VITAM - Manuel Intégraption Applicative

Version 0.20.0

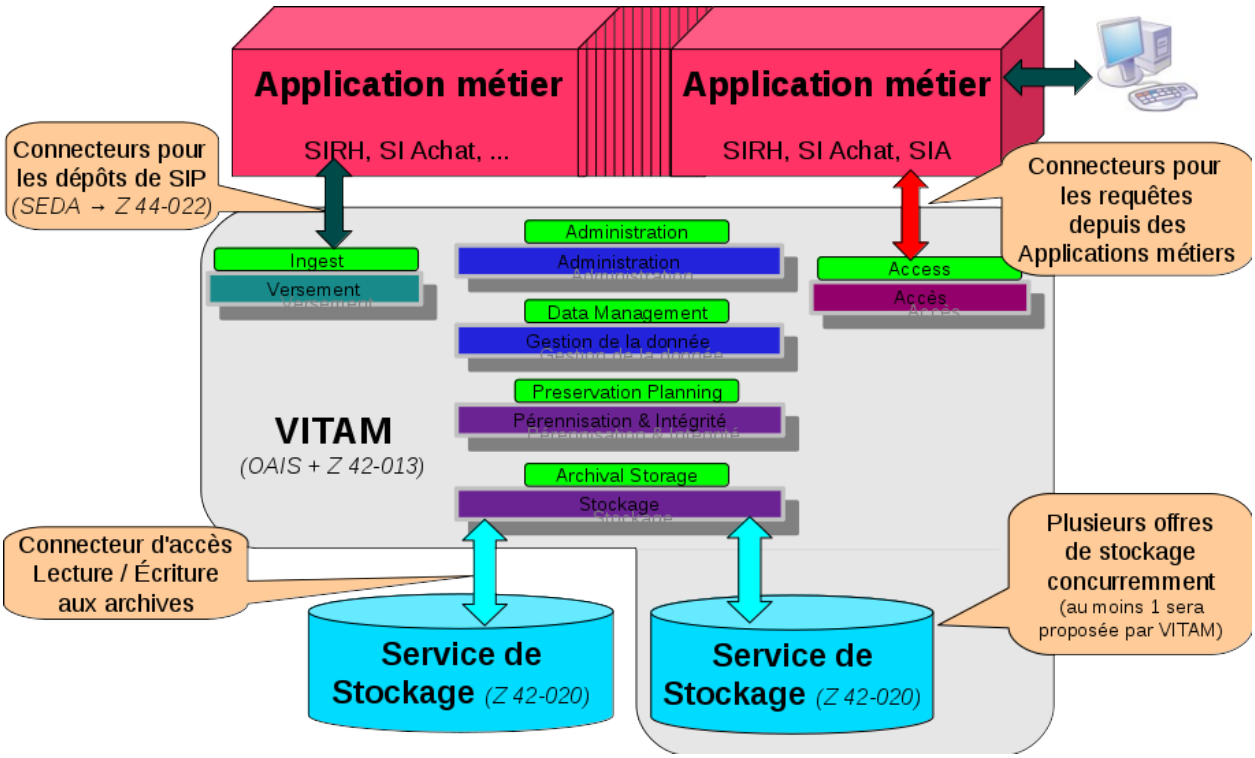
VITAM

juil. 21, 2017

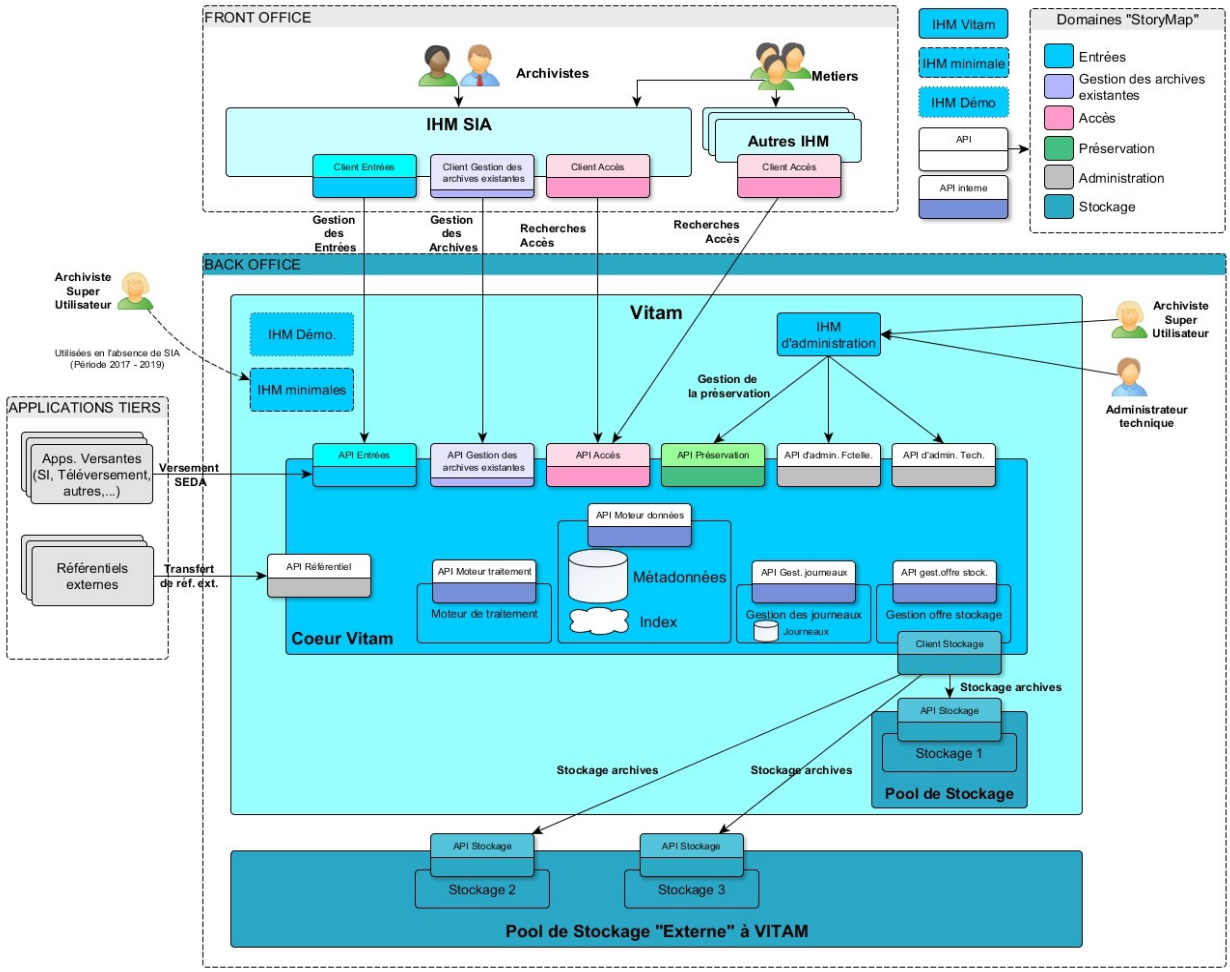
Table des matières

VITAM

1.1 Architecture générale



1.2 Architecture des flux



CHAPITRE 2

API

2.1 Documentation

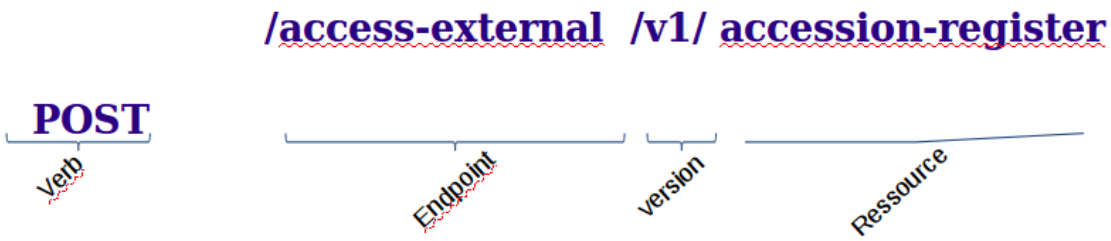
2.1.1 Services

- ingest-external : opérations d'entrées
- access-external : accès et journaux d'opérations
- admin-external : gestion du référentiel et opérations d'administration

2.1.2 Quelques Ressources

- `/ingest-external/v1/ingests`
- `/admin-external/v1/formats`
- `/access-external/v1/units`

