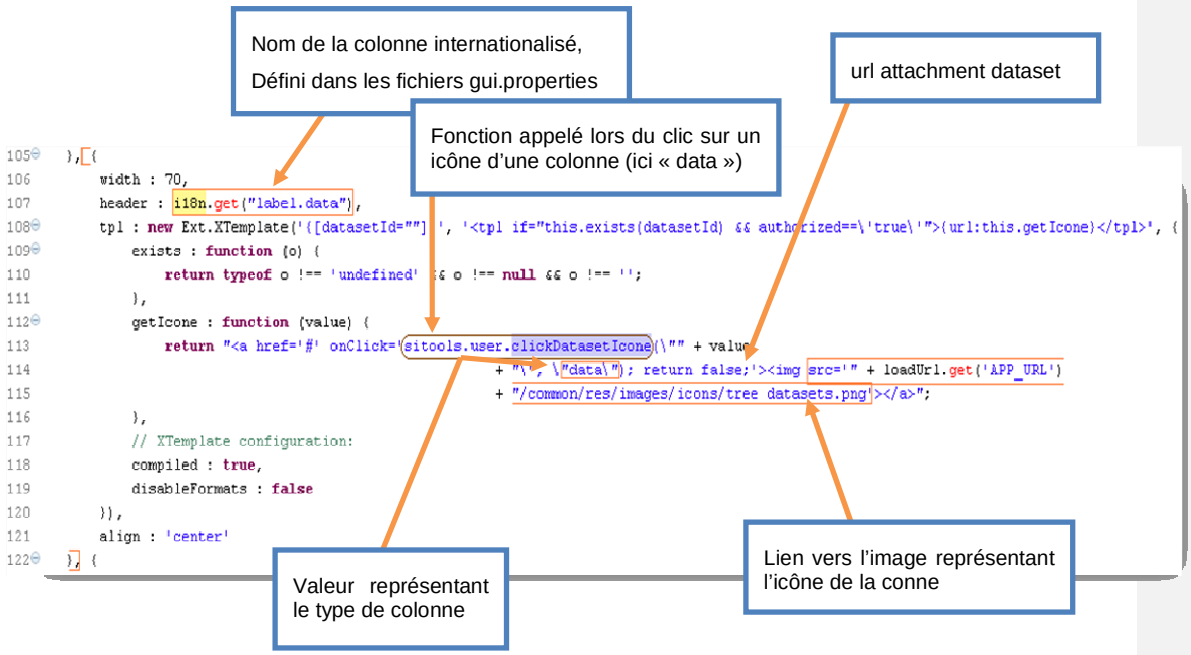
```
/**
showMeInFixedNav : function (me, config) {
    Ext.apply(config.windowSettings, {
        width : config.windowSettings.winWidth || DEFAULT_WIN_WIDTH,
        height : config.windowSettings.winHeight || DEFAULT_WIN_HEIGHT
    });
    SitoolsDesk.openModalWindow(me, config);
},
/**
showMeInDesktopNav : function (me, config) {
    Ext.apply(config.windowSettings, {
        width : config.windowSettings.winWidth || DEFAULT_WIN_WIDTH,
        height : config.windowSettings.winHeight || DEFAULT_WIN_HEIGHT
    });
    SitoolsDesk.openModalWindow(me, config);
}
```

Les paramètres passés sont le composant lui-même (me) et les options de configuration de ce composant (config). Ici, on va utiliser la hauteur et la largeur de la fenêtre définies dans config.windowSettings **si elles existent**, sinon on utilisera la hauteur et la largeur par défaut. Ensuite, on appelle la fonction openModalWindow qui s'occupe d'ouvrir une fenêtre modale avec la configuration de fenêtre passée en paramètre.

4.5.4 Comment ajouter une colonne au project graph ?

1. **Rendez-vous dans le répertoire d'installation de sitools puis allez dans : *{$root}/workspace/client-user/js/modules/projectGraph* et ouvrez le fichier *projectGraph.js***

2. **Les colonnes sont définis dans la fonction *initComponent ()* par la variable *this.columns*, Example de colonne :**




```
105  },{
106      width : 70,
107      header : i18n.get("label.data"),
108      tpl : new Ext.XTemplate('{[datasetId=""] ', '<tpl if="this.exists(datasetId) && authorized==\'true\'">{url:this.getIcone}</tpl>', {
109          exists : function (o) {
110              return typeof o !== 'undefined' && o !== null && o !== '';
111          },
112          getIcone : function (value) {
113              return "<a href='#' onClick='sitools.user.clickDatasetIcone(\"" + value
114                                          + "\", \"data\"); return false;'><img src='" + loadUrl.get('APP_URL')
115                                          + "/common/res/images/icons/tree_datasets.png'></a>";
116          },
117          // XTemplate configuration:
118          compiled : true,
119          disableFormats : false
120      }),
121      align : 'center'
122  }, {
```

3. **Pour rajouter une colonne, il vous faut copier l'exemple de colonne ci-dessous et le rajouter à la fin de la variable this.columns (qui est un tableau) :**

```
}, {
    width : 70,
    header : i18n.get("label.myNewColumn"),
    tpl : new Ext.XTemplate('{[dataset.Id=""]}', '<tpl if="this.exists(datasetId) && authorized==\'true\'">{url:this.getIcone}</tpl>', {
        exists : function (o) {
            return typeof o !== 'undefined' && o !== null && o !== '';
        },
        getIcone : function (value) {
            return "<a href='#' onClick='sitools.user.clickDatasetIcone(\"" + value
                                                + "\", \"newColumn\"); return false;'><img src='" + loadUrl.get('APP_URL')
                                                + "/common/res/images/icons/newColumnIcon.png'></a>";
        },
        // XTemplate configuration:
        compiled : true,
        disableFormats : false
    }),
    align : 'center'
}];
```

Fin du tableau this.columns

Ici, le nom de notre nouvelle colonne sera *label.myNewColumn*, le type de colonne sera *newColumn* et l'icône de la colonne sera */newColumnIcon.png*.

