Introduction

aux

API Vitam

Juillet 2016

Date	Version
18 juillet 2016	1.0

État du document

◯ En projet ◯ Vérifié ◉ Validé

Maîtrise du document

Responsabilité	Nom	Entité	Date
Rédaction	FB	Équipe noyau Vitam	
Vérification	FD, EL, EC	Équipe noyau Vitam	
Validation	JSL	Équipe noyau Vitam	

Suivi des modifications

Version	Date	Auteur	Modifications
avant 1.0			Versions de travail
1.0	18/07/2016	FB	Finalisation

Table des matières

1 Introduction

Ce document a pour but de présenter les API alpha qui seront utilisées aux interfaces entre le SAE Vitam et les applications front office connectées.

Après une présentation des fonctionnalités communes donnant quelques règles générales, ce document présente les API d'entrées permettant d'alimenter le SAE de nouvelles données puis les API d'accès permettant d'effectuer des recherches, de récupérer des données et d'effectuer des mises à jour de métadonnées et enfin les API des journaux..

Il montre ensuite trois sujets, l'entrée en mode programmatique, le mode d'authentification et les principes de pagination, qui sont en cours d'étude dans l'équipe Vitam où nous attendons vos commentaires et contributions.

Le document se termine par la liste des documents RAML disponible et par un descriptif des Mocks livrés avec ce document et ces limitations.

2 Historique

- 18/07/2016 : V0.alpha - publication des API alpha pour les fonctions d'entrées, d'accès et de journalisation

3 Fonctionnalités

Ce chapitre décrit les fonctionnalités présentes dans cette publication du Service d'Archivage Électronique (SAE) Vitam. L'objet de ce chapitre n'est pas de remplacer la documentation utilisateur, technique ou fonctionnelle, ni les documentations d'exploitation. Elle a pour cible les utilisateurs de l'API Vitam pour expliquer le contexte de celle-ci.

3.1 Fonctionnalités communes

3.1.1 Identifiant de corrélation

Vitam étant un service REST, il est "Stateless". Il ne dispose donc pas de notion de session en propre.
Cependant chaque requête retourne un identifiant de requête "**X-Request-Id**" qui est tracé dans les logs et journaux du SAE et permet donc de faire une corrélation avec les événements de l'application Front-Office cliente si celle-ci enregistre elle-aussi cet identifiant.

Pour cependant permettre le suivi d'une session d’un utilisateur connecté sur un Front-Office, il est proposé que l'application Front-Office puisse passer en paramètre dans le Header l'argument "**X-Application-Id**" correspondant à un identifiant de session de l'utilisateur connecté. Cet identifiant DOIT être non signifiant car il sera lui aussi dans les logs et les journaux de Vitam. Il sera en effet inclus dans chaque réponse de Vitam s'il est exprimé dans la requête correspondante.

Cet identifiant externe de session permet de retracer l'activité d'un utilisateur grâce d'une part au regroupement de l'ensemble des actions dans Vitam au travers de cet identifiant, et d'autre part grâce aux logs de l'application Front-Office utilisant ce même identifiant de session.

3.1.2 Domain Specific Language

Vitam utilise un langage spécifique de requête qui permet de prendre en compte les notions suivantes :

- La gestion de l'arborescence dans la structuration des données (unités d'archives)
- La possibilité d'effectuer des recherches « plein texte » sur les métadonnées
  - Pour rappel il n'est pas prévu dans Vitam de pouvoir rechercher sur le contenu des objets binaires. Pour obtenir une forme de recherche sur le contenu, il est proposé d'indexer dans les métadonnées une partie du contenu des documents lors de la confection des entrées (SIP) versées dans Vitam (via un champ « Abstract » par exemple).
- La possibilité d'offrir des usages étendus en fonction des besoins d'applications métiers Front-Office (comme l'usage de facettes, de pagination).