2.1.3 Formation générale des API externes



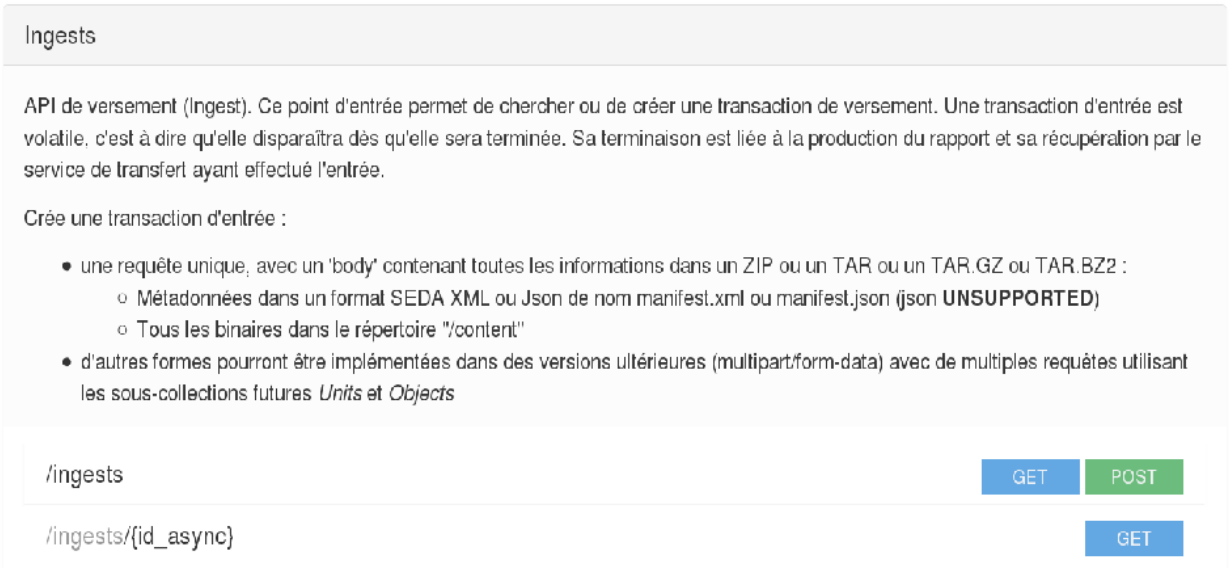
2.1.4 RAML

- Ressource documentaire : raml/externe/introduction.html [1]
- Ressource documentaire : raml/externe/ingest.html [2]

1. http://www.programmevitam.fr/ressources/Doc0.20.0/raml/externe/introduction.html
2. http://www.programmevitam.fr/ressources/Doc0.20.0/raml/externe/ingest.html

- Ressource documentaire : raml/externe/access.html [3]
- Ressource documentaire : raml/externe/functional-administration.html [4]
- Ressource documentaire : raml/externe/logbook.html [5]

exemple d'interface du RAML



2.1.5 Javadoc

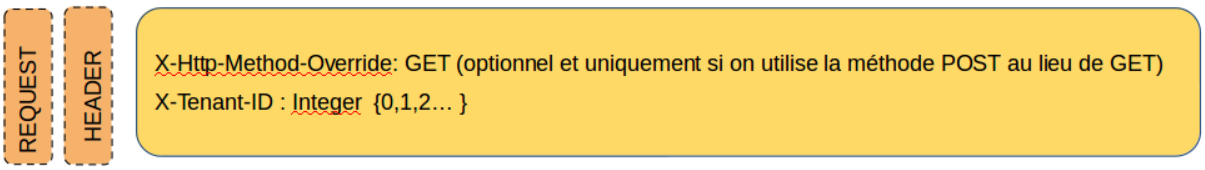
- Ressource documentaire : javadoc [6]

Notamment pour les packages des clients suivants :

- Ingest External Client : `fr.gouv.vitam.ingest.external.client` ;
- Access External Client : `fr.gouv.vitam.access.external.client` ;
  - Plus tard ce package sera découpé en trois parties :
    - `AccessExternalClient`
    - `AdminExternalClient`
    - `LogbookExternalClient`

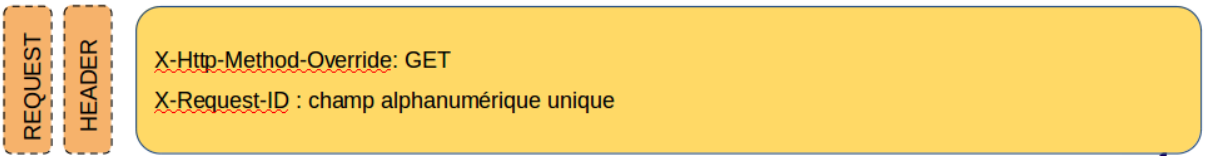
2.1.6 Requêtes et réponses

Les requêtes HTTP VITAM ont en commun les attributs suivants :