1. **Une fois notre nouvelle colonne ajoutée, il faut aller définir notre nouveau type (en l'occurrence *newColumn*) dans la fonction *clickDatasetIcone*. Cette fonction est définie dans le fichier *{$root}/workspace/client-user/js/def.js*.**

```
sitools.user.clickDatasetIcone = function (url, type, extraCmpConfig) {
    Ext.Ajax.request({
        method : "GET",
        url : url,
        success : function (ret) {
            var Json = Ext.decode(ret.responseText);
            if (showResponse(ret)) {
                var dataset = Json.dataset;
                var componentCfg, javascriptObject;
                var windowConfig = {
                    datasetName : dataset.name,
                    type : type,
                    saveToolbar : true,
                    toolbarItems : []
                };
                switch (type) {
                case "desc" :
                    Ext.apply(windowConfig, {
                        title : i18n.get('label.description') + " : " + dataset.name,
                        id : "desc" + dataset.id,
                        saveTcolbar : false,
                        iconCls : "version"
                    });

                    componentCfg = {
                        autoScroll : true,
                        html : dataset.descriptionHTML
                    };
                    Ext.applyIf(componentCfg, extraCmpConfig);
                    javascriptObject = Ext.Panel;
                    SitoolsDesk.addDesktopWindow(windowConfig, componentCfg, javascriptObject);

                    break;
```

```
821                 switch (type) {
822                 case "desc" :
823                     Ext.apply(windowConfig, {
824                         title : i18n.get('label.description') + " : " + dataset.name,
825                         id : "desc" + dataset.id,
826                         saveToolbar : false,
827                         iconCls : "version"
828                     });
829
830                     componentCfg = {
831                         autoScroll : true,
832                         html : dataset.descriptionHTML
833                     };
834                     Ext.applyIf(componentCfg, extraCmpConfig);
835                     javascriptObject = Ext.Panel;
836                     SitoolsDesk.addDesktopWindow(windowConfig, componentCfg, javascriptObject);
837
838                     break;
839                 case "newColumn" :
840                     // Code Spécifique à la nouvelle colonne
841
842                     break;
```

On vient de rajouter le « cas » ou la valeur de type serait ***newColumn***, il ne reste plus qu'à coder le traitement spécifique à effectuer.

4.5.5 Modifier la taille allouée aux formulaires dans la vue composite

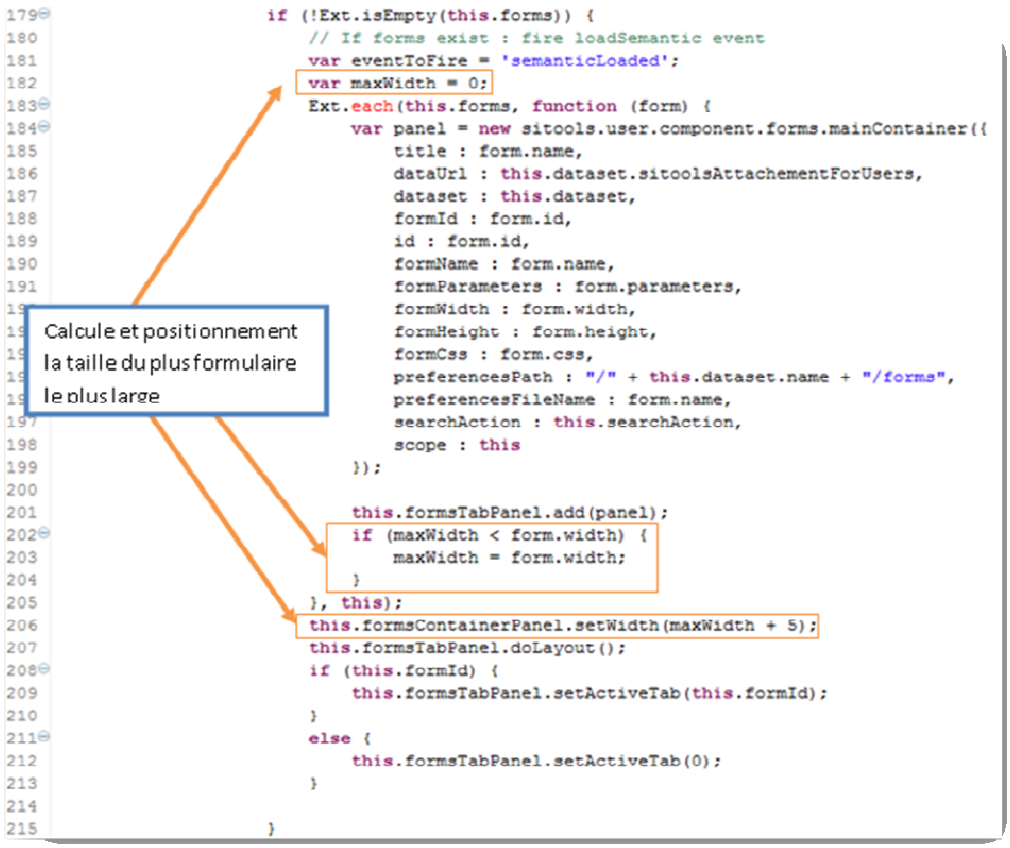
Le but de se tutoriel est de comprendre comment on peut modifier la taille allouée aux formulaires dans la vue composite. Cette vue présente à la fois les formulaires et les données provenant de la recherche dans un même écran.

La zone dans laquelle s'affichent les formulaires a une taille par défaut, il est possible de l'agrandir avec la souris mais ça peut ne pas être satisfaisant dans certains cas.

L'idée est de positionner la barre verticale pour pouvoir afficher entièrement le plus large des formulaires. On doit calculer la largeur maximale des formulaires et la positionner sur le panel correspondant.

