****

**Nakala**

**Guide d’utilisation de l’application batch d’envoi de données**

|  |  |  |  |
| --- | --- | --- | --- |
| *Historique des révisions* | | | |
| *Version* | *Date* | *Auteur* | *Modifications* |
| V1 | 07/11/2013 | Thomas Francart | Version initiale |
| V2 | 27/01/2014 | Thomas Francart | Indication des créations de liens inverses |
| V3 | 01/02/2014 | Thomas Francart | Ajout de la procédure de lancement batch |
| V4 | 10/02/2014 | Thomas Francart | Modification pour tenir compte du nouveau schéma XSD |
| V5 | 28/02/2014 | Thomas Francart | Renommage de l’application en Nakala |
| V6 | 30/03/2016 | Thomas Francart | Mise à jour suite aux évolutions du batch (options de remplacement et nouvelle structure de répertoire d’output) |
| V7 | 22/03/2017 | Thomas Francart | Mise à jour du batch et des options sur la nouvelle API REST de Nakala. |
| V8 | 02/2018 | Thomas Francart | Introduction du rdf :type pour le format CSV |

Sommaire

[Qu’est-ce que l’application batch d’envoi de données ? 4](#_Toc506997057)

[Comment installer l’application batch d’envoi de données ? 4](#_Toc506997058)

[Télécharger l’application 4](#_Toc506997059)

[Installer l’application 4](#_Toc506997060)

[Prérequis 4](#_Toc506997061)

[Installation 4](#_Toc506997062)

[Comment lancer l’application batch d’envoi de données ? 4](#_Toc506997063)

[Paramètres de lancement 4](#_Toc506997064)

[Lancement avec les scripts start.bat ou start.sh 6](#_Toc506997065)

[Lancement direct en java, avec la clé d’APIdans un fichier 6](#_Toc506997066)

[Fonctionnement du batch 6](#_Toc506997067)

[Description du format des paquets d’entrée à envoyer 7](#_Toc506997068)

[Structure des descripteurs en XML 8](#_Toc506997069)

[Structure des descripteurs en CSV 13](#_Toc506997070)

[Structure de base 14](#_Toc506997071)

[Eléments Nakala 14](#_Toc506997072)

[rdf :type pour indiquer « Data » ou « Collection » ? 15](#_Toc506997073)

[Limites du format CSV 15](#_Toc506997074)

[La validation FACILE 15](#_Toc506997075)

[Création de collections à partir du batch 15](#_Toc506997076)

[Description du format des paquets de sortie 16](#_Toc506997077)

[Description du fichier de rapport de batch 18](#_Toc506997078)

[Codes d’erreur 19](#_Toc506997079)

[Annexe : script utilitaire “paquet-create.sh” 20](#_Toc506997080)

# Qu’est-ce que l’application batch d’envoi de données ?

L’application batch d’envoi de données de Nakala, ou application « console », est une application ligne de commande en Java permettant d’envoyer à Nakala des paquets contenant un fichier de données d’une part et un fichier de métadonnées descriptive d’autre part.

# Comment installer l’application batch d’envoi de données ?

## Télécharger l’application

L’application batch est téléchargeable depuis l’interface web d’administration de Nakala, accessible avec vos identifiants Renater. Une fois dans l’interface web, naviguez dans l’onglet « Envoi d’un paquet », et cliquez sur le lien de téléchargement de « nakala-console.zip » dans cet onglet.

## Installer l’application

### Prérequis

Pour que l’application batch puisse fonctionner :

* Java 5 ou plus récent doit être installé sur votre machine ;
* La commande « java » doit être dans votre PATH de lancement ; pour vérifier cela, ouvrez une fenêtre de commande (Linux ou Windows), et tapez la commande « java -version » ; la version de java installer sur votre machine doit s’afficher ;

### Installation

* Dézippez le zip téléchargé dans le répertoire de votre choix, et c’est tout !
* Une fois dézippé, le répertoire nakala-console doit contenir :
  + Les répertoires :
    - Cfg (fichiers de configuration du batch)
    - Docs (cette documentation)
    - Input (le répertoire d’entrée par défaut)
    - Lib (les bibliothèques Java nécessaires au fonctionnement du batch)
    - Schema (le schema XSD des descripteurs XML du batch)
  + Les fichiers start.bat, start.sh, paquet-create.sh et nakala-console.jar
* Sous Linux, assurez-vous que « start.sh » est exécutable en exécutant « chmod 755 start.sh » ;
* Sous Linux, assurez-vous que le batch a les droits de lecture sur le répertoire d’entrée, et les droits d’écriture sur les répertoires de sortie et d’erreur ;