Ce langage est principalement utilisé pour la recherche sur les Unités d'archives et les Objets numériques (toujours sur les métadonnées). Il est également étendu sous une forme légèrement plus simple pour l'ensemble des collections (notamment les journaux).

Sa définition peut être trouvée dans la documentation des API (chapitre « DSL Vitam »).
Des outils d'accélération de développement sont également fournis (en Java) pour faciliter l'écriture des requêtes (partie Body) (package metadata-builder).

3.1.3 Modèle REST

Les URL sont découpées de la façon suivante :

- Protocole: https
- FQDN : exemple api.vitam.fr avec éventuellement un port spécifique
- Base : <nom du service>/<version>
- Ressource : le nom d'une collection
- L'URL peut contenir d'autres éléments (item, sous-collections)

  Exemple: https://api.vitam.fr/access/v1/units/id

Les méthodes utilisées :

- GET: pour l'équivalent de "Select" (possibilité d'utiliser POST avec X-Http-Method-Override: GET dans le Header)
- POST: pour l'équivalent de "Insert"
- PUT: pour l'équivalent de "Update"
- DELETE: pour l'équivalent de "Delete" (possibilité d'utiliser POST avec X-Http-Method-Override: DELETE dans le Header)
- HEAD: pour l'équivalent du "Test d'existence"
- OPTIONS: pour l'équivalent de "Lister les commandes disponibles"
  - Cette fonctionnalité n'est pour le moment indiquée qu'à titre prévisionnel.

Les codes retours HTTP standards utilisés sont :

- 200: Sur des opérations de GET, PUT, DELETE, OPTIONS
- 201: Sur l'opération POST (sans X-Http-Method-Override)
- 202: Pour les réponses asynchrones
- 204: Pour des réponses sans contenu (type HEAD sans options)
- 206: Pour des réponses partielles

Les codes d'erreurs HTTP standards utilisés sont :

- 400: Requête mal formulée

- 401: Requête non autorisée
- 403: Accès refusé
- 404: Ressource non trouvée
- 409: Requête en conflit
- 412: Des préconditions ne sont pas respectées
- 413: La requête dépasse les capacités du service
- 415: Le Media Type demandé n'est pas supporté
- 417: Résultat attendu non satisfaisant
- 501: Le service n'est pas implémenté
- 503: Service indisponible

3.2 Entrée (Ingest)

La fonctionnalité proposée dans cette API est celle de pouvoir réaliser une entrée (processus « ingest » dans l'Open Archival Information System (OAIS)).
La version proposée utilise le modèle d'un versement compatible avec le Standard d'Échange de Données pour l'Archivage (SEDA) (basée sur la norme MEDONA NFZ44-022).

Le principe est de transférer un ZIP (ou un TAR) contenant un « Submission Information Package » (SIP) dont la structure est la suivante :

- manifest.xml : le fichier répondant au standard SEDA du message « ArchiveTransfer »
- content/ : le répertoire contenant l'ensemble des fichiers binaires faisant l'objet de l'entrée

Les recommandations pour le SIP sont les suivantes :

- Les noms d'origine des fichiers binaires étant dans le manifeste, ils peuvent être renommés dans le répertoire « content » de manière unique sans rapport avec leur nom d'origine
  - Par exemple, le nom pourrait être l'empreinte du fichier + son extension d'origine
- Les fichiers binaires ne sont pas inclus dans le manifeste (non embarqués en mode Base64 dans le fichier XML) mais dans le répertoire « content ».
- La compression n'est pas une obligation (le ZIP peut être construit avec un niveau de compression à 0).

La procédure est asynchrone :

- Lorsque le fichier SIP est totalement transféré, le SAE renvoie un résultat « entrée acceptée » (202) ainsi que l'identifiant de la demande d'entrée.
- Cet identifiant peut être utilisé pour demander de manière asynchrone où en est le traitement.
- Lorsque le traitement d'entrée est terminé, le SAE retourne un résultat « entrée terminée » (200 ou un code d'erreur si l'entrée est en erreur) ainsi que le lien pour accéder au résultat détaillé (rapport d'entrée qui contient à nouveau le statut de l'opération).
- Le rapport d'entrée contient les informations compatibles avec le message SEDA « ArchiveTransferReply ».

