Version 1.5

ORI-OAI-repository : Exposition des métadonnées via le protocole OAI-PMH

> ⚠ **Module optionnel**

Voir l'architecture du système

Dans quels cas l'utiliser

Pour exposer via le protocole OAI-PMH les fiches de métadonnées locales ou provenant de la moisson d'autres entrepôts, ou pour exposer des fiches directement à partir d'un répertoire du système de fichier local.

Composants optionnels

- **ORI-OAI-vocabulary** pour créer des « sets OAI » en fonction de différents vocabulaires
- **ORI-OAI-indexing** pour la recherche des fiches de métadonnées à exposer

Description

Le module ORI-OAI-repository se charge, via le protocole OAI-PMH, de l'exposition des fiches de métadonnées saisies dans le module ORI-OAI-workflow et/ou de celles provenant de moissons OAI. Utilisant le logiciel OAICat, il expose les fiches dans le but d'être moissonnés par tout moissonneur OAI.

Un mode "File system" est également disponible permettant d'exposer des fiches locales, en dehors de l'utilisation des autres modules ORI-OAI (fonctionnement "standalone").

Ce module gère également le concept de « sets OAI-PMH ». Cet aspect du protocole OAI permet d'exposer les fiches de métadonnées sous forme d'ensembles distincts. Ces ensembles sont souvent liés à une thématique particulière. Nous pouvons par exemple identifier l'ensemble de toutes les fiches pédagogiques au format LOM associées aux notions de mathématiques. Pour identifier les fiches correspondant aux différents ensembles, le module ORI-OAI-repository construit des requêtes suivant les critères associés aux différents sets et les envoie au module ORI-OAI-indexing.

Dans le mode d'utilisation "File System", les sets sont issus des sous-répertoires où sont physiquement stockées les fiches.

Voir la documentation technique

Spécifications

Objectifs, Rôle et contrats

Le module OAI-OAI-Repository permet au système ORI-OAI d'exposer ou de disséminer les fiches référencées par le système ORI-OAI via le protocole OAI-PMH. Cela permet de rendre moissonnable les documents déposés dans ORI et de fédérer plusieurs instances d'ORI-OAI, par l'intermédiaire de ORI-OAI-Harvester qui assemble les récoltes des différentes instances.

Objectifs et rôle

Les objectifs sont de rendre disponibles ("moissonnables") à la fois les fiches des documents *déposés* dans ORI, mais aussi celles qui sont *moissonnées* par ORI, par l'utilisation du protocole [OAI-PMH|http://www.openarchives.org/OAI/openarchivesprotocol.html]. Ces objectifs sont remplis par la collaboration de ce module avec les modules :

- ORI-OAI-Workflow pour les fiches déposées
- ORI-OAI-Harvester pour les fiches moissonnées
- ORI-OAI-Indexing pour constituer les ensembles de fiches d'aprés les métadonnées indexées.

Le rôle du module Repository consiste à presenter un service Web répondant aux six verbes définis par le protocole OAI, afin d'assurer ces objectifs.

Contrats

Les contrats du module repository consistent à fournir un service Web répondant aux six verbes OAI : * pour exposer les fiches déposées par le module Workflow

- pour exposer les fiches moissonnées par le module Harvester
  En passant par le module d'indexation, le module Repository récupère la liste des fiches, par paquet de taille déterminée, concernant la sélection exprimée dans la requêté OAI.

Cas d'utilisation

Exposition des fiches déposées

Deux aspects sont à prendre en compte :

- le controle de flux sur les verbes ListRecords et ListIdentifiers

Le protocole OAI prévoit de renvoyer les liste de fiches par paquets de taille fixe, avec dans l'entête la présence d'un resumptionToken (jeton de continuation) , qui permet d'avoir la suite de la liste à la prochaine requête. Dés qu'il n'y a plus de resupmtion token, on a atteint la fin de la liste. Le deuxième aspect est la moisson sélective. La liste renvoyée par Voici le schéma que je vois étant donné

- la moisson sélective permet de "filtrer" les fiches d'un format donné, par date ou par ensemble.

Exposition des fiches moissonnées

Les fiches moissonnées peuvent être exposée à travers un nouvel entrepôt OAI.

Exposition des fiches du sytème de fichier

Un répertoire peut être défini pour exposer les fiches qu'il contient physiquement, constotuant alors l'entrepôt OAI.

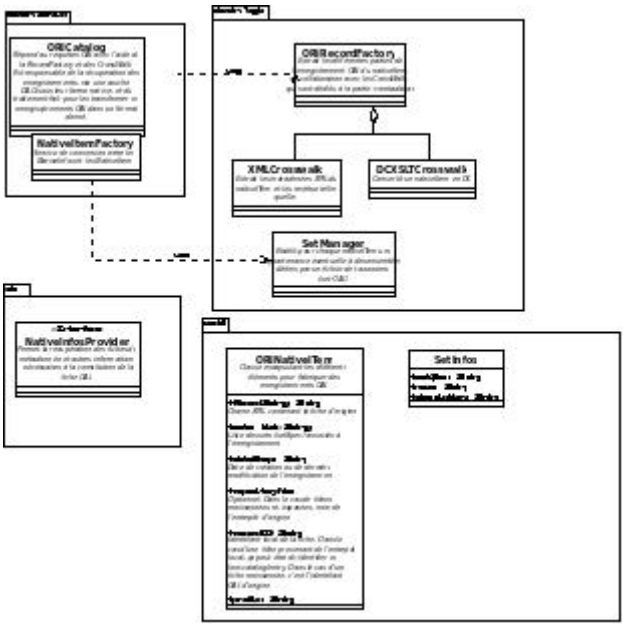
.