# Comment lancer l’application batch d’envoi de données ?

## Paramètres de lancement

La console prend en paramètres les informations suivantes

|  |  |  |  |
| --- | --- | --- | --- |
| **Nom du paramètre** | **Nom du paramètre dans la commande java** | **Description** | **Valeur par défaut** |
| Aide | -h ou –help | Affiche un message d’aide |  |
| email | -email | l’emaild’un compte de déposant dans Nakalaassocié à une clé d’API, avec lequel les données seront envoyées |  |
| Id du projet | -projectId | Le handle du projet dans lequel les données seront créées. Ce paramètre est optionnel si l’utilisateur n’a accès qu’à un seul projet. Sinon il est obligatoire. |  |
| Répertoire d’entrée | -inputFolder | Répertoire contenant les paquets à envoyer | input |
| Répertoire de sortie | -outputFolder | Répertoire dans lequel seront enregistrés les paquets de résultat. Ce répertoire contiendra 2 sous-répertoires, « ok » pour les paquets envoyés avec succès, et « error » pour les paquets en erreur. | output |
| Nettoyage du répertoire de sortie | -cleanOutput | Indique à l’application console de supprimer tout le contenu du répertoire de sortie si celui-ci n’est pas vide. Si cette option n’est pas activé, et que le répertoire de sortie contient déjà des fichiers, cela est considéré comme une erreur (pour éviter que des outputs de lancement successifs du batch se mélangent). | false |
| Validation avec FACILE | -facileValidation | Indique si on veut que les paquets envoyés avec la console fassent l’objet d’une validation par le service FACILE (voir ci-dessous). | false |
| Chemin vers le fichier contenant la clé d’API | -apiKeyFile | Donne le chemin vers un fichier qui contient la clé d’API à utiliser. la clé d’API doit être sauvegardée dans ce fichier, sans encodage particulier (en faisant un copier/coller depuis le backoffice Nakala) ; | nakala-key.txt |
| Remplacement | -replace | Indique si l’on souhaite remplacer les métadonnées de descriptions déjà existantes dans Nakala avec celles envoyées, ainsi que le fichier de données correspondant. Les paquets peuvent contenir uniquement le descripteur de métadonnées, sans données, pour ne remplacer que les métadonnées. | false |
| Remplacement des données seulement | -updateDataOnly | Indique si l’on souhaite seulement mettre à jour les fichiers de données, sans mettre à jour les métadonnées. Cette option est incompatible avec « -replace ».. | false |

## Lancement avec les scripts start.bat ou start.sh

Les scripts start.bat ou start.sh demandent les paramètres de lancement à l’utilisateur de façon interactive :

* Répertoire d’entrée ; (appuyer directement sur Entrée pour laisser la valeur par défaut « input »)
* Répertoire de sortie ;(appuyer directement sur Entrée pour laisser la valeur par défaut « output »)
* Nettoyage du répertoire de sortie (appuyer directement sur Entrée pour laisser la valeur par défaut « false »)
* Validation avec FACILE ; (appuyer directement sur Entrée pour laisser la valeur par défaut « NO »)
* Remplacement des données et métadonnées existantes (appuyer directement sur Entrée pour laisser la valeur par défaut « false »)
* Remplacement des données existantes (appuyer directement sur Entrée pour laisser la valeur par défaut « false »)
* Email ;
* Chemin vers le fichier contenant la clé d’API. Par défaut le batch lit la clé d’API dans le fichier « nakala-key.txt » ; la clé d’API doit être sauvegardée dans ce fichier, sans encodage particulier (en faisant un copier/coller depuis le backoffice Nakala) ;
* Handle du projet (appuyer directement sur Entrée pour laisser vide ; cela n’est possible que si l’utilisateur n’a accès qu’à un seul projet, sinon cette valeur est obligatoire) ;

## Lancement direct en java, avec la clé d’APIdans un fichier

Il est également possible de lancer directement la commande en java, en utilisant les paramètres indiqués dans le tableau ci-dessus.

Le synopsys de la commande est :

java -jar nakala-console.jar -email $EMAIL -inputFolder $INPUT\_FOLDER -outputFolder $OUTPUT\_FOLDER [-projectId $HANDLE] [-facileValidation] [-cleanOutput] [-replace] [-updateDataOnly] –apiKeyFile $API\_KEY\_FILE

Le fichier de cléd’API API\_KEY\_FILE doitcontenirla cléd’API sans encodageparticulier et sans saut de ligne.