3.2.1 Mécanismes asynchrones

Dans le cas d'une opération asynchrone, deux options sont possibles :

3.2.1.1 Mode Pooling

Dans le mode pooling, le client est responsable de requêter de manière répétée mais raisonnée l'URI de vérification du statut.

Le principe est le suivant :

- Création de l'opération à effectuer
  - Exemple: POST /ingests et retourne 202 ainsi que l'identifiant #id de l'opération
- Pooling sur l'opération demandée
  - Exemple: GET /ingests/#id et retourne 202 ainsi que l'identifiant #id tant que l'opération est non terminée
  - Intervalle recommandé : pas moins que la minute
- Fin de pooling sur l'opération demandée
  - Exemple: GET /ingests/#id et retourne 200 ainsi que le résultat

3.2.1.2 Mode Callback

Dans le mode Callback, le client soumet une création d'opération et simultanément propose une URI de Callback sur laquelle Vitam rappellera le client pour lui indiquer que cette opération est terminée.

Le principe est le suivant :

- Création de l'opération à effectuer avec l'URI de Callback
  - Exemple: POST /ingests ainsi que dans le Header la variable « **X-Callback:** https://uri?id={id}&status={status} » et retourne 202 ainsi que l'identifiant #id de l'opération et le Header « X-Callback » confirmé
- A la fin de l'opération, Vitam rappelle le client avec l'URI de Callback
  - Exemple: HTTPS GET /uri?id=idop&status=OK
- Le client rappelle alors Vitam pour obtenir l'information
  - Exemple: GET /ingests/#id et retourne 200 ainsi que le résultat

3.3 Accès (Access)

L'objet de cette API est d'offrir le service de recherche, de consultation et de mises à jour des Unités d'archives (Units) ainsi que le service de recherche et de consultation des Objets d'archives (Objects).

Cette API est globalement reproduite dans tous les autres points d'accès lorsque l'accès à des Units et Objects est nécessaire. Les fonctionnalités offertes peuvent par contre varier (droit en modification, effacement...) selon le contexte.

3.3.1 Units

L'Unité d'archives (Unit) est le point d'entrée pour toutes les descriptions d'archives. Celles-ci contiennent les métadonnées de description et les métadonnées archivistiques (métadonnées de gestion).

Une Unité peut être soit un simple dossier (comme dans un plan de classement), soit la description d'un item. Les Unités portent l'arborescence d'un plan de classement. Ce plan de classement peut être multiple :

- Plusieurs racines (*Unit* de plus haut niveau)
- Plusieurs parents (un dossier peut être rattaché à plusieurs dossiers)

Il s'agit d'une représentation dite de "graphe dirigé sans cycle" (DAG en Anglais pour "Directed Acyclic Graph").
NB : il est interdit de créer une boucle dans l'arborescence (Unit1 → Unit2 → Unit3 → Unit1).

Pour le standard SEDA, il est équivalent à l'« ArchiveUnit », notamment pour les parties « Content » et « Management ». Pour l'Isad(G) / EAD, il est équivalent à « Description Unit ».

Un seul groupe d'Objets d'archives (Object) est attaché à une Unité. Cela signifie que si une Unité devait avoir plus d'un groupe d'Objets d'archives attaché, il faudrait spécifier des sous-Unités à l'Unité principale, chaque sous-Unité n'ayant qu'un groupe d'Objets d'archives attaché. Un même groupe d'Objets d'archives peut par contre être attaché à de multiples Unités.

Aucun effacement, ni aucune mise à jour complète ne sont autorisés (seules les mises à jour partielles sont autorisées via la commande PUT).