Choix techniques

Le module utilise l'API OAICat d'OCLC, qui fournit une servlet répondant aux six verbes OAI. Cet API permet de constituer des enregistrements OAI depuis différents types de resources : JDBC, XML, etc... Elle offre un Framework qui aide également à gérer le controle de flux par resumptionToken.

Diagrammes

Package



Changements de version

Changements version 1.5.0

- possibilité d'exposer les fiches provenant d'un répertoire du système de fichier
- configuration permettant l'utilisation en entrepôt de test (pour le moissonneur)

Changements version 1.4.0

- -démarrage indépendant du module vocabulary (feature #4276)
- compatibilité complète pour la validation open-archives OAI-PMH (bug 3839)
- n'expose que les fiches du workflow par défaut (respository-domain.xml)
- support du resumptionToken dans le ListSets
- possibilité de faire du filtre sur n'importe quelle métadonnée (pas uniquement le repository)
- possibilité de faire plusieurs sets à partir de plusieurs vocabulaires
- vocabulaire dewey utilisé par défaut à la place d'UNIT et vocabulaire gratuit/payant du LOM
- ajout des xpath pour la dewey en DDC et CDD
- corrections de bugs

Exposition des fiches indexées

La version 1.4.0+ permet de gérer plusieurs aspects de la dissémination des fiches indexées par le module ORI-OAI-indexing :

- Les **formats** disséminés : ce package inclu par défaut la dissémination à l'identique (IdentityCrosswalk) et la dissémination par transformation XSLT du format LOM vers OAI_DC.
  Voir la section _Configuration des formats exposés _ pour plus de détails.
- La **génération d'ensembles** (sets) basés sur des vocabulaires en fonction des champs de la fiche (taxonomiques ou autres). Le package inclus par défaut des ensembles pour les fiches LOM uniquement :
  - une classification des 100 premiers éléments Dewey (niveau 1 et 2)
  - une sélection par cout (payant non payant) pour les fiches LOM uniquement
  - la classification UNIT (commentée depuis la 1.4.0).
    Voir la section Configuration des ensembles à partir d'une classification pour plus de détails. Ces vocabulaires sont des vocabulaires centraux.
- La configuration par défaut n'expose que les fiches en provenance du workflow.
  Voir la section Filtrage des fiches indexées à exposer pour plus de détails.

Installation

Il existe plusieurs modes d'installation de ce module. Le mode recommandé est l'utilisation ori-oai-quick-install. Ceci vous permettra de déployer la suite ori-oai avec un minimum de personnalisation tout ceci en utilisant un seul fichier de configuration.

L'installation manuelle vous fera éditer manuellement différents fichiers afin de configurer au mieux votre application.

Il est préférable d'utiliser la première solution. En effet, celle-ci vous apportera un déploiement rapide de ORI-OAI sur un serveur de production avec une configuration de base. Vous pourrez toutefois après cette installation apporter toutes les configurations avancées que vous souhaitez à vos modules.

Reportez-vous à la documentation en ligne d'installation de ORI-OAI.

Installation manuelle

Pré-requis

JDK

L'entrepôt ORI-OAI est une application Java fonctionnant avec un **JDK 1.5** ou supérieur. La variable d'environnement JAVA_HOME doit exister et pointer sur le répertoire d'installation du JDK.

Toutefois, si cette variable pointe sur un autre JDK, les scripts *ant.bat* (pour Windows) et *ant.sh* (pour Linux) sont fournis, dans lesquels vous pouvez définir l'emplacement d'une JDK 1.5 utilisée de façon alternative pour le moissonneur.

ANT

Les tâches de compilation, de déploiement et certaines actions utilisent [ANT|http://ant.apache.org/] 1.6. La variable d'environnement ANT_HOME doit exister et pointer sur le répertoire d'installation de ANT.

Tomcat

Une version de Tomcat 6 doit être disponible sur la machine de déploiement, et la variable d'environnement CATALINA_HOME doit être définie pour pointer son emplacement.

> Pour assurer l'application de fonctionner avec l'encodage UTF-8, il ajouter cette ligne dans le fichier startup.sh ou catalina.sh (répertoire tomcat-xxx/bin) : export CATALINA_OPTS="*-Dfile.encoding=UTF-8* $CATALINA_OPTS"

Installation manuelle de l'entrepôt OAI

Installation

> Une possibilité de configuration centralisée permet de changer automatiquement la valeur de certains paramètres entre crochets et en majuscules, grâce à la configuration du module *quick-install*. Les fichiers *build.properties* et *ori-oaicat.properties* sont concernés. Ainsi, vous avez le choix entre une installation "par module" où vous devez *créer et editer* ces fichiers à partir de init-build.properties et conf/properties/ori-oaicat.example.properties, et une installation "centralisée" où ces fichier seront automatiquement générés lors de la compilation et du déploiement. Les sections suivantes décrivent l'installation "par module", pour l'installation centralisée, se reporter en plus à la documentation du [quick-install|http://sourcesup.cru.fr/ori-oai-commons/quick-install/installation.html].

L'installation se fait en 5 étapes :

1. décompression de l'archive ori-repository-x-x.zip

2. configuration du fichier build.properties

Un fichier build.properties doit être crée par recopie de init-build.properties.
Il faut indiquer l'endroit ou l'on veut installer l'application, l'emplacement du Tomcat, ainsi que l'emplacement d'un JDK 1.5+. Les paramètres *entre crochet* sont à modifier, ou seront modifés par la configuration centrale du quick-install :

```
#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#
## 2) Installation manuelle du module
# Dans ce cas, il est nécessaire de commenter
# le paramètre commons.parameters.central.file.url
#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#

# Cette partie du fichier doit être mise à jour avant
# la première utilisation de votre application dans votre environnement
#JDK 1.5 directory
java.home=[JAVA_HOME]
#Tomcat directory
tomcat.home=[PATH_TOMCAT_REPOSITORY]
#Deployment directory
deploy.home=[PATH_TOMCAT_REPOSITORY]/webapps
#Nom de ditribution de l'application - distribution name
app.name.deploy=[CONTEXT_REPOSITORY]
#Log Directory (optionnal - default : ${tomcat.home}/logs )
#log.dir=E:/logs
#Log mode : prod or debug
log.mode=prod
```

Les chemins utilisant des anti-slashes comme sous Windows, doivent s'écrire soit avec un double anti-slash soit avec un slash, comme sous Linux.