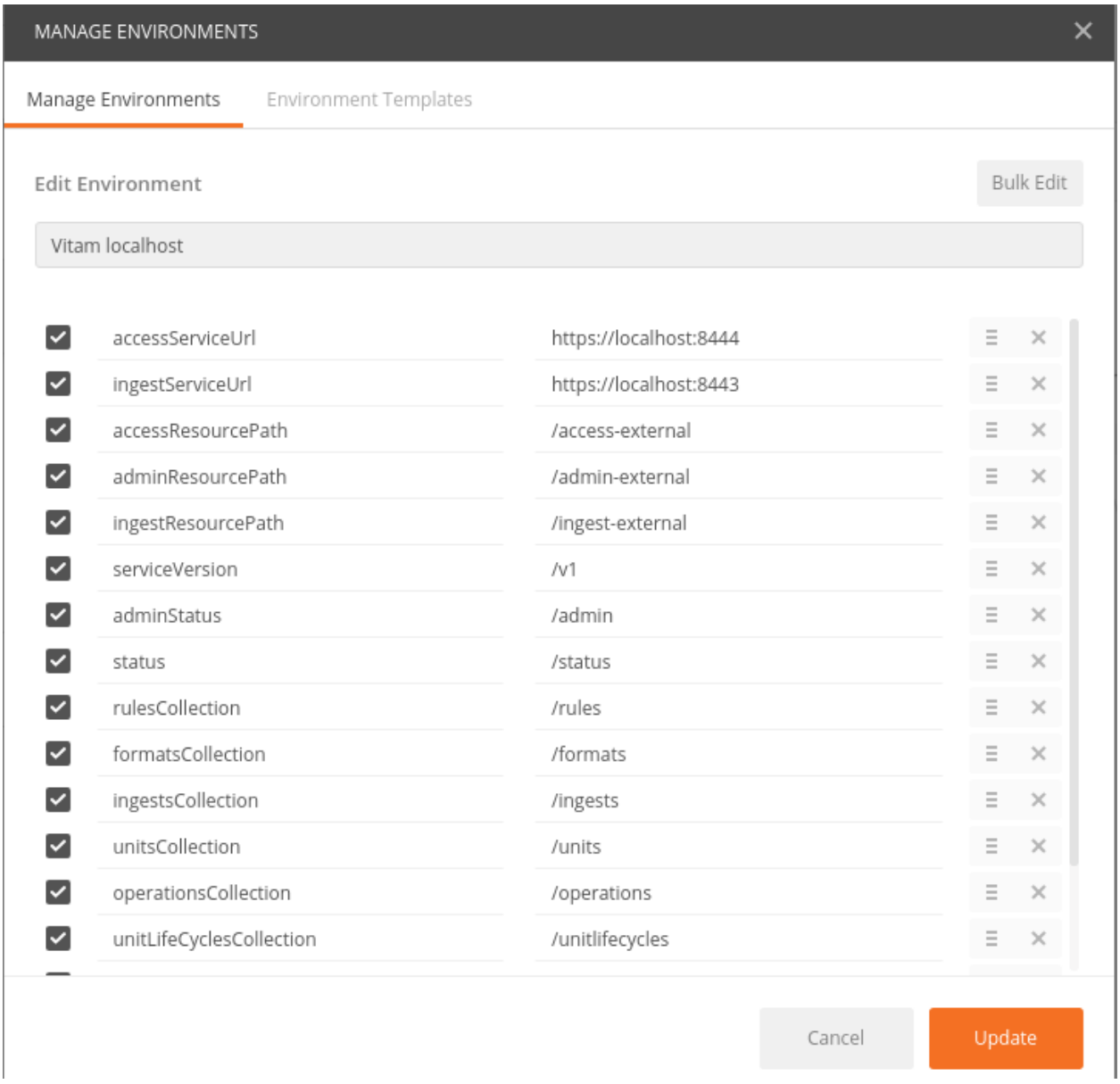
3. http://www.programmevitam.fr/ressources/Doc0.20.0/raml/externe/access.html
4. http://www.programmevitam.fr/ressources/Doc0.20.0/raml/externe/functional-administration.html
5. http://www.programmevitam.fr/ressources/Doc0.20.0/raml/externe/logbook.html
6. http://www.programmevitam.fr/ressources/Doc0.20.0/javadoc

Les réponses HTTP de VITAM ont en commun les attributs suivants :



2.1.7 Utilisation de PostMan

Possibilité de gestion des collections



Exemples d'élements à renseigner pour une requête, ici `POST /ingest-external/v1/ingests` :