# Fonctionnement du batch

Lorsqu’on le lance, le batch effectue les opérations suivantes :

* Si l’option de nettoyage du répertoire d’output a été passée en paramètre, les fichiers de ce répertoire sont supprimés ; si le répertoire d’output existe déjà et n’est pas vide, et que l’option de nettoyagen’a pas été activée, le batch s’arrête ;
* Il lit les paquets « zip » dans le répertoire d’entrée ; la lecture s’effectue par ordre alphabétique des noms de fichier ;
* Ces paquets sont envoyés au serveur Nakala;
* Le serveur renvoie une réponse « DONE » si les conditions suivantes sont remplies :
  + L’email et la clé d’APIindiqués sont valides ;
  + La validation FACILE n’a pas été demandée, ou bien elle a été demandée et a renvoyé un code de retour valide ;
  + Les métadonnées du paquet sont valides (voir ci-dessous) ;
  + Le fichier et les métadonnées sont correctement enregistrés dans Nakala ;
* Le serveur renvoi une réponse « ERROR » si l’une des conditions ci-dessus n’est pas remplie ;
* Pour chaque paquet d’entrée, un fichier de réponse en XML est créé dans le répertoire de sortie :
  + Soit dans le sous-répertoire « ok » si le paquet a été créé avec succès ;
  + Soit dans le sous-répertoire « error » si le paquet n’a pas été créé ;
* A la fin du traitement, un rapport d’exécution en XML est créé à la racine du répertoire de sortie ;

Les fichiers dans le répertoire d’entrée sont déplacés dans le répertoire de sortie au fur et à mesure de leur traitement, soit dans le sous-répertoire « ok » ou dans le sous-répertoire « error ». A la fin d’un traitement batch, le répertoire d’entrée sera donc vide si le batch est terminé ; si le batch s’est arrêté au milieu du traitement, le répertoire d’entrée contiendra donc toujours les paquets qui n’ont pas été traités, ce qui permet ensuite de relancer le batch sur ces paquets.

# Description du format des paquets d’entrée à envoyer

Les fichiers dans le répertoire d’entrée doivent être des fichiers ZIP (extension \*.zip).

Chaque zip doit contenir :

* Un fichier de données (image, son, vidéo, texte, texte, etc.). L’extension du fichier n’importe pas si la validation FACILE n’est pas requise ;
* Un fichier XML ou un fichier CSV décrivant les métadonnées du fichier ;

Les noms des fichiers de données et de métadonnées n’ont pas d’importance.Ces deux fichiers doivent se trouver à la racine du zip, ou bien dans un sous-répertoire à l’intérieur du zip. Si vous n’êtes pas sûrs de la structure du zip, utilisez un utilitaire zip tel que 7Zip ou le gestionnaire d’archives pour visualiser son contenu.

Le fichier de métadonnées doit être soit :

* Un fichier XML obligatoirement avec l’extension \*.xml ;
* Ou un fichier CSV obligatoirement avec l’extension \*.csv ;

## Structure des descripteurs en XML

Lorsque le fichier de métadonnées est un fichier XML, il doit respecter les règles suivantes :

* Il doit commencer par une racine « nkl:Data » et doit déclarer l’espace de noms « nkl » comme [http://nakala.fr/schema#](http://nakala.fr/schema). Voici une entête typique d’un fichier :

<nkl:Dataxmlns:nkl="http://nakala.fr/schema#"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:dcterms="http://purl.org/dc/terms/"

xsi:schemaLocation="http://purl.org/dc/terms/ http://dublincore.org/schemas/xmls/qdc/2008/02/11/dcterms.xsd">

< !--snip -->

</nkl:Data>

* Il doit ensuite contenir 4 métadonnées Dublin Coreobligatoires : title, creator, type, created, dans cet ordre :

<dcterms:title>Venezuela : Estado Portuguesa : Guanare : Esquina Calle 13 et Carrera 5 : Vue d'une rue bordée d'habitations collectives et d'une friche urbaine</dcterms:title>

<dcterms:creator>Pouyllau, Michel</dcterms:creator>

<dcterms:type>Image</dcterms:type>

<dcterms:created>1983</dcterms:created>

* Il peut ensuite contenir n’importe quelles métadonnées génériques exprimées à l’aide du vocabulaire Dublin CoreTerms (<http://dublincore.org/documents/dcmi-terms/>). On se reportera à la documentation du vocabulaire Dublin CoreTerms pour une liste et une description de ces métadonnées. Toutes ces balises doivent contenir une valeur textuelle de métadonnée à l’intérieur de la balise, par exemple :

<dcterms:subject>habitat traditionnel</dcterms:subject>

<dcterms:subject>friche urbaine</dcterms:subject>

<dcterms:coverage>Venezuela, BolivarianRepublic Of, Guanare</dcterms:coverage>

<dcterms:spatialxsi:type="dcterms:Point">east=-69.748947; north=9.04549;name=Guanare</dcterms:spatial>

<dcterms:created>1983</dcterms:created>

<dcterms:type>Image</dcterms:type>

<dcterms:rights>Michel Pouyllau</dcterms:rights>

<dcterms:source>Centre de Documentation REGARDS</dcterms:source>

* *Note avancée* : Certaines de ces métadonnées Dublin CoreTerms peuvent être des références à des URIs, comme par exemple :

<dcterms:licence xsi:type="dcterms:URI">http://creativecommons.org/licenses/by-nc-sa/2.0/fr/</dcterms:licence>

Ces valeurs seront alors interprétées comme des « objectproperties » dans le RDF de Nakala, et non des litéraux.