3. Configurer les fichiers de propriétés du module

Un fichier properties/ori-oaicat.properties doit être créé à partir de l'exemple ori-oaicat.example.properties, dans lequel les valeurs entre crochet doivent être adaptées :

```
# ORI-OAI-repository Configuration

AbstractCatalog.millisecondsToLive=3600000
AbstractCatalog.harvestable=true

# datestamp granularity
#AbstractCatalog.granularity=YYYY-MM-DD | YYYY-MM-DDThh:mm:ssZ
# note : in case of reexposition of many repositories with different granularities
# choose YYYY-MM-DD form
AbstractCatalog.granularity=YYYY-MM-DD

# Important note : final OAI identifiers will be as :
# 1. for harvested record : original OAI id of the harvested store
# 2. for local record (provided by workflow module) :
# oai:[SCHEME_IDENTIFIER][REPOSITORY_IDENTIFIER]:[indexing id]
# where indexing id is local id provided by workflow module
# TAKE CARE not to change often these values if you don't want to have
# your records harvested several times with different ids
#(have to read http://www.openarchives.org/OAI/2.0/guidelines-oai-identifier.htm)

ORICatalog.maxListSize=40
ORICatalog.listSets.maxListSize=1000
ORIRecordFactory.repositoryIdentifier=[REPOSITORY_IDENTIFIER]
# note : this scheme will be automatically prefixed with "oai:"
ORIRecordFactory.identifierScheme=[REPOSITORY_SCHEME_IDENTIFIER]

# Custom Identify response values
Identify.repositoryName=[REPOSITORY_NAME]
Identify.adminEmail=[SMTP_ADMINISTRATOR_MAIL]
Identify.earliestDatestamp=2006-12-04T00:00:00Z
Identify.deletedRecord=no
Identify.description.1=<description><oai-identifier xmlns\="http\:
//www.openarchives.org/OAI/2.0/oai-identifier" xmlns\:xsi\=
"http\://www.w3.org/2001/XMLSchema-instance" xsi\:schemaLocation\=
"http\://www.openarchives.org/OAI/2.0/oai-identifier
http\://www.openarchives.org/OAI/2.0/oai-identifier.xsd"
><scheme>oai</scheme><repositoryIdentifier>oai\:[REPOSITORY_SCHEME_IDENTIFIER][REPOSITORY_IDENTIFIER]

# xslt files for crosswalk formats
dctolomFile=properties/xslt/lom2dc.xsl

# File System parameters (used ONLY if you choose fileSystem provider in repository-domain.xml)
filesystem.directory=/path/to/records/directory
filesystem.md.namespace=http://www.openarchives.org/OAI/2.0/oai_dc/
filesystem.filter=*.xml

# Web service for ori-oai-indexing (used ONLY if you choose indexer provider in
repository-domain.xml)
indexing.ws.wsdlDocumentUrl=http:
//[HOST_INDEXING]:[PORT_INDEXING]/[CONTEXT_INDEXING]/xfire/IndexingService?WSDL
#indexing.ws.wsdlDocumentUrl=http://demo.ori-oai.org/indexing/xfire/IndexingService?WDSL
indexing.ws.lookupServiceOnStartup=false

# Web service for ori-oai-vocabulary
vocabulary.ws.wsdlDocumentUrl=http:
//[HOST_VOCABULARY]:[PORT_VOCABULARY]/[CONTEXT_VOCABULARY]/xfire/OriVocabularyService?WSDL
vocabulary.ws.lookupServiceOnStartup=false


# static MD names
static.mdIdentifier=md-ori-oai-id
static.mdFormat=md-ori-oai-namespace
static.mdRepository=md-ori-oai-repository
static.mdDatestamp=md-ori-oai-datestamp

# use to append repository name to oai identifier
# note : must match worklow_name defined in indexing and search modules properties as
<workflow_name>ori-oai-workflow</workflow_name>
workflow.repositoryName=ori-oai-workflow
```

* Les éléments sous **Identify.*** correspondent à l'identité de votre entrepôt :

- Les **identifiants OAI** de vos fiches locales seront de la forme : **oai:[SCHEME_IDENTIFIER][REPOSITORY_IDENTIFIER]**

:[identifiant d'indexation]

- Renseigner les paramètres **filesystem.*** pour l'utilisation en mode "File System"
  - **filesystem.directory** : définit le répertoire dans lequel on va disposer les fiches à exposer
  - **filesystem.md.namespace** : spécifie le format de métadonnées exposé (http://ltsc.ieee.org/xsd/LOM pour le LOM)
  - **filesystem.filter** : spécifie un filtrage des fichiers considérés, à l'aide d'une expression régulière, par défaut "*.xml" (voir documentation WilcardFileFilter)
- Renseigner les paramètres **indexing.*** pour l'utilisation *en mode "Indexing"** **indexing.ws.wsdlDocumentUrl** : définit l'URL d'accès au module ORI-OAI-indexing
  - **indexing.ws.lookupServiceOnStartup :** initialise la connection au démarrage (laisser à ***false***)
- Renseigner les paramètres **vocabulary.*** pour l'utilisation de vocabulaires (set OAI en mode "Indexing") :
  - **vocabulary.ws.wsdlDocumentUrl** : définit l'URL d'accès au module ORI-OAI-indexing
  - **vocabulary.ws.lookupServiceOnStartup :** initialise la connection au démarrage (laisser à ***false***)

- Optionnel : Par défaut, un fichier log4j.properties est produit automatiquement à partir des fichiers d'exemple (voir section "Niveau de debogage").