2.2 Ingest

Exemple d'implémentation Java :

`/ingest-external/v1/ingests`

```
File sipFile = .....
/*  Client auto closeable */
try (IngestExternalClient client = IngestExternalClientFactory.getInstance().getClient())  {
        finalResponse = client.upload(new FileInputStream(sipFile), tenantId);
        final String guid = finalResponse.getHeaderString(GlobalDataRest.X_REQUEST_ID);
        final InputStream inputStream = (InputStream) finalResponse.getEntity();
        ....
        finalResponse.close();

} ...
```

`/ingest-external/v1/ingests/{objectId}/{type}`

type = {REPORTS, MANIFESTS}

```
try (IngestExternalClient client = IngestExternalClientFactory.getInstance().getClient()) {
   Response response = client.downloadObjectAsync(objectId, IngestCollection.REPORTS, tenantId);
...
response.close();'
}
```

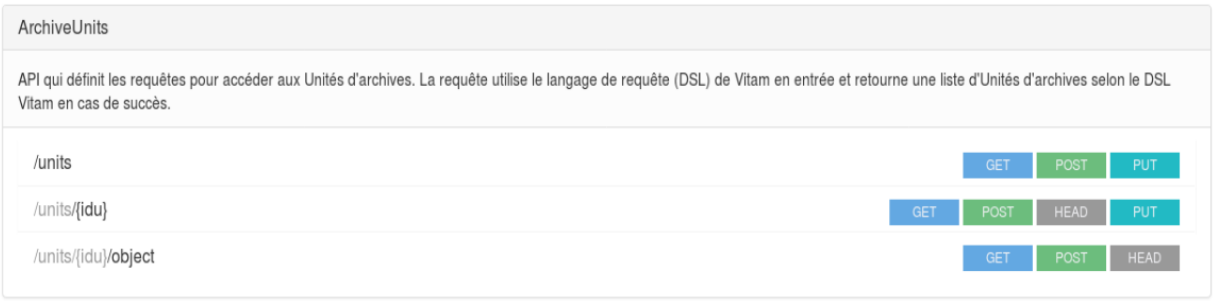
2.3 Access

RAML : Ressource documentaire : raml/access/introduction.html [7] ; API qui définit les requêtes pour accéder aux Unités d'archives.

La requête utilise le langage de requête (DSL) de Vitam en entrée et retourne une liste d'Unités d'archives selon le DSL Vitam en cas de succès.

- `/units`

7. http://www.programmevitam.fr/ressources/Doc0.20.0/raml/access/introduction.html


ArchiveUnits
API qui définit les requêtes pour accéder aux Unités d'archives. La requête utilise le langage de requête (DSL) de Vitam en entrée et retourne une liste d'Unités d'archives selon le DSL Vitam en cas de succès.
/units
GET POST PUT
/units/{idu}
GET POST HEAD PUT
/units/{idu}/object
GET POST HEAD

- `/objects`


Objects
API qui définit les requêtes pour accéder aux Objets d'archives. La requête utilise le langage de requête (DSL) de Vitam en entrée et retourne une liste d'Objets d'archives selon le DSL Vitam en cas de succès.
/objects
GET POST
/objects/{ido}
GET POST HEAD

CHAPITRE 3

Conventions REST Générales

Ici sont présentées des conventions dans le cadre des API Vitam.

3.1 Modèle REST

Les URL sont découpées de la façon suivantes :

- protocole : https
- FQDN : exemple api.vitam.fr avec éventuellement un port spécifique
- Base : <nom du service>/<version>
- Resource : 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"

Les codes retours HTTP standards utilisés sont :

- 200 : Sur des opérations de GET, PUT, DELETE, HEAD, 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 ou des réponses en mode Warning (OK mais avec une alerte)

Les codes d'erreurs HTTP standards utilisés sont :

- 400 : Requête mal formulée
- 401 : Requête non autorisée
- 404 : Resource 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
- 500 : si une erreur interne est survenue dans le back-office (peut être un bug ou un effet de bord d'une mauvaise commande)
- 501 : Le service n'est pas implémenté

3.2 Modèle asynchrone

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

3.2.1 Mode Pooling

Dans le mode pooling, le client est responsable de requêter de manière répétée l'URI de vérification du statut, et ce de manière raisonnée (pas trop souvent).