* *Note avancée* : Il est également possible d’indiquer sur les métadonnées Dublin Core une information de langue en ajoutant un attribut « xml:lang »:

<dcterms:subjectxml:lang="es">Estado Portuguesa</dcterms:subject>

* *Note avancée* : Il est également possible d’indiquer sur les métadonnées Dublin Core une information de type de donnée (datatype) en ajoutant un attribut « xsi:type» ; les types de données possibles sont ceux du Dublin Core documentés à http://dublincore.org/documents/dcmi-terms/#H5 (Box, Point, Period, etc.). On ne peut pas indiqué de datatype RDF ;

<dcterms:spatialxsi:type="dcterms:Point">east=-69.748947; north=9.04549;name=Guanare</dcterms:spatial>

* Ensuite, il doit contenir dans cet ordre des métadonnées spécifiques à Nakala, qui appartiennent à l’espace de nom [http://nakala.fr/schema#](http://nakala.fr/schema). Ces métadonnées spécifiques sont décrites dans le tableau ci-dessous. Toutes ces métadonnées sont facultatives, sauf « nkl:dataFormat » seulementsi on demande la validation FACILE.

|  |  |
| --- | --- |
| **Métadonnée** | **Description** |
| nkl:inCollection | Indique l’appartenance de la donnée envoyée à une collection dans Nakala. La balise doit contenir l’identifiant handle de la collection. Cet identifiant doit exister comme un identifiant de collection, sinon l’envoi du paquet générera une erreur. Cette balise peut être répétée plusieurs fois pour des données qui doivent appartenir à plusieurs collections.  Exemples :  <nkl:inCollection>11280/a19a5e26</nkl:inCollection> |
| nkl:relation | Indique une relation entre la donnée envoyée et une autre donnée dans Nakala.   * Le type de la relation doit obligatoirement être indiqué dans l’attribut « type ». Seule l’une de ces valeurs est autorisée comme valeur de l’attribut type :   + <http://purl.org/dc/terms/isVersionOf>   + <http://purl.org/dc/terms/hasFormat>   + <http://purl.org/dc/terms/isFormatOf>   + <http://purl.org/dc/terms/hasVersion>   + <http://purl.org/dc/terms/isVersionOf>   + <http://purl.org/dc/terms/replaces>   + <http://purl.org/dc/terms/isReplacedBy>   + <http://purl.org/dc/terms/hasPart>   + <http://purl.org/dc/terms/isPartOf>   + <http://purl.org/dc/terms/isReferencedBy>   + <http://purl.org/dc/terms/references>   + <http://purl.org/dc/terms/isRequiredBy>   + <http://purl.org/dc/terms/requires>   + <http://purl.org/dc/terms/relation>   Si l’attribut “type” contientuneautrevaleur, l’envoi du paquetgénérerauneerreur. La verification ne tientcependant pas compte des majuscules/minuscules.   * La balise doit contenir l’identifiant handle de l’autre donnée que l’on souhaite référencer dans Nakala. Cet identifiant doit exister dans la base Nakala. Si celui-ci n’existe pas, l’envoi du paquet générera une erreur. * Lorsqu’une relation est indiquée, Nakala créera automatiquement la relation inverse (si « A dcterms:requires B » est créé, Nakala insérera automatiquement « B dcterms:isRequiredBy A »).   Cette balise peut être répétée plusieurs fois pour indiquer plusieurs relations.  Exemples :  <nkl:relation type="http://purl.org/dc/terms/isVersionOf">11280/bf86bec2</nkl:relation>  <nkl:relation type="http://purl.org/dc/terms/isrequiredby">11280/bf86bec2</nkl:relation> |
| nkl:accessEmail | Dans le cas de données protégées en lecture, indique l’email d’un utilisateur RENATER qui sera autorisé à accéder à la donnée. Cette balise peut être répétée plusieurs fois pour indiquer plusieurs adresses emails.  Les données pour lesquelles cette métadonnée est absente sont publiques.  Exemples :  <nkl:accessEmail>charles@cnrs.fr</nkl:accessEmail>  <nkl:accessEmail>mon.email@cnrs.fr</nkl:accessEmail> |
| nkl:dataFormat | Indique le format de la donnée pour la validation avec FACILE. Cette balise doit contenir l’une des valeurs listée à <http://facile.cines.fr/?media=XML>.  Cette balise est obligatoire si la validation FACILE est demandée.  Cette balise ne peut pas être répétée plusieurs fois.  Note : cette balise doit contenir obligatoirement l’une des valeurs listée à <http://facile.cines.fr/?media=XML>. Au moment de l’écriture de cette documentation, cettelisteest la suivante, maisilestconseillé de se reporter à la documentation enligne de FACILE pour la liste la plus à jour :   * AAC AAC * AIFF PCM * DAE UTF-8 1.4.1 * FLAC FLAC 1.2.1 * GIF 87a * GIF 89a * GeoTIFF * HDF5 1.0 * HDF5 2.0 * JPEG * JPEG2000 * JPEG 1.00 * JPEG 1.01 * MKV AVC/FLAC * MPEG-4 AAC * MPEG-4 AVC * MPEG-4 AVC/AAC * ODS 1.0 * ODS 1.1 * ODS 1.2 * ODT 1.0 * ODT 1.1 * ODT 1.2 * OGG Theora/Vorbis * OGG Vorbis * PDF 1.4 * PDF 1.4 * PDFA 2a * PDFA 2u * PDF 1.4 * PDF 1.0 * PDF 1.1 * PDF 1.2 * PDF 1.3 * PDF 1.4 * PDF 1.5 * PDF 1.6 * PDF 1.7 * PNG 1.0 * PNG 1.1 * PNG 1.2 * SIARD * SVG UTF-8 1.0 * SVG UTF-8 1.1 * SVG UTF-8 1.2 * TEI UTF-8 2.9.1 * TIFF * TIFF 4.0 * TIFF 5.0 * TIFF 6.0 * TXT UTF-8 * WAV * XDMF * XML UTF-8 1.0 * XSD   Notez que le format de la donnée peut aussi être indiqué de manière libre dans l’attribut dcterms:format. |
| nkl :identifier | Indique le handle de la donnée que l’on souhaite modifier. Cette balise est utile uniquement lorsqu’on souhaite faire une mise à jour de métadonnées ou de données déjà chargées dans Nakala, c’est-à-dire si une des options « -replace » ou « -updateDataOnly » a été utilisée au lancement. Il ne faut pas indiquer cette métadonnée pour le chargement d’une nouvelle donnée.  Exemples :  <nkl:identifier>11280/abcd1234</nkl:identifier> |