4. Configuration en mode "FileSystem"

Mode "File system" versus mode "Indexing"

Par défaut, la configuration proposée utilise le mode Indexing, qui permet d'exposer les fiches indexées, qui sont soient issues du workflow, soit du moissonnage, et définit des dépendances à ces modules. Pour des détails sur cette fonctionnalité, consulter la section Implémentation File system.

Pour activer le mode **standalone** "File System ", **il faut suivre les étapes suivantes :**

4.1. Activer le bean correspondant.

Pour ce faire, il faut modifier le fichier *conf/properties/repository-domain.xml* :

- **commenter** l'alias **setManager** défini sur *indexerSetManager* ainsi que le bean **nativeInfosProvider** utilisant la classe **org.orioai.repository.dao.NativeInfosProviderImpl**
- **décommenter** à la place l'alias **setManager** défini sur fs*SetManager* et le **nativeInfosProvider** mais qui utilise cette fois-ci la classe **org.orioai.repository.dao.fs.SimpleFSNativeInfosProvider** :

```
<alias name="fsSetManager" alias="setManager"/>

 <bean id="nativeInfosProvider"
  class="org.orioai.repository.dao.fs.SimpleFSNativeInfosProvider">
  <description>
   this nativeInfosProvider enables to expose quickly via
   OAI-PMH some LOM / OAI_DC files stored in a simple File
   System Directory
  </description>

  <property name="dataMDFilesDirectory">
   <value>${filesystem.directory}</value>
  </property>

  <property name="metadataFormat">
   <value>${filesystem.md.namespace}</value>
  </property>

  <property name="namespacePrefixMapper">
   <ref bean="namespacePrefixMapper" />
  </property>

  <property name="workflowName"
   value="${workflow.repositoryName}" />

 </bean>
```

4.2. Paramétrer les propriétés

Il faut ensuite paramétrer ce nouveau nativeInfosProvider, et pour cela il faut renseigner les propriétés suivantes dans *conf/properties/orioaicat.properties* :

- **filesystem.directory** : définit le répertoire dans lequel on va disposer les fiches à exposer
- **filesystem.md.namespace** : spécifie le format de métadonnées exposé (http://ltsc.ieee.org/xsd/LOM pour le LOM)
- **filesystem.filter** : spécifie un filtrage des fichiers considérés, à l'aide d'une expression régulière, par défaut "*.xml" (voir documentation WilcardFileFilter)

Vous n'avez plus qu'à lancer votre Tomcat et tester l'entrepôt via son interface graphique ... ou/et en utilisant la version de démonstration d'ORI-OAI (**.exe** ou **.jar**, installable en quelques clic) pour vous moissonner !

5. Configurations avancées (optionnel)

Pour le mode "Indexing" **uniquement** (utilisation avec les modules ORI-OAI-indexing et ORI-OAI-vocabulary), il y a la possibilité, en consultant la section Configuration avancée, de pousser plus avant la configuration de certains aspects comme :

- le filtrage des fiches
- la constitution d'ensembles
- les formats exposés

> **Note**
>
> Par défaut, et sans modification des fichiers décrits dans ces sections, la configuration permet d'exposer aux format Dublin Core (oai_dc) et LOM(Learning Object Metadata) , avec les ensembles issus de la classification Dewey, les fiches provenant du module Workflow (**mode "Indexing"**).

6. Déploiement

Lancer la tâche **ant all**: vous pouvez alors accéder à l'interface Web avec le contexte **ori-oai-repository**, par exemple : http://localhost:8080/ori-oai-repository

Configurations avancées

Configurations avancées

Filtrage des fiches indexées à exposer

Dans le fichier **properties/repository-filters.xml**, il y a possibilité de filtrer les fiches indexées que l'on veut exposer. Ce filtrage s'opère sur n'importe quelle **métadonnée indexée** (ex. : les entrepôts OAI , les formats de fiches....).

Chaque filtrage s'opère en ajoutant une entrée dans une des deux catégories de filtre :

- commonFiltersList : **ce sont les filtres communs à tous les formats**.
- formatFiltersList : **ce sont les filtres spécifiques à chaque format**.

En l'**absence de filtre**, **toutes les fiches** de métadonnées indexées **sont exposées** (aucun filtrage n'est effectué).

1. Exemples de filtres communs à tous les formats

Dans la configuration proposée **par défaut**, on filtre sur l'**entrepôt d'origine** (md-ori-oai-repository), en ne choisissant d'exposer que les fiches en provenance du workflow (ori-oai-workflow), donc de l'entrepôt local, ce qui correspond à l'entrée dans le bean **commonFiltersList** ainsi définie :

```
<entry key="md-ori-oai-repository">
 <list>
  <value>ori-oai-workflow</value>
 </list>
</entry>
```

En l'absence de liste, aucun filtrage n'est effectué sur l'entrepôt d'origine.

Autre exemple : Filtrage sur les **formats de métadonnées**
Définition d'une liste de formats, qui permet de restreindre la liste définie par les Crosswalks (voir section "Configuration des formats exposés", plus bas) :

```
<entry key="md-ori-oai-namespace">
 <list>
  <value>http://www.openarchives.org/OAI/2.0/oai_dc/</value>
</list>
</entry>
```

En l'absence de liste ou de filtre de ce type, le filtrage sur les formats se base uniquement sur les Crosswalks (voir "Configuration des formats exposés", plus bas).