1. **Rendez-vous dans le répertoire d'installation de sitools puis allez dans : *{$root}/workspace/client-user/js/component/datasetOverview* et ouvrez le fichier *datasetOrverview.js***

2. **Effectuez les modifications suivantes :**




```
179            if (!Ext.isEmpty(this.forms)) {
180                // If forms exist : fire loadSemantic event
181                var eventToFire = 'semanticLoaded';
182                var maxWidth = 0;
183                Ext.each(this.forms, function (form) {
184                    var panel = new sitools.user.component.forms.mainContainer({
185                        title : form.name,
186                        dataUrl : this.dataset.sitoolsAttachementForUsers,
187                        dataset : this.dataset,
188                        formId : form.id,
189                        id : form.id,
190                        formName : form.name,
191                        formParameters : form.parameters,
                            formWidth : form.width,
                            formHeight : form.height,
                            formCss : form.css,
                            preferencesPath : "/" + this.dataset.name + "/forms",
                            preferencesFileName : form.name,
197                         searchAction : this.searchAction,
198                         scope : this
199                     });
200
201                     this.formsTabPanel.add(panel);
202                     if (maxWidth < form.width) {
203                         maxWidth = form.width;
204                     }
205                 }, this);
206                 this.formsContainerPanel.setWidth(maxWidth + 5);
207                 this.formsTabPanel.doLayout();
208                 if (this.formId) {
209                     this.formsTabPanel.setActiveTab(this.formId);
210                 }
211                 else {
212                     this.formsTabPanel.setActiveTab(0);
213                 }
214
215             }
```

On ajoute 5 pixel à la taille maximale pour être sur qu'il n'y ait pas de barre de défilement qui s'affiche.

Il faut ensuite forcer un réaffichage de la fenêtre pour la taille soit prise en compte.

```
111             descriptionLoaded : function () {
112 //              this.unMask();
113                 this.doLayout();
114                 return;
115             }
```

5 Packager un développement

Cette partie a pour but d'expliquer l'architecture du projet « sitools-install-izpack » dans le cas où un développeur voudrait rajouter une fonctionnalité. Cette partie n'explique en revanche pas comment ajouter une fonctionnalité, pour cela se référer à la documentation de IzPack : http://izpack.org/documentation/ ou http://docs.codehaus.org/display/IZPACK/User+documentation

5.1 Présentation du projet Sitools-install-izpack

Sitools2 peut être installé grâce à un installeur généré à l'aide d'IzPack version 4.3.4. Le projet sitools-install-izpack contient l'ensemble des fichiers nécessaires à la génération de cet installeur.

Ce projet contient un ensemble de dossier :

- src : Sources des classes Java nécessaire à la vérification de la connectivité de la base de données et à la création des bases de données
- bin : Jars des classes contenues dans src
- conf : Les scripts ant de compilation des classes contenu dans src
- install-res : Fichiers de langues et de formulaires nécessaires à l'installeur.
- lib : Librairie IzPack ainsi que les drivers JDBC sous forme de jar
- res : Ensemble des ressources nécessaires pour la génération IzPack
    - data : Dossiers et fichiers copié lors de l'installation dans le répertoire data
    - database : Scripts d'installation des bases de données pour PostgreSQL et MySQL
    - ext : Jar de génération du classpath
    - img : Images utiles à l'installeur
    - properties : Fichier de « properties » de Sitools généré lors de la génération de l'installeur
    - script : Scripts de démarrage de Sitools pour Linux et Windows. Certains scripts, startSitools.bat et startSitools.sh, sont générés lors de la génération de l'installeur

Certains fichiers présents dans le projet sont également importants :

- install.xml : Il s'agit du fichier principal qui décrit l'installeur IzPack
- build.xml : Il s'agit du script ant permettant la génération de l'installeur.

5.2 Etapes de la génération de l'installeur

La génération de l'installeur est effectuée grâce à un script ant. Il y a certaines étapes qui sont effectuées avant la génération effective de l'installeur :

Pré requis

Compiler le projet fr.cnes.sitools.core et le projet fr.cnes.sitools.extensions et mettre à jour la propriété ROOT_DIRECTORY_LOCAL dans le fichier build.properties (cette propriété contient le chemin du dossier qui contient le workspace courant).

Compilation des sources du projet IzPack

Le projet IzPack contient des classes Java qui permettent de vérifier le bon paramétrage de la base de données ainsi que la création des tables de Sitools2 et de tests. Cette étape compile ces sources et créer 2 jar. Un pour la validation de la connexion et un autre pour la création des tables.

Génération des scripts

Les scripts de lancement de Sitools2 sont générés de manière à ce qu'ils puissent être modifié lors de l'installation pour s'adapté au contexte d'installation. En bref, certaines variables sont positionnées dans les scripts pour pouvoir être remplacées lors de l'installation.

Génération du fichier de properties

Le fichier de properties de Sitools2 est généré en positionnant des variables qui seront modifiées lors de l'installation.

Génération de l'installeur

L'installeur est généré avec une « target ant » contenue dans le jar standalone-compiler.jar (dans le dossier lib)

5.3 Description de l'installeur Izpack de SITools2

Le fichier install.xml contient la description de l'installeur.

Les différents packs (package installable) défini sont les suivants :

- Server : Package obligatoire, ensemble des packages de la partie serveur
    - Sitools : Jar du cœur de Sitools2 et la création du dossier de logs de Sitools2
    - Librairies : Ensemble des librairies utiles à Sitools2

  - Cots-Restlet : Framework Restlet
  - Scripts : Ensemble des scripts de démarrages de Sitools2
  - Config : Fichiers de configuration de Sitools2
  - Data : Données de base de Sitools2
  - Database : Scripts d'installation des bases de données
- Server Extension : Package facultatif, ensemble des packages d'extensions du serveur Sitools2
  - Extensions : Jar des extensions du serveur Sitools2
- Client : Package facultatif, ensemble des packages de la partie cliente de Sitools2
  - Ext-js : Librairies Ext-js, installé si l'un des packages suivant est demandé
  - Client-public : Ressources clientes communes à l'interface admin et user, installé si l'un des packages suivant est demandé
  - Client-user : Ressources clientes de l'interface utilisateur
  - Client-admin : Ressources clientes de l'interface administrateur

5.4 Ajustements aux besoins de SITools2

2 besoins ont demandés l'ajout de fonctionnalités à l'installeur :

- La vérification de la connexion à la base de données.
- La création des tables des bases de données

Vérification de la connexion

Cette vérification est effectuée par un « Validator » IzPack. Il s'agit d'une classe Java qui permet de valider une étape de l'installation, si la validation passe, l'installation passe à l'étape d'après sinon une erreur est affichée. Il y a également un mécanisme d'avertissement. Il a donc fallu implémenter une classe Java (fr.cnes.sitools.izpack.validator.JDBCConnectionValidator), cette classe implémente l'interface « DataValidator » et vérifie que les paramètres de base de données saisie sont corrects et que la base de données est accessibles. Dans le cas où le type de base de données est PostgreSQL, on vérifie également que le schéma n'existe pas, si c'est le cas, on affiche un avertissement car la création des tables risque d'échouer.