L’ordre de ces balises est important et elles doivent apparaitre dans l’ordre donné dans ce tableau.

Voici un exemple complet de fichier de métadonnées XML d’entrée :

<nkl:Dataxmlns:nkl="http://nakala.fr/schema#"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:dcterms="http://purl.org/dc/terms/"

xsi:schemaLocation="http://purl.org/dc/terms/ http://dublincore.org/schemas/xmls/qdc/2008/02/11/dcterms.xsd">

<dcterms:title>Venezuela : Estado Portuguesa : Guanare : Esquina Calle 13 et Carrera 5 : Vue d'une rue bordée d'habitations collectives et d'une friche urbaine</dcterms:title>

<dcterms:creator>Pouyllau, Michel</dcterms:creator>

<dcterms:type>Image</dcterms:type>

<dcterms:created>1983</dcterms:created>

<dcterms:identifier>http://medihal.archives-ouvertes.fr/medihal-00668270</dcterms:identifier>

<dcterms:identifier>http://medihal.archives-ouvertes.fr/docs/00/66/82/70/IMG/REGARDS\_Venezuela\_DC065.jpg</dcterms:identifier>

<dcterms:subject>[SHS:GEO] Humanities and Social Sciences/Geography</dcterms:subject>

<dcterms:coverage>Venezuela, BolivarianRepublic Of, Guanare</dcterms:coverage>

<dcterms:spatialxsi:type="dcterms:Point">east=-69.748947; north=9.04549;name=Guanare</dcterms:spatial>

<dcterms:rights>Michel Pouyllau</dcterms:rights>

<dcterms:licence xsi:type="dcterms:URI">http://creativecommons.org/licenses/by-nc-sa/2.0/fr/</dcterms:licence>

<dcterms:source>Centre de Documentation REGARDS</dcterms:source>

<nkl:inCollection>11280/a19a5e26</nkl:inCollection>

<nkl:relationtype="http://purl.org/dc/terms/isVersionOf">11280/bf86bec2</nkl:relation>

<nkl:relation type="http://purl.org/dc/terms/isrequiredby">11280/bf86bec2</nkl:relation>

</nkl:Data>

## Structure des descripteurs en CSV

### Structure de base

Les fichiers CSV (Comma Separated Values) sont des tableaux dont chaque colonne est séparée par une virgule. Si le descripteur de métadonnées est un fichier CSV, il doit respecter les règles suivantes :