2. Exemples de filtres propres à un format

Si on souhaite n'exposer que les fiches LOM dont le titre est "java" ou commence par "info", il faut ajouter dans le bean **formatFiltersList** l'entrée suivante :

```
<entry key="http://ltsc.ieee.org/xsd/LOM">
<map>
  <entry key="//lom:general/lom:title/lom:string[starts-with(@language,'fr')]">
<list>
    <value>java</value>
    <value>info*</value>
   </list>
  </entry>
 </map>
</entry>
```

Configuration des formats exposés

Les formats exposés sont configurés dans le fichier **properties/repository-crosswalks.xml**.

Deux types de configurations simples existent : l'exposition d'une fiche à l'identique et l'exposition d'une fiche par transformation XSLT.

Exposition d'un format à l'dentique (IdentityCrosswalk)

Ce type d'exposition ne fait qu'ajouter une couche OAI à une fiche de métadonnées déjà composée dans le format requis.

La configuration de ce type d'exposition se résume à définir le **nom d'espace** (namespace), l'**URL du schéma XSD** et le **préfixe** utilisé pour ce format. Ci-dessous, exemple pour l'exposition en LOM de fiches LOM :

```
<bean id="LOMIdentityCrosswalk"
 class="org.orioai.repository.domain.logic.crosswalk.IdentityCrosswalk" init-method="init">
 <property name="nameSpaceURL" value="http://ltsc.ieee.org/xsd/LOM" />
<property name="schemaURL" value="http://ltsc.ieee.org/xsd/lomv1.0/lom.xsd" />
<property name="inPrefix" value="lom" />
</bean>
```

La propriété **inPrefix** définit quel format d'entrée ce Crosswalk utilise, alors que le nom d'espace et le schema XSD sont ceux de la fiche produite en sortie (ici les deux schémas coïncident).

Exposition d'un format par transformation XSLT

Il s'agit de convertir une fiche d'un format donné vers un format différent, en utilisant une feuille de transformation XSL

La configuration doit, en plus des paramètres du format, préciser le **nom du fichier XSLT**. Ci-dessous, exemple pour l'exposition en Dublin Core de fiches LOM :

```
<bean id="LOMtoDCCrosswalk"
 class="org.orioai.repository.domain.logic.crosswalk.XSLTCrosswalk"
 init-method="init">
 <property name="nameSpaceURL" value="http://www.openarchives.org/OAI/2.0/oai_dc/ " />
<property name="schemaURL" value="http://www.openarchives.org/OAI/2.0/oai_dc.xsd" />
<property name="inPrefix" value="lom" />
 <property name="outPrefix" value="oai_dc" />
 <property name="xsltFile" value="${xslt:dclom}"/>

</bean>
```

La propriété **inPrefix** définit le format d'entrée (ici le LOM). La propriété **outPrefix** désigne le préfixe du format en sortie, après la transformation (ici le Dublin Core).

Crosswalk composite

Lorsqu'un format à produire en sortie utilise plusieurs formats possibles en entrée, comme c'est le cas pour OAI_DC qui peut provenir aussi bien de fiches LOM que Dublin Core, on configure une troisième sorte de Crosswalk, le Crosswalk composite, qui rassemble plusieurs Crosswalk des types précédents :

```
<bean id="compositeDCCrosswalk"
 class="org.orioai.repository.domain.logic.crosswalk.CompositeCrosswalk">
 <constructor-arg>
 <value>http://www.openarchives.org/OAI/2.0/oai_dc/</value>
</constructor-arg>
 <constructor-arg>
 <value>http://www.openarchives.org/OAI/2.0/oai_dc.xsd</value>
</constructor-arg>
 <property name="outPrefix" value="oai_dc" />
 <property name="crosswalks">
 <list>
 <ref bean="DCIdentityCrosswalk" />

 <ref bean="LOMtoDCCrosswalk" />
 </list>
 </property>
</bean>
```

Configuration des ensembles (sets) à partir d'une classification

(voir Cas d'utilisation : définir des nouveaux sets)

La génération d'un lot de sets s'appuie sur un **vocabulaire de référence**, qui doit être géré dans le module ORI-OAI-Vocabulary, sous la forme d'un fichier XML. Ce vocabulaire peut être utilisé par ailleurs dans les autres modules. Ce fichier est donc centralisé pour assurer la cohérence de l'ensemble, et accessible par Web Service via le module ORI-OAI-Vocabulary.

Le vocabulaire de référence va donner autant de sets qu'il contient de valeurs : **1 valeur dans le vocabulaire = 1 set**.

Une implémentation simulacre (Mock) est fournie au cas où aucun module vocabulary ne serait disponible. Elle permet d'utiliser des fichiers sur le système de fichier local.

Comme les autres Web Service, il est configurable dans le fichier **repository-ws.xml**, mais par défaut les valeurs importées depuis le fichier **ori-oaicat.properties** (décrit plus haut) suffisent au fonctionnement de base.

La **configuration des ensembles** proprement dite se situe dans le fichier **conf/properties/repository-sets.xml**. Elle se fait à l'aide des **beans de class OaiSetInfos**. Chacun de ces beans permet de définir le **vocabulaire de référence** et la **façon dont il est mis en relation avec une métadonnée** d'un format particulier.

Par défaut, trois beans OaiSetInfos sont définis dans le fichier livré par défaut (dont un, celui d'UNIT, est commenté et donc inactif). L'établissement voulant exposer ses fiches aura en effet vraisemblablement une préférence pour une classification propre à son établissement, ce qui ne l'empêchera pas, loin de là, d'être interopérable avec UNIT grâce à l'utilisation du schéma pivot Dewey.