Création des tables utilisateurs

Pour la création des tables utilisateurs, il a fallu créer une nouvelle étape lors de l'installation.

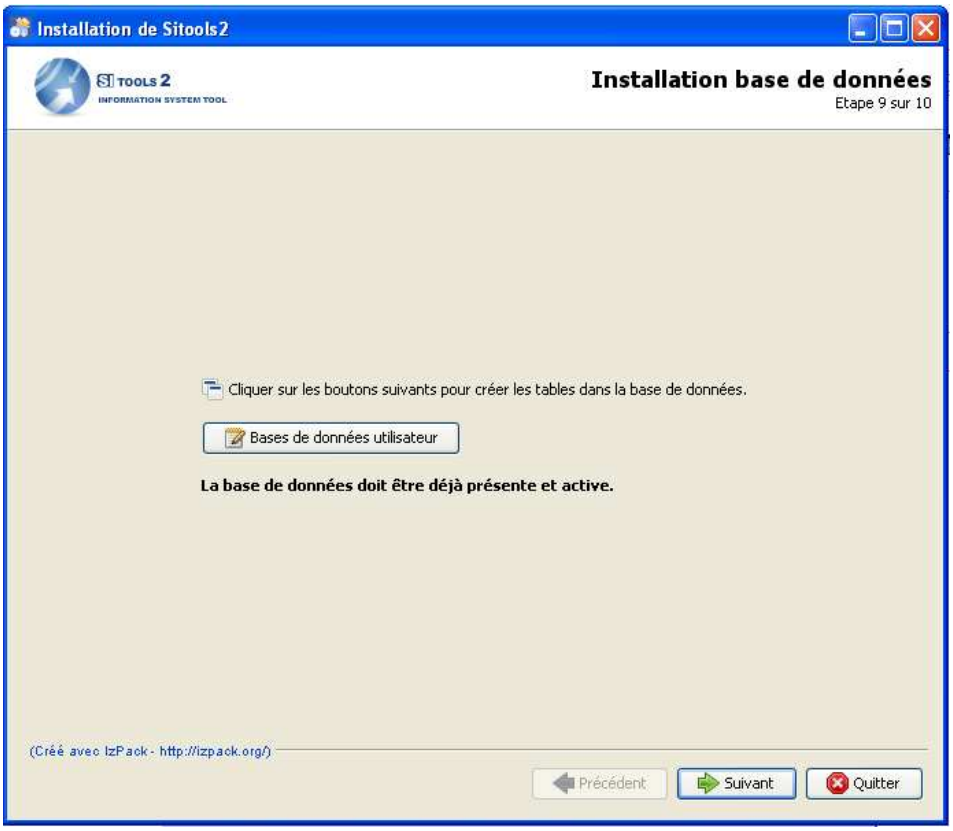


Figure 29 : Etape d'installation des tables de base de données

Pour ajouter une étape personnalisée, il faut créer un nouveau Panel. Ce Panel est une classe Java (fr.cnes.sitools.izpack.panel.DatabasePanel) qui étend la classe IzPanel.

6 Divers

6.1 Notification des arrêts planifiés

Pour notifier des arrêts planifiés, on utilise la partie flux RSS (feeds) de Sitools2. On peut donc spécifier des arrêts planifiés sur un jeu de données ou sur un projet.

La procédure est donc la suivante :

Ajout d'un flux RSS dans l'interface administrateur :

- Cliquer sur le menu « Project Feeds »
- Choisir son projet
- Cliquer sur « create »
- Remplir les informations du flux RSS
    - Titre (doit être simple, sans accent ni espace car il est utilisé dans l'URL du flux, dans le cas ou celui-ci est modifié ultérieurement, c'est le titre utilisé lors de la création qui est utilisé)
    - Description
    - Un lien
    - Un auteur et son email
    - Un type de flux (RSS2 ou ATOM)
- Ajouter un « item » spécifiant de l'arrêt planifié dans l'onglet « Feed items »
    - Cliquer sur « Add »
    - Remplir les informations de l'arrêt planifié
        - Titre
        - Description
        - Un lien
        - Un auteur et son email
        - Une date de publication
        - Une date de mise à jour
    - Cliquer sur OK
- Cliquer sur OK

Le flux est maintenant ajouté, on peut désormais le lire dans l'interface utilisateur au niveau du portail et dans le bureau du projet.

On peut également vouloir mettre un lien vers ce flux dans la page de statut de service (page affichée dans le cas ou la ressource demandée n'existe pas, c'est-à-dire dans notre cas si le projet est arrêté)

Pour ce faire, il faut modifier le fichier « statut.ftl » qui se trouve dans le dossier « fr.cnes.sitools.core/conf/resources/templates »

Il faut ensuite ajouter un lien HTML vers le flux RSS. L'url du flux RSS est :

<nom du host Sitools2> :<port>/sitools/projects/<nom du projet>/clientFeeds/<titre du flux>

6.2 HTML5 dans un module

Il est possible d'utiliser HTML5 dans un module de Sitools2 (sous réserve de compatibilité du navigateur utilisé avec HTML5).