* Le fichier doit contenir 2 colonnes :
  + La première colonne donne le nom d’une métadonnée Dublin CoreTerms (<http://dublincore.org/documents/dcmi-terms/>). On se reportera à la documentation du vocabulaire Dublin CoreTerms pour une liste et une description de ces métadonnées. Le nom des métadonnées doit être indiqué sans espace de nom particulier (donc par exemple « title », « created », « abstract », etc.)
  + La deuxième colonne (après la virgule), donne la valeur de la métadonnée correspondante ; Si la valeur contient elle-même une virgule, elle doit être mise entre double guillemet ;
* Le fichier doit contenir obligatoirement au minimum les métadonnées suivantes (dans n’importe quel ordre et à n’importe quel endroit du fichier – contrairement au format XML qui impose d’avoir ces éléments en premier) :
  + title
  + creator
  + type
  + created
* Si on souhaite répéter une métadonnée plusieurs fois, il suffit de répéter la ligne avec le même nom de métadonnée ;
* La première ligne d’entête indiquant les noms des colonnes est optionnelle ; si elle est présente, elle sera ignorée ;

Un exemple basique de descripteur CSV ressemble donc à ceci :

Métadonnée, Valeur

title,Gargantua compisse les parisiens

creator,Gustave Doré

type,Gravure

created,01/01/1873

subject,"Paris, France"

subject,"Rabelais, François"

### Eléments Nakala

* Le CSV peut également contenir des métadonnées spécifiques Nakala, qui doivent être préfixées par « nkl : » pour les distinguer des métadonnées Dublin Core. Il est donc possible d’indiquer :
  + nkl:inCollection
  + nkl:accessEmail
  + nkl:dataFormat

On se reportera à la documentation du format XML pour ces éléments.

* Il n’est pas possible de créer l’équivalent de « nkl :relation » en CSV ;

### rdf :type pour indiquer « Data » ou « Collection » ?

Il peut être nécessaire dans certaines situations d’indiquer explicitement si le descripteur est un descripteur de Collection ou de Data. De la même façon que le descripteur XML a pour racine « nkl:Data » ou « nkl:Collection », le descripteur CSV peut indiquer cette information à l’aide de l’élément « rdf:type », avec pour valeur soit « nkl:Data » soit « nkl:Collection ». Cette information est optionnelle.

Par exemple :

Métadonnée, Valeur

rdf:type,nkl:Data

title,Gargantua compisse les parisiens

…

L’utilisation typique du « rdf:type » dans ces descripteurs CSV est la mise à jour des métadonnées uniquement via le batch.

Cette ligne additionnelle est optionnelle ; par défaut, les types d’entrées sont considérés comme des « nkl:Data ».

### Limites du format CSV

Le format CSV possède des limites par rapport au format XML :

* Il n’est pas possible d’indiquer une langue sur une valeur ;
* Il n’est pas possible d’indiquer un datatype sur une valeur ;
* Il n’est pas possible de créer de « nkl:relation » ;

Notez qu’il est possible de créer ou de mettre à jour une Collection Nakala à l’aide d’un descripteur CSV.

# La validation FACILE

Si la validation FACILE a été demandée au moment du lancement du batch, Nakala appellera le service de validation FACILE à l’adresse <http://facile.cines.fr/?media=XML>.

Pour cela, la métadonnée « nkl:dataFormat » doit obligatoirement être présente dans les fichiers de métadonnées des paquets. Si la validation FACILE est demandée mais que cette métadonnée est absente du paquet, l’envoi du paquet générera une erreur.

# Création de collections à partir du batch

Le batch permet d’envoyer des données mais également de créer des collections dans la hiérarchie des collections. Pour cela, il faut :

* Créer un zip ne contenant pas de fichier de données, mais uniquement un fichier de métadonnées XML  (il n’est pas possible de créer des Collections avec un descripteur CSV);
* Utiliser la balise racine « nkl:Collection » à la place de « nkl:Data » ;

La balise nkl:Collection peut contenir :

* Une balise dcterms:title obligatoire
* Toutes les balises Dublin CoreTerms comme pour les données ;
* Uniquement la balise « nkl:inCollection » de l’espace de nom Nakala. Les autres balises « nkl:relation », « nkl:accessEmail », etc. ne sont pas autorisées. Cette balise « nkl:inCollection » sert à indiquer la collection parente de la collection envoyée. Elle doit contenir pour cela l’identifiant handle de la collection parente. Si cette balise n’est pas présente, la collection sera créée directement sous la collection racine du compte email utilisé pour lancer le batch.