Ces trois ensembles prédéfinis sont :

- les 100 premiers codes Dewey
- le champ "cout"
- la classification UNIT (commentée donc)

```
<bean id="set100DeweyTaxonomy" class="org.orioai.repository.domain.model.set.OaiSetInfos"
init-method="init">
  <property name="vocabularyId" value="dewey_100_taxonomie_regexp"/>
  <property name="rootTag" value="vdex"/>
  <property name="termXpath" value="//vdex:term"/>
<property name="valueXpath" value=".//orioai:value"/>
<property name="setSpecXpath" value="vdex:termIdentifier"/>
  <property name="setNameXpath" value="vdex:caption/vdex:langstring[@language = 'fr']"/>
  <property name="vocabularyNameXpath" value="//vdex:vocabName/vdex:langstring[@language =
'fr']"/>

  <property name="xpathSources">
   <map>
    <entry>
     <key>
      <value>http://ltsc.ieee.org/xsd/LOM</value>
</key>
     <bean class="org.orioai.repository.domain.model.set.OaiSetSourceInfos">
      <property name="xpath">
       <list>
        <value>
//lom:classification/lom:taxonPath[lom:source/lom:string='dewey']/lom:taxon/lom:id</value>
<value>
//lom:classification/lom:taxonPath[starts-with(lom:source/lom:string[starts-with(@language,'fr
<value>
//lom:classification/lom:taxonPath[starts-with(lom:source/lom:string[starts-with(@language,'en
</list>
      </property>
     </bean>

    </entry>
    <!--entry>
     <key>
      <value>http://www.openarchives.org/OAI/2.0/oai_dc/</value>
</key>
     <bean class="org.orioai.repository.domain.model.set.OaiSetSourceInfos">
      <property name="xpath" value="//dc:dewey"/>
</bean>
    </entry-->
   </map>
  </property>
 </bean>
```

- Configuration du vocabulaire:
  Dans la première partie du bean, on trouve la propriété vocabularyId qui définit l'identifiant du vocabulaire utilisé, rootTag et valueTag qui désignent les tag XML respectivement de la racine du vocabulaire et celui contenant les valeurs de la catégorie, qui seront comparées aux valeurs trouvée dans la taxonomie des fiches.
  Lorsque qu'une des valeurs d'une catégorie correspond avec celle trouvée dans la fiche, la fiche appartient à l'ensemble de cette catégorie.
  Les propriétés setSpec et setName désignent les éléments XML(tag ou attribut) pour le nom et le code des ensembles générés.
- Mise en correspondance avec une donnée de la fiche :
  La propriété xpathSources regroupent sous forme de Map la correspondance, pour différents formats de fiche, avec les valeurs définies pour les ensembles de la classification. Ainsi, pour chaque entrée la clé (key) désigne le nom d'espace d'un format de fiche, et les sous-propriétés xpath et luceneFieldName, respectivement le chemin XPATH de la donnée dans la fiche et le nom utilisé dans le module ORI-OAI-indexing pour indexer cette donnée.

Niveau de déboggage

Le niveau de débogage peut être modifié en éditant le fichier properties\log4j.*.properties. Deux modèles de fichiers sont fournis, log4j.prod.propeties en mode production, et log4j.debug.properties en mode débogage.

.Les niveaux disponibles sont : *debug*, *info*, *warn*, *error* et *fatal*, du plus prolixe au plus concis. Le niveau de déboggage influe sur les performances de l'application.

Note : Ce fichier est produit automatiquement lors de l'appel de la tâche ANT deploy, en fonction de ce qui est défini dans build.properties. Si vous désirez le changer à la main, ne pas oublier de préciser le chemin du fichier de log, par exemple :

```
log4j.appender.R.File=E:\Java\jakarta-tomcat-5.0.28\logs\ori-repository.log
```

Test

Pour accéder à l'interface Web de l'entrepôt, utilisez l'URL :

```
http://[HOST_INSTALL]:8180/ori-oai-repository
```

Vous devez accéder à cette interface :

Pour l'affichage en XML brut, vous pouvez également tester l'URL suivante dans un navigateur :

```
http://[HOST_INSTALL]:8180/ori-oai-repository/OAIHandler?verb=Identify
```

Aspects pratiques

- Implémentation FileSystem

Implémentation FileSystem

Implémentation FileSystem - Mise en place rapide d'un entrepôt OAI-PMH "standalone"

Dans un certain nombre de cas d'utilisation, rendre moissonnable un ensemble de fiches simplement en les mettant dans un répertoire donné est intéressant.

Parmi ces cas d'utilisation, on retiendra notamment le cas d'un système de référencement/indexation pré-existant que l'on souhaiterait rendre rapidement moissonnable.

Utilisé seul (mode standalone), ori-oai-repository permet de rendre moissonnable un répertoire contenant des fiches de métadonnées (LOM par exemple) ... Ce répertoire joue alors le rôle d'un entrepôt OAI-PMH.

L'installation et la configuration de ce seul module sont alors rapides !

Rendre moissonnable un système pré-existant

On notera que quelque soit la solution envisagée, être capable de générer depuis ce système pré-existant des fiches XML dans le format souhaité (format manipulé usuellement par les entrepôts OAI-PMH) est une nécessité. Cela passe par l'élaboration d'une "moulinette" permettant l'export des informations en format XML d'un schéma donné.
Dans le cas d'un entrepôt OAI-PMH dédié aux ressources pédagogiques, le schéma sera le LOM (LOMFR / SupLOMFR).

- Une fois que l'on sait faire cela, on peut alors envisager de rendre moissonnable son système en ajoutant les fonctionnalités "entrepôt OAI-PMH" directement dans l'applicatif, cela en redéveloppant toute la couche logicielle adéquate. On peut utiliser des bibliothèques adapatés déjà développés et disponibles dans le langage de son applicatif, celles-ci contenant tout l'aspect métier d'OAI-PMH. *il existe de telles bibliothèques dans la plupart des langages* et donc pour la plupart des "plateformes web" (php, python/zope, perl ... et bien sûr Java/J2EE) .
  On obtient alors une intégration dite forte de la couche logicielle OAI-PMH dans son applicatif.