On doit donc construire un Panel et y ajouter le contenu HTML5 dans sa propriété html. Par exemple pour ajouter la lecture d'un fichier son et vidéo :

```
this.normalPanel = new Ext.Panel({
        title : "Standard Panel",
        html : "<audio controls='controls'>" +
                "<source src='url_audio' type='audio/wav' />" +
                "Your browser does not support the audio tag." +
                "</audio>" +
                "<video controls='controls'>" +
                "<source src='url_video' />" +
                "Your browser does not support the video tag." +
                "</video>"
});
```

On peut également référencer directement un fichier html pour éviter d'avoir tout le code HTML5 dans les fichiers JavaScript :

```
this.normalPanel = new Ext.Panel({
        title : "Standard Panel",
        autoLoad : "url_fichier_html"
});
```

6.3 Analyser les LOG d'accès

Analog est un outil d'analyse de log, il peut être utilisé dans le cadre de Sitools2. Une ressource est configuré par défaut sur l'application « AdministratorApplication », elle doit être configuré en fonction de l'environnement de la machine sur laquelle est installé Sitools2.

Le rapport de log est désormais stocké dans le dossier « analog » du dossier « data ». L'application « AdministratorApplication » offre un accès à ce dossier via un « Directory ». Ce « Directory » est configuré avec l'option « no-cache » ce qui permet au rapport de log affiché d'être toujours à jour. Ce dossier est configurable dans « sitools.properties » (paramètre Starter.ANALOG_REPORTS_DIR) mais il est préférable de ne pas le changer, il faudrait ensuite changer la configuration de la ressource.

Sous Windows

Par soucis de cohérence, Analog n'est pas distribué avec Sitools (comme pour la version Linux). Il faut donc le télécharger sur le site d'Analog (http://www.analog.cx/download.html). Il s'agit d'une archive Zip qu'il suffit d'extraire quelque part sur la machine.

Sous linux

Etant donné le nombre de distribution Linux différente il est impossible de fournir une version d'Analog par distribution. Il faut donc installer Analog sur le serveur puis configurer la ressource.

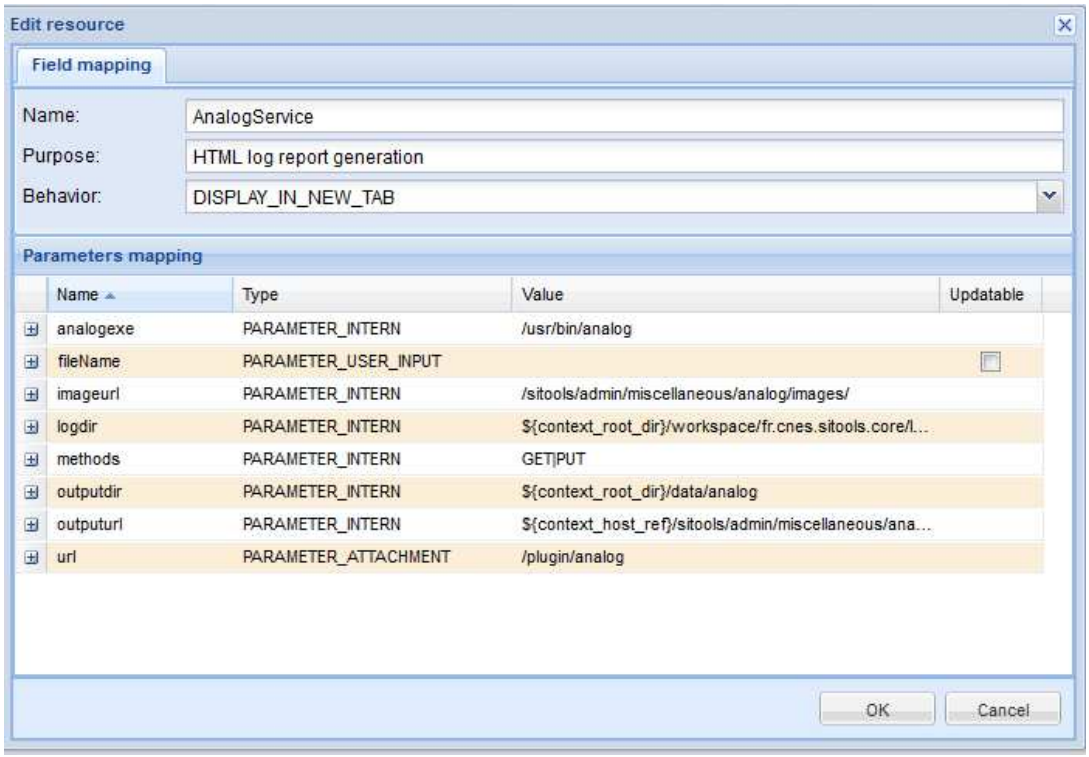
Par exemple sous Ubuntu :

sudo apt-get install analog

Configuration de la ressource

La configuration d'une ressource par défaut est fournie lors de l'installation de Sitools2. Cette configuration peut convenir dans la plupart des cas sur une machine linux.

Dans le cas où la configuration par défaut ne fonctionne pas, il faut modifier le paramètre analogexe. Il permet de spécifier le chemin complet vers l'exécutable d'Analog. (On peut utiliser la variable « ${context_root_dir} » pour pointer directement dans le répertoire racine de l'installation de Sitools2).



Les autres paramètres ne doivent pas être modifiés.

6.4 Optimisation des scripts JavaScript

Pour l'IHM, la version livrée est une version de développement : tous les scripts sont inclus unitairement au niveau du HTML. Il est possible d'optimiser le temps de chargement de l'IHM en utilisant un seul fichier.

Pour ce faire :

- Utiliser le script Ant du projet client-admin build.xml pour générer les fichiers Javascript nécessaires.
- Dans le adminIndex.ftl, commenter toutes les inclusions situées entre les balises <!-- BEGIN_JS_DEV_INCLUDES--> et <!-- END_JS_DEV_INCLUDES -->,
- Décommenter l'une des 2 inclusions après le commentaire A INCLURE POUR LA VERSION DE DEBUG
- Modifier l'inclusion au fichier ext-all-debug.js en ext-all.js

6.5 Démarrage avec données obsolètes

Suite à un changement de version, il est fort probable que les modèles de données aient changés. Dans ce cas, les fichiers XML de données ne sont plus compatibles avec la nouvelle version installée et Sitools2 ne peut pas démarré.