3.3.2 Objects

L'Objet d'archives (Object) est le point d'entrée pour toutes les archives binaires mais également les non binaires (comme une référence à des objet d'archives physiques ou externes au système). Elles contiennent les métadonnées techniques. Il est constitué de plusieurs usages et versions d'usages du même contenu. C'est dans ce sens qu'il est aussi appelé un Groupe d'Objets.

Un Groupe d'Objets peut être constitué de plusieurs versions (sous-Objets) pour différencier des usages comme version de conservation, version de diffusion...

Pour le standard SEDA, il est équivalent à un « DataObjectGroup ». Pour l'EAD, il est équivalent à un « Digital Archive Object Group or Set ».

Chaque Groupe d'Objets doit être attaché à au moins un parent Unit.

Seul l'accès est autorisé (via les commandes GET/HEAD), aucune modification n'est possible.

Pour le modèle SEDA, chaque usage/version est équivalent à un « DataObject » (binaire avec « BinaryDataObject » ou physique avec « PhysicalDataObject »). Pour l'EAD, il est équivalent à un « Digital Archive Object ».

Chaque Objet n'est lié qu'à un seul Groupe. Chaque Objet a un qualifier (spécifiable via « #qualifiers » dans le DSL et « X-Qualifier » en Header) :

- PhysicalMaster : il s'agit de l'original papier ou physique s'il existe
- BinaryMaster : il s'agit de l'original numérique s'il existe
- Dissemination : il s'agit d'une version adaptée à la diffusion via le réseau et l'usage dans des navigateurs
- Thumbnail : il s'agit d'une version adaptée à la diffusion dont la taille et la qualité correspondent à une vignette (utile pour l'affichage d'une liste)
- TextContent : il s'agit d'une version ne contenant que le texte du document, sans la mise en forme (son usage est en prévision du futur, par exemple pour des opérations d'analyse sémantique)
- All : il s'agit, en consultation et en cas de ZIP ou TAR, d'un accès à l'ensemble des usages et versions.

Ces « qualifiers » peuvent être utilisés dans une requête de consultation (pour accéder en lecture) ou une requête de check d'existence (pour un usage particulier).
Le qualifier « All » est ajouté pour permettre l'accès à l'ensemble des usages, notamment dans le cadre du téléchargement d'un « Dissemination Information Package » (DIP) complet.

3.4 Journaux (Logbook)

L'objet de cette API est d'offrir le service de recherche et de consultation des journaux du SAE :

- Journal des opérations (à ce stade, journal des opérations d'entrée et des opérations de modification)
  - Ainsi une entrée donne lieu à l'enregistrement des étapes par lesquelles cette entrée est passée avant son état final, y compris en cas d'échec.
  - Le journal des opérations ne constitue pas le journal des entrées au sens archivistique mais il y contribue. Un autre service fournira les informations du journal des entrées.
  - Le journal des opérations assure la traçabilité des événements à gros grains pour le SAE, c'est à dire une consolidation des grandes étapes effectuées durant l'opération mais pas de manières fines sur chacun des objets ou unités d'archives concernées.
- Journal des cycles de vie (journal pour les Unités d'archives et pour les Objets d'archives) en lien avec les opérations auxquelles ces archives ont contribué
  - Chaque Unité d'archives ou chaque Groupe d'Objets dispose de son propre journal du cycle de vie qui trace toutes les actions qui ont pris place pour cet item.
  - Il assure la traçabilité des événements de grains fins sur les archives, toujours en lien avec une opération et vient donc compléter le journal des opérations.

Par exemple, une entrée de 1000 Unités d'archives et 100 Objets d'archives donnera lieu à :

- 1 entrée dans le journal des opérations pour l'entrée correspondante, avec toutes les étapes par lesquelles le processus est passé.
- 1000 journaux de cycles de vie de Unit (1 par Unité d'archives) et 100 journaux de cycles de vie d'Object (1 par Groupe d'Objets d'archives).

4 Sujets à l'étude

Ce chapitre liste les études en cours, pour lesquelles nous demandons également des avis externes très en avance de phase.