Exemple de fichier pour créer une Collection par le batch :

<nkl:Collectionxmlns:nkl="http://nakala.fr/schema#"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:dcterms="http://purl.org/dc/terms/"

xsi:schemaLocation="http://purl.org/dc/terms/ http://dublincore.org/schemas/xmls/qdc/2008/02/11/dcterms.xsd">

<dcterms:title>Le titre de ma collection</dcterms:title>

<nkl:inCollection>11280/a19a5e26</nkl:inCollection>

</nkl:Collection>

*Note avancée* : pour les collections, le contenu de la balise dcterms :title sera copiée automatiquement par Nakala dans un attribut « skos :prefLabel ».

Lorsque la collection est créée par le batch, elle est immédiatement visible dans l’interface web de gestion des collections de Nakala.

# Description du format des paquets de sortie

Chaque fichier présent dans le répertoire d’entrée donne lieu à un fichier de résultat dans le répertoire de sortie, au format XML ou CSV en fonction du format utilisé pour le descripteur d’entrée. Ce fichier de résultat est une copie du fichier de métadonnées du paquet, enrichi avec des métadonnées spécifiques à Nakala, décrites dans le tableau ci-dessous :

|  |  |
| --- | --- |
| **Nom de la métadonnée Nakala** | **Description** |
| nkl:identifier | Contient l’identifiant handle assigné à la donnée stockée dans Nakala. C’est cet identifiant qu’il faut utiliser pour accéder à la donnée en ligne à l’adresse [[URI\_Nakala]/data/<handle](http://silo.huma-num.fr/silo/data/%3chandle)>, et pour créer des relations vers cette donnée dans les balises « nkl:relation » des données envoyées. |
| nkl:checksum | Un checksum du fichier de données calculée par Nakala |
| nkl:fileName | Le nom du fichier de données qui était présent dans le zip |

Exemple de fichier de sortie XML :

<?xml version="1.0" encoding="UTF-8" standalone="no"?><nkl:Dataxmlns:nkl="http://nakala.fr/schema#" xmlns:dcterms="http://purl.org/dc/terms/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://purl.org/dc/terms/ http://dublincore.org/schemas/xmls/qdc/2008/02/11/dcterms.xsd">

<dcterms:identifier>http://medihal.archives-ouvertes.fr/medihal-00668270</dcterms:identifier>

<dcterms:identifier>http://medihal.archives-ouvertes.fr/docs/00/66/82/70/IMG/REGARDS\_Venezuela\_DC065.jpg</dcterms:identifier>

<dcterms:title>Venezuela : Estado Portuguesa : Guanare : Esquina Calle 13 et

Carrera 5 : Vue d'une rue bordée d'habitations collectives et d'une friche

urbaine</dcterms:title>

<dcterms:creator>Pouyllau, Michel</dcterms:creator>

< !--snip : toutes les autres métadonnées du fichier d’entrée -->

<nkl:identifier>11280/17d7f0f1</nkl:identifier>

<nkl:checksum>7f449e3096b191379211a99e7411577bcc9f8a4b</nkl:checksum>

<nkl:fileName>test.file</nkl:fileName>

</nkl:Data>

Exemple de fichier de sortie CSV :

Métadonnée, Valeur

title,Gargantua compisse les parisiens

creator,Gustave Doré

type,Gravure

created,01/01/1873

subject,"Paris, France"

subject,"Rabelais, François"

nkl:identifier,"11280/17d7f0f1"

nkl:fileName,"Gargantua\_compisse\_les\_Parisiens.jpeg"

nkl:checksum,"7f449e3096b191379211a99e7411577bcc9f8a4b"

# Description du fichier de rapport de batch

A la fin de chaquelancement, l’application batch génère un fichier de rapport en XML dans le repertoire de sortie. Ce fichierestnommé “report\_<timestamp>.xml”

Ce fichier donner les informationssuivantesdans un format XML :

* Date de début et de fin du batch
* Nombre de fichiers traités
* Nombre de fichiers valides
* Nombre de fichiers invalides

Et pour chaquefichier :

* Son nom
* Sa date de début de traitement et de fin de traitement
* Son status DONE ou ERRORSi c’estuneerreur :
* Un numéroà 2 chiffreindiquant un code d’erreur (voir plus bas)
* un message d’erreurdonnant la cause de l’erreur

Exemple de fichier de rapport :