Il existe désormais un mode de démarrage de Sitools2 qui peut permettre de gagner du temps lors d'un changement de version. Ce mode permet de démarrer Sitools2 même si certains champs sont présents dans le XML de données mais plus dans le modèle de données de Sitools2.

Pour utiliser ce mode, il faut démarrer Sitools2 avec l'option –migration :

```
./sitools start -migration
```

Cette option charge toutes les données et réécrit les fichiers XML avec les définitions de modèle de la nouvelle version de Sitools2.

Attention : Les fichiers d'origine ne sont pas conservés et les champs qui n'existent plus sont supprimés. Il est donc conseillé de sauvegarder ses données avant de lancer Sitools2 avec le mode migration.

Agence ou Service : NTSC

Projet : ULISSE/SITOOLS2

6.6 Liste composants SITools2

Titre	Description	Type	Actions permettant l'acces au composant	Comportement figé	Comportement fenêtré	Nom de l'objet JavaScript	Chemin vers l'Objet
DatasetExplorer	Liste les datasets sous forme arborescente	Module	Menu	incrusté zone principale	Fenêtre	sitools.user.modules.datasetExplorer	/workspace/client-user/js/modules/dataSetExplorer/datasetExplorer.js
Project Description	Description du projet	Module	Menu	incrusté zone principale	Fenêtre	sitools.user.modules.projectDescription	/workspace/client-user/js/modules/projectDescription/projectDescription.js
Project Graph	Visualisation du graph	Module	Menu	incrusté zone principale	Fenêtre	sitools.user.modules.projectGraphTree	/workspace/client-user/js/modules/projectGraph/projectGraph.js
Help	Visualisation de l'aide	Module	Menu	Fenêtre modale	Fenêtre	sitools.user.modules.help	/workspace/client-user/js/components/help/help.js
Project Feeds	Visualisation de tous les flux	Module	Menu	incrusté zone principale	Fenêtre	sitools.user.modules.feedsReaderProject	/workspace/client-user/js/modules/feedsReader/feedReaderProject.js
Forms Module	Visualisation de tous les formulaires	Module	Menu	incrusté zone principale	Fenêtre	sitools.user.modules.formsModule	/workspace/client-user/js/modules/FormsModule/formsModule.js
OpenSearch Module	Visualisation de tous les openSearch	Module	Menu	incrusté zone principale	Fenêtre	sitools.user.modules.openSearchModule	/workspace/client-user/js/modules/openSearchModule/openSearchModule.js
FormAsMenu	Visualisation de formulaires depuis un menu	Module	Menu	Menu	Menu	sitools.user.modules.formsAsMenu	/workspace/client-user/js/modules/formsAsMenu/formsAsMenu.js
Poster	Permet de lancer des requetes REST	Module	Menu	incrusté zone principale		sitools.user.modules.extPoster	/workspace/client-user/js/modules/extPoster/extPoster.js
Poster Result	Fenêtre de résultat d'une action du module poster.	Composant	Click Poster Send btn			sitools.user.modules.extPoster.windowResult	/workspace/client-user/js/modules/extPoster/extPoster.js
Mapshup	Visualisation de l'extension MapShup	Module	Menu			sitools.component.html.ModuleMapShup	/workspace/client-user/js/modules/html/ModuleMapShup/htmlModuleJwobrowser.js
User Space -> diskSpace	Explorateur du répertoire de l'utilisateur	Panel	Click surDisk Space de la fenêtre userSpace			sitools.user.component.entete.userProfile.diskSpace	/workspace/client-user/js/components/entete/userProfile/diskSpace.js
User Space -> Tasks	Liste des taches effectuées par l'utilisateur	Panel	Click suruserTask de la fenêtre userSpace			sitools.user.component.entete.userProfile.tasks	/workspace/client-user/js/components/entete/userProfile/tasks.js
contentEditor	Créer/Editer du contenu html	Module	Menu			sitools.user.modules.contentEditorModule	/workspace/client-user/js/modules/cmsModule/contentEditorModule.js
contentViewer	Visualiser du contenu html	Module	Menu			sitools.user.modules.contentViewerModule	/workspace/client-user/js/modules/cmsModule/contentViewerModule.js
DataStorageExplorer	Visualisation de plusieurs datasets (explorateur)	Module	Menu			sitools.user.modules.datastorageExplorer	/workspace/client-user/js/modules/cmsModule/datastorageExplorer.js
ajoutFileExplorer		Composant	Création file	Fenêtre modale		sitools.user.modules.chooseFileInExplorer	/workspace/client-user/js/modules/cmsModule/chooseFileInExplorer.js
AjoutImage		Composant	Création file			ImageExplorer.prototype	/workspace/client-public/js/widgets/htmlEditorAdvanced/src/imageExplorer.js
Dataviews			Dataset Explorer	incrusté zone principale	Fenêtre	sitools.user.modules.datasetExplorer	/workspace/client-user/js/modules/dataSetExplorer/datasetExplorer.js
			Dataset Graph	incrusté zone principale	Fenêtre	sitools.component.graphs.graphsDatasetWin	/workspace/client-admin/js/graphs/graphsDatasetWin.js
	Visualisation des données de datasets	Composant	resultat de formulaire multiDs	incrusté zone principale	Fenêtre	sitools.user.component.forms.overviewResultsProjectForm	/workspace/client-user/js/components/forms/overviewResultsProjectForm.js
			Formulaire	N.A.	Fenêtre	sitools.user.component.formComponentsPanel	/workspace/client-user/js/components/forms/formComponentsPanel.js
			Dataset Link	incrusté zone principale	Fenêtre	sitools.admin.datasets.columnRenderer.datasetLinkPanel	/workspace/client-admin/js/datasets/columnRenderer/datasetLinkPanel.s
Formulaires	Visualisation d'un formulaire	Composant	Modules forms Module Dataset Explorer	Composite incrusté zone principale	Fenêtre formulaire seul	sitools.user.component.forms.mainContainer	/workspace/client-user/js/components/forms/mainContainer.js