4.1 Entrée en mode programmatique

Si une entrée en mode SEDA est le standard, afin de faciliter la prise en compte des archives numériques, et en particulier dans le cadre des archives intermédiaires et a fortiori courantes, il est proposé une version n'imposant pas la création d'un fichier manifest.xml mais fonctionnellement identique.
Le principe proposé serait le suivant :

1. Création (POST) d'une transaction d'entrée (**/ingests**) avec dans le body les informations en JSON du contrat d'entrée associé et quelques informations dites d' « en-tête ».
    1. Le SAE retourne une réponse « Accepted » (202) avec l'identifiant de la transaction
2. Sélection (GET) de l'Unité d'archives parente des Unités d'archives qui vont être versées (**/ingests/idTransaction/units**)
    1. Le comportement est similaire à l'API de recherche (/units)
    2. Si l'identifiant est déjà connu, cette étape peut être ignorée.
3. Pour une Unité d'archives parente existante, création (POST) des sous-Unités d'archives avec l'Objet numérique associé (**/ingests/idTransaction/units/idParent/units**)
    1. L'Objet numérique est optionnel. Ce mode opératoire n'autorise pas de multiples versions ou usages (qualifiers). La version transmise est donc considérée comme un « BinaryMaster ».
    2. Les informations transmises sont dans le corps de la requête (multiple arguments) :
        1. La description de l'Unité d'archives et les métadonnées de gestion au format JSON
        2. Dans le cas d'un Objet numérique, chaque Objet est transmis avec son nom de fichier (filename) ainsi que son mime-type (Content-Type) ; il est recommandé a priori d'utiliser le mode binaire (Content-Transfer-Encoding : binary)
    3. Le SAE retourne une réponse « Accepted » (202) pour l'ajout de cette Unité d'archives et de son Objet numérique associé
    4. L'application Front-Office peut répéter les étapes 2 et 3.
4. Une fois l'ensemble des Unités d'archives transmises, l'application Front-Office clôture la transaction via une mise à jour (PUT) de la transaction (**/ingests/idTransaction/commit**).
    1. Le SAE retourne une réponse « Accepted » (202) et procède aux opérations liées à l'entrée
    2. Le processus reprend alors le cours normal, comme pour une entrée standard d'un SIP avec un manifest.xml (SEDA) (transaction asynchrone).

Exemple :

```
POST /ingests { MessageIdentifier : NomDuMessage, ArchivalAgreement : IdentifiantContratEntrée, … }
        202 : { #id : idTransaction, … }
(optionnel) GET /ingests/idTransaction/units { DSL provoquant une recherche d'un Unit }
        200 : { …, $results : [ { #id : idParent } ] }
POST /ingests/idTransaction/units/idParent/units { metadata + management } + ObjetBinaire
        202 : { #id : idTransaction, … }
PUT     /ingests/idTransaction/commit
        202 : { #id : idTransaction, … }
GET /ingests/idTransaction
        200 : { résultat final }
```

Cette proposition tend à faciliter la réalisation d'entrées rapidement et plus simplement pour des entrées récurrentes.

4.2 Authentification

L'authentification dans Vitam concerne l'application Front-Office qui se connecte à ses API. Cette authentification s'effectue en trois temps :

- Un premier composant authentifie la nouvelle connexion
  - La première implémentation sera basée sur une authentification du certificat client dans la connexion TLS
- Le premier composant passe au service REST la variable Header "**X-Identity**" contenant l'identifiant de l'application Front-Office
- Le service REST, sur la base de cette authentification, s'assure que l'application Front-Office a bien l'habilitation nécessaire pour effectuer la requête exprimée.

4.3 Pagination

Vitam ne dispose pas de notion de session en raison de son implémentation « Stateless ». Néanmoins, pour des raisons d'optimisation sur des requêtes où le nombre de résultats serait important, il est proposé une option tendant à améliorer les performances : **X-Cursor** et **X-Cursor-Id**.