<report\_data>

<date\_started>2013-11-06T19:06:51.467+01:00</date\_started>

<date\_ended>2013-11-06T19:10:13.051+01:00</date\_ended>

<nb\_files\_total>7</nb\_files\_total>

<nb\_files\_valid>5</nb\_files\_valid>

<nb\_files\_invalid>2</nb\_files\_invalid>

<batch\_status>DONE</batch\_status>

<files\_treated>

<file>

<file\_name>01-dctermsCreation</file\_name>

<file\_status>DONE</file\_status>

<date\_started>2013-11-06T19:06:51.473+01:00</date\_started>

<date\_ended>2013-11-06T19:07:56.989+01:00</date\_ended>

<file\_status\_code>0</file\_status\_code>

</file>

<file>

<file\_name>02-URI-translation</file\_name>

<file\_status>DONE</file\_status>

<date\_started>2013-11-06T19:07:56.996+01:00</date\_started>

<date\_ended>2013-11-06T19:08:30.559+01:00</date\_ended>

<file\_status\_code>0</file\_status\_code>

</file>

<file>

<file\_name>03-accessEmails</file\_name>

<file\_status>DONE</file\_status>

<date\_started>2013-11-06T19:08:30.560+01:00</date\_started>

<date\_ended>2013-11-06T19:09:04.090+01:00</date\_ended>

<file\_status\_code>0</file\_status\_code>

</file>

<file>

<file\_name>01b-dctermsWithXmlLang</file\_name>

<file\_status>FAILED</file\_status>

<date\_started>2016-03-30T16:49:14.876+02:00</date\_started>

<date\_ended>2016-03-30T16:49:14.959+02:00</date\_ended>

<file\_status\_code>56</file\_status\_code>

<file\_status\_desc>Cette donnée déjà existe dans Nakala: http://www.nakala.fr/resource/11280/f7229bc2</file\_status\_desc>

</file>< !-- etc. -->

</report\_data>

# Codes d’erreur

Le rapport peut contenir les codes d’erreur suivants :

|  |  |  |
| --- | --- | --- |
| **Code d’erreur** | **Nom** | **Explications** |
| 0 | Pas d’erreur | Pas d’erreur particulière (le fichier est correctement traité). |
| 40 | CODE\_GENERIC\_CLIENT\_EXCEPTION | Une exception est intervenue côté client (dans le batch), sans plus d’indications. |
| 41 | CODE\_ZIP\_FILE\_INVALID | Le fichier d’input n’est pas un zip valide. |
| 50 | CODE\_GENERIC\_EXCEPTION | Une exception est intervenue côté serveur (Nakala), lors du traitement du fichier, sans plus d’indications. |
| 51 | CODE\_INVALID\_ZIP\_FILE\_SERVER | Le fichier de données est un ZIP invalide et le dézippage sur le serveur a échoué. |
| 52 | CODE\_INVALID\_INPUT\_PACKET | Il y a un problème structurel dans le paquet ZIP (descripteur manquant, etc.) |
| 53 | CODE\_INVALID\_METADATA\_DESCRIPTOR | Le descripteur de métadonnées est invalide. |
| 54 | CODE\_FACILE\_VALIDATION\_FAILED | La validation FACILE aéchouée |
| 55 | CODE\_ACCESS\_RIGHTS\_EXCEPTION | Il y a un problème de droits d'accès à une donnée ou à une collection |
| 56 | CODE\_DUPLICATE\_FILE | Le fichier de données existe déjà dans Nakala (vérification sur la base du checksum) |
| 57 | CODE\_UNKNOWN\_REFERENCE | Une référence à un handle dans la base Nakala n'a pas été trouvée (par exemple le handle d'une collection, ou de la collection parente) |
| 58 | CODE\_UNKNOWN\_UPDATED\_OBJECT | Le handle de la donnée à modifier n'a pas été trouvé dans la base. |
| 59 | CODE\_DUPLICATE\_COLLECTION | Une collection avec le même nom existe déjà |
| 99 | CODE\_SYSTEM\_ERROR | Une erreur grave (type manque de mémoire ou manque d’espace disque), a priori pas spécifiquement liée au paquet, est intervenue côté serveur. Contactez un administrateur Nakala. |

# Annexe : script utilitaire “paquet-create.sh”

Une fois dézippé, le batch se lance avec “start.sh” ou “start.bat”. Cependant un autre script utilitaire est inclus : “paquet-create.sh”. Ce script permet d’envoyer un seul paquet que l’on passe en paramètre du script, et gère automatiquement la création des répertoires d’input et d’output temporaires.

Pour utiliser ce script :

* Enregistrer votre clé d’API dans un fichier « nakal-key.txt » (comme pour le lancement classique du batch) ;
* Editez le fichier « paquet-create.sh » et ajustez dans les premières lignes du fichier l’email de votre compte Nakala ;
* Préparez le paquet à envoyer et copiez-le dans le répertoire du batch (là où se trouve le script paquet-create.sh ainsi que nakala-console.jar) ; ne copiez pas ce paquet dans le répertoire d’input, c’est le script qui s’en chargera ;
* Lancez le script avec le synopsys suivant :

./paquet-create.sh monPaquet.zip

* Le résultat et le rapport de batch pour l’envoi de cet unique paquet sont recopiés dans le répertoire du batch ;