Titre	Description	Type	Actions permettant l'acces au composant	Comportement figé	Comportement fenêtré	Nom de l'objet JavaScript	Chemin vers l'Objet
Formulaires Multi DS	Visualisation d'un formulaire multiDS	Composant	ModuleForms	Incrusté zone principale	Fenêtre	sitools.user.component.forms.projectForm	/workspace/client-user/js/components/forms/projectForm.js
Resultat multiDS Form	Affichage du résultat d'une recherche multiDS	Composant	Formulaire MultiDS	Incrusté zone principale	Fenêtre	sitools.user.component.forms.resultsProjectForm	/workspace/client-user/js/components/forms/resultsProjectForm.js
Plot		Composant	Visualisation des données	Incrusté zone principale	Fenêtre	ssitools.user.component.dataPlotter	/workspace/client-user/ st/components/plotDataPlotter.js
Edit Profile	Edition de son profil	Composant	UserSpace	Fenêtre modale		sitools.user.component.editProfile	/workspace/client-public/js/userProfile/editProfile.js
Columns definition	Affichage des colonnes dun dataset	Composant	Visualisation des données	Fenêtre modale	Fenêtre	sitools.user.component.columnsDefinition	/workspace/client-user/js/components/columnsDefinition/dependencies/columnsDefinition.js
Dataset OpenSearch	Affichage de la recherche openSearch sur un dataset	Composant	Graph Dataset Explorer	Incrusté zone principale	Fenêtre	sitools.user.component.datasetOpensearch	/workspace/client-user/js/components/datasetOpensearch/datasetOpensearch.js
Detail View	Vue de détail d'un record de dataset	Composant	Visualisation des données OpenSearch	Fenêtre modale	Fenêtre	sitools.user.component.viewDataDetail	/workspace/client-user/js/components/viewDataDetail/viewDataDetail.js
ImageWindow	Vue d'une image d'un record de dataset	Composant	Visualisation des données	Fenêtre modale	Fenêtre	sitools.widget.WindowImageViewer	/workspace/client-public/js/widgets/WindowImageViewer.js
Task status	description d'une tache en cours	Composant	Visualisation des données	Fenêtre modale		sitools.user.modules.userSpaceDependendestasks	/workspace/client-user/js/modules/userSpacedependencies/tasks.js
TaskResult Window	Affichage du résultat d'une tache asynchrone	Composant	Visualisation des données			sitools.usermodules.userSpaceDependencies.orderProp	/workspace/client-user/js/modules/userSpace/dependencies/orderProp.js
Task Detail	Détail d'une tache	Composant	Task Window			sitools.user.modules.userSpaceDependences.tasksDetails	/workspace/client-user/js/modules/userSpace/dependencies/taskDetail.js
View Json File	Affichage d'un fichier Json	Composant	Disk Space			viewFileContent	/workspace/client-user/js/def.js
Feed Viewer	Affichage des flux (projet, dataset, etc...)	Composant	Dataset Explorer Graph	Incrusté zone principale	Fenêtre	sitools.widget.FeedGridFlux	/workspace/client-public/js/feedsReader/feedsReader.js
FeedItemDetail	Détail d'un item de flux Res	Composant	Feed Viewer	Fenêtre modale		sitools.widget.feedItemDetails	/workspace/client-public/js/feedsReader/feedItemDetails.js
ResourcePluginParamsWindow	Affichage des parametres d'une resource plugin	Composant	Visualisation des données			sitools.user.component.dataviews.resourcePluginParamsPanel	/workspace/client-user/js/components/dataviews/resourcePluginParamsPanel.js
Fenêtre des unités	Permet de modifier les unités d'un composant ce formulaire	Widgets	Formulaires			sitools.admin.datasets.unitWin	/workspace/client-admin/js/datasets/unitWin.js
Version	Affichage de la version	Widgets	UserSpace			stools.component.version.sitoolsVersion	/workspace/client-public/js/version/sitoolsVersion.js
Login	Fenêtre de Login	Widgets	UserSpace			sitools.userProfile.Login	/workspace/client-public/js/userProfile/login.js
Register	Fenêtre pour s'enregistrer	Widgets	UserSpace			sitools.userProfile.Register	/workspace/client-public/js/userProfile/register.js
ResetPassword	Depuis la fenêtre de login	Widgets	UserSpace			sitools.userProfile.resetPassword	/workspace/client-public/js/userProfile/resetPassword.js
SorterTools	Depuis une dataview, Tri multiple	Widgets	Visualisation des données			sitools.widget.GridSorterToolbar	/workspace/client-public/js/wicgets/stoolsItems.js
filterTools	Depuis une dataview, Filtres multiple	Widgets	Visualisation des données			sitools.widget.filterTool	/workspace/client-public/js/wicgets/stoolsItems.js

6.7 Gestion des utilisateurs via l'API

6.7.1 Ajout d'un utilisateur

6.7.1.1 Pré requis

Afin d'ajouter un utilisateur dans SITools2 en utilisant l'API, il faut disposer d'un utilisateur ayant les droits administrateur.

6.7.1.2 Ajout de l'utilisateur

Quand on dispose de cet utilisateur, il faut faire un appel HTTP POST sur l'url suivante :

 HYPERLINK "http://localhost:8182/sitools/admin/security/users" http://localhost:8182/sitools/admin/security/users

-

Par exemple pour le login sitools et le mot de passe ptsc :

```
Authorization :Basic WVdSdGFXNDZZV1J0YVc0PQ==
```

- Une entête « Content-Type » avec comme valeur « application/json » pour informer le serveur que le corps du message est du JSON.
- Une entête « Accept » avec comme valeur « application/json » pour informer le serveur que l'on veut une réponse au format JSON.

Le corps de la requête doit contenir la structure JSON suivante :

```
{
        "firstName" : "firstname",
        "lastName" : "lastname",
        "email" : "email@domain.com",
        "identifier" : "login",
        "secret" : "password",
        "properties" : []
}
```

Si l'utilisateur est correctement créé, le serveur retourne une réponse avec un code 200 et un corps de message contenant un attribut « success ». Cet attribut a la valeur « true » si l'utilisateur est bien créé ou « false » dans le cas contraire.