De manière standard, il est possible de paginer les résultats en utilisant le DSL avec les arguments suivants dans la requête : (pour GET uniquement)

- **$limit** : le nombre maximum d'items retournés par la base de données (limité à 1 000 par défaut, maximum à 100 000, minimum à 1)
- **$per_page** : le nombre maximum d'items retournés par page (limité à 100 par défaut, maximum à 100, minimum à 1)
- **$offset** : la position de démarrage dans la liste retournée (positionné à 0 par défaut, maximum à 100 000)

En raison du principe « Stateless », l'utilisation de cette seule manière de requêter (c'est-à-dire en générant de nouvelles requêtes en manipulant le $offset manuellement) implique que la requête est rejouée sur la base de données et conduit donc à des performances réduites.

Afin d'optimiser les performances, il est proposé d'ajouter de manière optionnelle dans le Header lors de la première requête le champ suivant : **X-Cursor: true**
Si la requête nécessite une pagination (plus d'une page de réponses possible), le SAE répondra alors la première page (dans le Body) et dans le Header :

- **X-Cursor-Id:** id (identifiant du curseur)
- **X-Cursor-Timeout:** datetime (date limite de validité du curseur)

Le client peut alors demander les pages suivantes en envoyant simplement une requête GET avec un Body vide et dans le Header : **X-Cursor-Id:** id.

5 Documentation RAML

Ce chapitre liste les documentations RAML disponibles.

5.1 Fichier raml/ingest.raml

L'objet de cette API est d'offrir le service d'entrées des archives.

5.2 Fichier raml/access.raml

L'objet de cette API est d'offrir le service de recherche, de consultation et de mises à jour des Unités d'archives ainsi que de le service de recherche et de consultation des Objets d'archives.

5.3 Fichier raml/logbook.raml

L'objet de cette API est d'offrir le service de recherche et de consultation des journaux du SAE :

- Journal des opérations (à ce stade, journal des entrées et des modifications)
- Journal des cycles de vie (journal pour les Unités d'archives et pour les Objets d'archives) en lien avec les opérations auxquelles ces archives ont contribué.

6 Implémentation MOCK

Ce chapitre note les limitations connues de l'implémentation en mode MOCK (bouchon) des API externes du SAE Vitam.

6.1 Readme.txt

Contenu du fichier Readme.txt présent dans l'archive ZIP :

6.1.1 Pré-requis

- Un environnement d'exécution Java 8 (Java Runtime Environment) doit être installé et configuré dans la variable système PATH.
- Un OS Windows ou Linux

6.1.2 Description

L'archive contient :

- la documentation html des API Vitam
- la documentation RAML des API Vitam
- un serveur contenant les bouchons des API Vitam et l'outil RAML Console (https://github.com/mulesoft/api-console)
- les Javadoc de quelques modules Vitam

6.1.3 Utilisation

6.1.3.1 Exécution du serveur

Pour exécuter le serveur, lancer l'exécutable correspondant à votre OS :

- run.sh (Linux)
- run.bat (Windows)

Pour s'assurer que le serveur s'est bien démarré, vérifier les pages suivantes :

- la RAML Console
  - URL : http://localhost:8082/doc/api-console.html
  - résultat attendu : l'outil RAML Console est chargé
- les bouchons des API Vitam :
  - URL : http://localhost:8082/external/v1/status
  - résultat attendu : une réponse 200 est envoyée

6.1.3.2 Utilisation de la RAML Console

Les fichiers RAML de documentation de l'API Vitam sont chargés dans la RAML Console en les renseignant dans le champ "INITIALIZE FROM THE URL OF A RAML FILE" l'un des chemins suivants :

- raml/access.raml
- raml/ingest.raml
- raml/logbook.raml

Les fichiers RAML de documentation de l'API Vitam sont dans le dossier console/raml. Toute modification d'un de ces fichiers nécessite de redémarrage du serveur.