- On peut aussi, et au moins dans une première étape, se constituer **rapidement** un entrepôt OAI-PMH en mettant en place un outil spécialisé permettant de rendre accessible via OAI-PMH l'ensemble de fichiers XML prédisposés dans un répertoire donné. Ces fiches seront régulièrement mis à jour simplement via une moulinette (telle que décrite plus haut) :
  le mieux ici est que la moulinette puisse tourner régulièrement et mettre à jour ou ajouter si nécessaire les fiches XML dans le répertoire indiqué.

ori-oai-repository - fonctionnement en configuration FileSystem

Dans sa configuration FileSystem, ori-oai-repository fonctionne seul : **aucun des autres modules ORI-OAI n'est nécessaire**, il permet de rendre accessible **rapidement** via OAI-PMH un ensemble de fichiers XML (LOM par exemple) stockés dans un dossier de son ordinateur, de son serveur.

Voici ses caractéristiques :

- il se base sur la date de modification du système de fichiers ; il est donc préférable (mais cependant pas obligatoire) de modifier/remplacer un fichier seulement si nécessaire,
- il permet de répondre au protocole OAI-PMH sans le support des suppressions de fiches (comportement autorisé par le protocole OAI-PMH),
- il permet d'utiliser les Sets OAI-PMH (comportement autorisé par le protocole OAI-PMH), en reproduisant l'arborescence des répertoires
- il ne permet pas d'utiliser les **fonctions de filtrage** qui sont liées à l'indexation
- il utilise les fonctionnalités usuelles apportées par ORI-OAI, notamment :
  - une configuration aisée des paramètres importants dans la mise en place d'un entrepôt OAI-PMH,
  - la conversion des fiches en OAI_DC via des XSL adapatés.
    => dans le cas de fiches LOM, disposer uniquement les fiches LOM dans un dossier permet aussi de répondre en OAI_DC

installation et mise en place

installation / configuration usuelle

On suivra au mieux la documentation fournie ici http://www.ori-oai.org/display/ORIOAIrepository/Installation#Installation-Installation.

On notera que certaines fonctionnalités basées sur l'indexation des données ne sont actuellement pas disponibles. Il s'agit principalement du filtrage et des Sets depuis les classifications. De plus, en choisissant ce mode Filesystem, on doit valuer la propriété **Identify.deletedRecord=no**, car ce système ne permet pas de garder trace des fichiers supprimés.

Il faudra donc être attentif aux paragraphes donnés dans **Configuration et installation de l'entrepôt OAI** :

- Installation
- Fichiers de propriétés

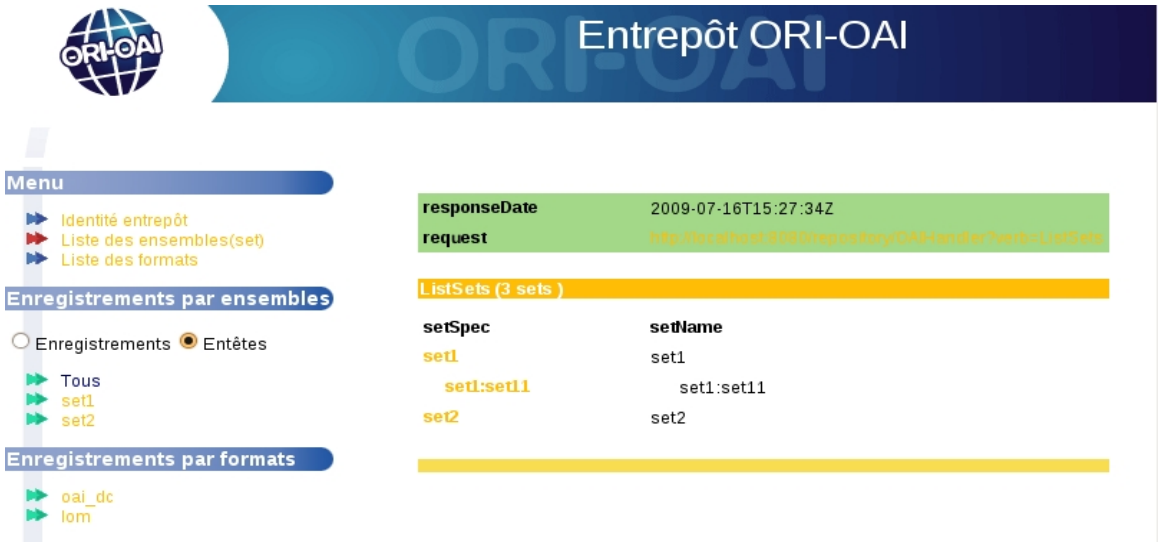
Sets et arborescence des répertoires

Le DirectorySetManager est une implémentation qui permet de reproduire l'arborescence des répertoires du système de fichier sous forme d'ensemble hiérarchiques.

Ainsi, pour un répertoire "multiset" défini comme racine de l'entrepôt (paramètre filesystem.directory), ayant cette structure :

```
-set1
  -set11
-set2
```

on aura la "vue" OAI-PMH sous forme de sets OAI :



Configurations de tests pour le moissonneur

Des répertoires représentant des entrepôts "File System" de test sont fournis dans le répertoire **conf/properties/filesystem/test-repository**.