Exemple de réponse :

L'utilisateur a bien été créé :

```
{
        "success" : true,
        "user" : {
                "identifier" : "login",
                "firstName" : "firstname",
                "lastName" : "lastname",
                "email" : "email@domain.com",
                "properties" : []
        }
}
```

L'utilisateur n'a pas été créé, un utilisateur existe déjà avec le login donné :

```
{
  "success": false,
  "message": "CREATE_USER ERROR: duplicate key value violates unique constraint \"PK_USER\""
}
```

6.7.2 Ajout de l'utilisateur à un rôle

6.7.2.1 Récupération des détails du rôle

Pour ajouter un utilisateur à un rôle, il faut commencer par récupérer les détails du rôle, en particulier la liste des utilisateurs associés.

Il faut faire une requête GET à l'adresse suivante (bien penser à mettre les mêmes entêtes HTTP que pour l'ajout d'un utilisateur)

```
http://localhost:8182/sitools/admin/security/roles/{id_role}
```

« {id_role} » doit être remplacé par l'identifiant du rôle pour lequel on veut ajouter un utilisateur

Le résultat de cette requête sera une structure JSON comme ceci :

```
{
  "success": true,
  "role": {
    "id": "admin",
    "name": "Administrator",
```

```
   "description": "Administrator can create/access/modify/delete users and groups, associate them to roles, then authorizations, and globally manage informations.",
   "users": [{
      "id": "admin"
    }
   ]
 }
}
```


Pour ajouter un nouvel utilisateur, récupérer le contenu de la clé « role » et ajouter l'identifiant de l'utilisateur au tableau « users ». Ensuite faire une requête PUT à la même adresse que précédemment (bien penser à mettre les mêmes entêtes HTTP que pour l'ajout d'un utilisateur).

Par exemple pour ajouter un utilisateur avec comme identifiant « login » au rôle « admin » :

Url : http://localhost:8182/sitools/admin/security/roles/admin

Corps du message :


```
{
        "id" : "admin",
        "name" : "Administrator",
        "description" : "Administrator can create/access/modify/delete users and groups, associate them to roles, then authorizations, and globally manage informations.",
        "users" : [{
                        "id" : "admin"
                }, {
                        "id" : "login1"
                }
        ]
}
```


Pour vérifier que l'utilisateur est bien ajouté au rôle, le statut de la requête doit être 200 et le contenu de la clé « success » à « true », comme pour l'ajout d'un utilisateur.

6.8 Génération de la documentation JavaScript

On utilise l'outil JsDuck pour générer la documentation du code JavaScript. Pour pouvoir utiliser JsDuck il est nécessaire d'avoir Ruby et RubyGem d'installé.

6.8.1 Installation de Ruby et Ruby gem

Installation du paquet ruby1.8

```
sudo apt-get install ruby1.8
```

Installation du paquet rubygem1.8

```
sudo apt-get install rubygems1.8
```

Installation du paquet ruby1.8-dev

```
sudo apt-get install ruby1.8-dev
```

6.8.2 Installation de JsDuck

```
sudo gem install jsduck
```

S'il y a besoin de passer un proxy ;

```
sudo gem install jsduck -p http://<username>:<password>@<proxy_host>:<proxy_port>
```

6.8.3 Génération de la documentation

Génération de la documentation JavaScript sur les différents projets de SITools2, se placer dans le dossier racine de SITools2 puis exécuter la commande suivante :

```
/var/lib/gems/1.8/bin/jsduck cots/extjs/ext-all-debug-w-comments.js workspace/client-public/js workspace/client-user/js workspace/client-admin/js --output jsDoc --warnings=-all
```

La documentation sera générée dans le dossier jsDoc.

6.9 Comportement de la sauvegarde des formulaires en mode incrusté

Le tableau suivant décrit les différents comportements lors de la sauvegarde d'un formulaire ou de la vue d'affichage des données en mode incrusté.

Restauration / *Sauvegarde*	*Avec formulaire*	*Avec formulaire mais formulaire n'existe plus*	*Sans formulaire*

Un formulaire	Ouverture du formulaire sauvegardé	Ouverture du premier formulaire de la liste	Ouverture des données
Affichage des données directement (pas de formulaire)	Ouverture du premier formulaire de la liste	Ouverture du premier formulaire de la liste	Ouverture des données
Formulaire + recherche	Ouverture du formulaire sauvegardé	Ouverture du premier formulaire de la liste	Ouverture des données

Globalement, dans le cas où l'on sauvegarde un formulaire et qu'on le retrouve, on affiche le formulaire sauvegardé. Sinon on applique le fonctionnement par défaut du mode incrusté. Ce fonctionnement par défaut est l'affichage des données s'il n'y a pas de formulaire, sinon affichage du premier formulaire de la liste.

6.10 Configuration du serveur Jetty

SITools2 fonctionne grâce à un serveur Jetty embarqué. Ce serveur peut être paramétré grâce à un ensemble de propriétés. Toutes ces propriétés sont modifiables dans le fichier « sitools.properties » :

```
Starter.REQUEST_HEADER_SIZE=524288
Starter.RESPONSE_HEADER_SIZE=524288
Starter.MIN_THREADS=1
Starter.MAX_THREADS=255
Starter.THREAD_MAX_IDLE_TIME_MS=60000
Starter.LOW_RESOURCES_MAX_IDLE_TIME_MS=2500
Starter.ACCEPTOR_THREADS=1
Starter.ACCEPT_QUEUE_SIZE=0
Starter.REQUEST_BUFFER_SIZE=8192
Starter.RESPONSE_BUFFER_SIZE=32768
Starter.IO_MAX_IDLE_TIME_MS=30000
Starter.SO_LINGER_TIME=1000
Starter.GRACEFUL_SHUTDOWN=0
```

Pour plus d'informations sur les paramètres de Jetty :

http://restlet.org/learn/javadocs/2.1/jse/ext/org/restlet/ext/jetty/JettyServerHelper.html

7 Documents applicables et de référence (A/R)

A/R	Référence	Titre

8 Glossaire et abréviations

8.1 Glossaire

Terme	Définition

8.2 Abréviations

Abréviation	Nom détaillé