6.1.3.3 Utilisation de la fonctionnalité TRY-IT de la RAML Console

Une fonctionalité TRY-IT permettant de tester les API est fournie par la RAML Console. Pour des raisons de manque de stabilité de la fonctionnalité offerte par cet outil externe au Programme Vitam, elle est désactivée par défaut.

Elle reste accessible sur l'URL http://localhost:8082/doc/api-console-try-it.html

Pour utiliser la fonctionnalité TRY-IT de la RAML Console, il est préférable d'utiliser les chemins suivants :

- raml/access-nosecurity.raml
- raml/ingest-nosecurity.raml
- raml/logbook-nosecurity.raml

La fonctionnalité TRY-IT de la RAML Console utilise par défaut les bouchons fournis par le serveur. Pour utiliser un autre serveur, il faut modifier la valeur de l'attribut baseUri des fichiers RAML utilisés.

6.1.3.4 Utilisation des bouchons de l'API VITAM

Les bouchons sont déployés sur l'URL http://localhost:8082/external/v1

6.1.3.5 Documentation html de l'API VITAM

La documentation html de l'API VITAM se trouve dans le dossier html et peut être consultée directement en se rendant à l'adresse http://localhost:8082/doc/html

Elle inclut les Javadoc des modules suivants :

- Query Builder (builder DSL pour Access)
- Access Client (dont la factory)
- Ingest Client (dont la factory)
- Logbook Client (dont la factory)

Ainsi que ce présent fichier et la licence globale du logiciel Vitam.

6.1.4 Configuration

6.1.4.1 Changer le port HTTP du serveur

La valeur du port HTTP sur lequel s'exécute le serveur est 8082 par défaut.

Pour modifier cette valeur, il faut l'ajouter en argument de l'exécutable :

- run.sh 8088 (Linux)
- run.bat 8088 (Windows)

6.1.4.2 Configuration des logs

Les logs utilisent la bibliothèque LOGBack (http://logback.qos.ch/).

Le fichier de configuration des logs est : config/logback.xml. Toute modification de ce fichier nécessite le redémarrage du serveur.

6.1.5 Exemples de codes clients

Le répertoire « samples » contient des exemples de codes Java (avec un modèle de pom.xml) pour illustrer l'usage des API Java proposées :

- AccessSamples.java
- IngestSamples.java
- LogbookSamples.java

Il propose également des exemples de commandes Curl pour utiliser les API :

- access_curl.txt
- logbook_curl.txt

6.2 Fonctionnalités non supportées

- Le support SSL n'est pas actif.
- L'authentification n'est pas active. Il faut utiliser le mode « Basic Authentication » qui ne sera pas maintenu plus tard.
- Le DSL n'intègre pas encore les fonctionnalités suivantes :
  - le support des facettes
  - le support des requêtes géomatiques
  - le support « per_page »
  - la définition de tous les champs spécificiques Vitam (préfixés par '#')
  - le support pour les requêtes DSL autres que pour Access (le builder pour Ingest et Logbook n'est pas finalisé mais il reprend en très grande partie les mêmes fonctionnalités que le DSL pour Access)
  - des évolutions sont possibles sur ce langage dans les évolutions d'ici la version Beta

6.3 Simulations

- Les requêtes DSL ne sont pas validées ni analysées.
-
- Le SIP est ignoré lors de l'entrée.

- Seul le mode « Accept : application/json » est supporté (sauf indication contraire).
- Les réponses sont statiques, quelles que soient les requêtes et par défaut en mode JSON.
- Pour le module « Entrée », la réponse finale de /ingests/#id en XML et donc le mode « Accept : application/xml » est utilisé.

6.4 Accélérateurs de développement

- Des outils d'accélération de développement sont fournis (en Java) pour faciliter l'écriture des requêtes (partie Body) sur la base d'un module toujours en cours de développement (Query builder).
- Des outils d'accélération de développement sont fournis (en Java) pour faciliter l'accès aux API en HTTP (IngestClientFactory, AccessClientFactory, LogbookClientFactory).