Ces entrepôts peuvent servir à tester un moissonneur OAI, il suffit de suivre les instructions dans le fichier **conf/properties/filesystem/test-repository/README.txt.**

Pour un test basic de bon fonctionnement, il suffit de définir les propriétés suivantes (dans ori-oaicat.properties) :

```
filesystem.directory=/path/to/install/ori-oai-repository/conf/properties/filesystem/test-repository/r
//www.openarchives.org/OAI/2.0/oai_dc/
filesystem.filter=*.xml
```

En lancant ensuite le module, dans l'IHM on doit avoir 3 fiches oai_dc, sans ensemble OAI définis :

Utilisation

Cas d'utilisation : exposer des fiches depuis le système de fichier

Voir implémentation File System

Cas d'utilisation : définir des nouveaux sets

Objectif :

- On veut ajouter des **nouveaux sets OAI** afin de proposer aux moissonneurs des ensembles de fiches selon un critère précis.
- Dans cet exemple, on veut utiliser la métadonnées **dc:publisher** du format Dublin Core, et en fonction de sa valeur, dire si telle fiche appartient ou non à tel set.

Les sets du module repository sont construits d'après un vocabulaire au format VDEX géré par le module vocabulary. Ce vocabulaire contient les valeurs de référence qui seront comparées à la valeur enregistrée dans la fiche pour une métadonnée ciblée. Cette valeur saisie dans la fiche est définie par une expression XPATH.

Etape 1 : module vocabulary

Il faudra éventuellement ajouter un vocabulaire dans le repertoire "conf/properties/ori_vocabularies/override" du **module vocabulary**. Par exemple **my-univ-sets.xml**.

Ce sera notamment le cas si les vocabulaires par défaut ne conviennent pas :

- besoin propre dans tel(s) module(s) ;
- restriction de la liste de valeurs d'un vocabulaire existant par défaut ;
- personnalisation des noms (labels) des sets.

Voici un exemple avec deux sets :

```
<?xml version="1.0" encoding="utf-8"?>
<vdex:vdex xmlns:vdex="http://www.imsglobal.org/xsd/imsvdex_v1p0"
xmlns:orioai="http://www.ori-oai.org/static/xsd/orioaivocab"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.imsglobal.org/xsd/imsvdex_v1p0
http://www.imsglobal.org/xsd/imsvdex_v1p0.xsd"
profileType="flatTokenTerms">
<vdex:vocabName>
  <vdex:langstring language="fr">my-univ-sets</vdex:langstring>
</vdex:vocabName>
<vdex:vocabIdentifier isRegistered="false">my-univ-sets</vdex:vocabIdentifier>

<vdex:term validIndex="true">
  <vdex:termIdentifier>Set1</vdex:termIdentifier>
  <vdex:caption>
   <vdex:langstring language="fr">[Label pour Set1]</vdex:langstring>
  </vdex:caption>
</vdex:term>

<vdex:term validIndex="true">
  <vdex:termIdentifier>Set2</vdex:termIdentifier>
  <vdex:caption>
   <vdex:langstring language="fr">[Label pour Set2]</vdex:langstring>
  </vdex:caption>
</vdex:term>

</vdex:vdex>
```

Etape 2 : module repository

Ensuite, il faut déclarer dans le fichier "conf/properties/repository-sets.xml" du **module repository**, un bean de class OaiSetInfos "mySets" référençant le vocabulaire défini plus haut en correspondance avec l'expression XPATH de l'attribut **valueXpath** :

```
<bean id="mySets"
class="org.orioai.repository.domain.model.set.OaiSetInfos"
init-method="init">
        <property name="vocabularyId" value="my-univ-sets"/>
        <property name="rootTag" value="vdex"/>
        <property name="termXpath" value="//vdex:term"/>
        <property name="valueXpath" value=".//vdex:termIdentifier"/>
        <property name="setSpecXpath" value="vdex:termIdentifier"/>
        <property name="setNameXpath"
value="vdex:caption/vdex:langstring[@language = 'fr']"/>
        <property name="vocabularyNameXpath"
value="//vdex:vocabName/vdex:langstring[@language = 'fr']"/>
        <property name="vocabularyNameDefault" value="my-univ-sets"/>

        <property name="xpathSources">
            <map>
                <entry>
                    <key>

<value>http://www.openarchives.org/OAI/2.0/oai_dc/</value>
                    </key>
                    <bean
class="org.orioai.repository.domain.model.set.OaiSetSourceInfos">
                        <property name="xpath">
                            <list>
                                <value>//dc:publisher</value>
                            </list>
                        </property>
                    </bean>

                </entry>
            </map>
        </property>
    </bean>
```

Enfin, il faut déclarer ce bean dans la liste du bean setManager (toujours dans le fichier repository-sets.xml) :

```
<property name="setInfosList">
 <list>
  ...
  ...
  <ref bean="mySets"/>
 </list>
</property>
```

Etape 3 : module indexing

Si ce n'est pas déjà la cas, il faut indiquer au moteur d'indexation que la métadonnée doit être indexée, pour que sa valeur puisse être comparée aux valeurs du vocabulaire de référence.

Pour cela, il faut ajouter le XPATH de la métadonnée dans la section du format correspondant (dans l'exemple //dc:publisher) dans le fichier properties/liusConfig.xml du **module indexing** :

```
<!-- DC -->
<xmlFile ns="http://purl.org/dc/elements/1.1/" setBoost="1.0">
<indexer class="org.orioai.indexing.index.indexer.OriOaiXmlFileIndexer">
 <mime>text/xml</mime>
</indexer>
<fields>
...
...
<luceneField name="%2F%2Fdc%3Apublisher" xpathSelect="//dc:publisher" type="Text" setBoost="1.5"/>
```

Note : dans cet exemple, le champs dc:publisher est déjà déclaré dans la configuration par défaut.

La procédure complète est décrite dans la documentation du module ORI-OAI-indexing, section "Ajout d'un nouveau xpath à indexer dans le format" :

http://www.ori-oai.org/display/ORIOAIindexing/Personnalisation+de+la+configuration