Le principe est le suivant :

- Création de l'opération à effectuer
  - Exemple : POST /ingests et retourne 202 + X-Request-Id noté id
- Pooling sur l'opération demandée
  - Exemple : GET /operations/id et retourne 202 + X-Request-Id tant que non terminé - Intervalle recommandé : pas moins que la minute
- Fin de pooling sur l'opération demandée
  - Exemple : GET /operations/id et retourne 200 + le résultat

3.2.2 Mode Callback UNSUPPORTED

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 + dans le Header X-Callback : https://uri ?id={id}&status={status} et retourne 202 + #id + Header X-Callback confirmé
- A la fin de l'opération, Vitam rappelle le client avec l'URI de Callback
  - Exemple : GET /uri ?id=idop&status=OK
- Le client rappelle alors Vitam pour obtenir l'information
  - Exemple : GET /ingests/#id et retourne 200 + le résultat

3.2.3 Perspectives d'évolution

Dans le cas où l'accès au résulat final ne doit pas se faire sur l'URI /resources/id, il faudra ajouter une réponse 303.

3.3 Authentification

L'authentification dans Vitam authentifie 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
  - Note : il est possible au travers d'un ReverseProxy de passer le certificat client dans le Header
- Le premier composant passe au service REST la variable Header "X-Identity" contenant l'identifiant de l'application Front-Office.
  - Comme cette identification est actuellement interne, ce Header est actuellement non généré.
- Le service REST, sur la base de cette authentification, s'assure que l'application Front-Office ait bien l'habilitation nécessaire pour effectuer la requête exprimée.

3.4 Identifiant de corrélation

Vitam étant un service REST, il est "State Less". Il ne dispose donc pas de notion de session en propre. Cependant chaque requête retourne un identifiant de requête "**X-Request-Id**" qui est traçé 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.

UNSUPPORTED Considérant que cela peut rendre difficile le suivi d'une session 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 est inclus dans chaque réponse de Vitam si celui-ci est exprimé dans la requête correspondante. Grâce à cet identifiant externe de session, il est alors plus facile 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.

Afin de gérer plusieurs tenants, il est imposé (pour le moment) que l'application Front-Office puisse passer en paramètre dans le Header l'argument **X-Tenant-Id** correspondant au tenant sur lequel se baser pour exécuter la requête.

3.5 Pagination

Vitam ne dispose pas de notion de session en raison de son implémentation « State Less ». Néanmoins, pour des raisons d'optimisations 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.

3.5.1 Méthode standard

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

- **$limit** : le nombre maximum d'items retournés (limité à 1000 par défaut, maximum à 100000)
- **$per_page** : le nombre maximum des premiers items retournés (limité à 100 par défaut, maximum à 100) (**UNSUPPORTED**)
- **$offset** : la position de démarrage dans la liste retournée (positionné à 0 par défaut, maximum à 100000)

En raison du principe State-less, les requêtes suivantes (en manipulant notamment $offset) seront à nouveau exécutées, conduisant à des performances réduites.

3.5.2 Méthode optimisée UNSUPPORTED

Afin d'optimiser, il est proposé d'ajouter de manière optionnelle dans le Header lors de la première requête le champs 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.

CHAPITRE 4

DSL

4.1 Principes

- Dans le body - langage de requête
  - DSL VITAM
  - SQL : pas de plein texte, parser difficile
  - NoSQL : pas de norme
  - Abstraction indispensable (masquer l'implémentation)
- Typographie
  - Snake - « propriete_avec_multiple_noms »
  - Mais pas « proprieteAvecMultipleNoms »
  - Body au format JSON
  - Contient des informations spécifiques à la requête pour la collection
  - Peut contenir une « Query » (DSL)
- Pagination
- offset / limit dans la Query
- Range dans le Header pour les octets d'un fichier binaire
- Tri
- orderby dans la Query

4.2 Corps de la requête

Une requête DSL se décompose en 4 parties principales :

Projections, Collections, Requêtes (critères=query), Filtres (tri, limite)

Pour comparaison avec le langage SQL

```
SELECT field1, field2 FROM table WHERE field3 < value LIMIT n SORT field1 ASC
```

- *SELECT field1, field2* : la Projection
- *FROM table* : la Collection
- *WHERE field3 < value* : la partie Query
- *LIMIT n SORT field1 ASC* : les filtres

Dans le cas de multiples queries, un champ de plus peut intervenir, il s'agit du paramètre **$depth** qui indique la direction dans l'arborescence de chaque *Query*.

Modèle générique CRUD

```
Create = POST
 data : { champ : valeur, champ : { champ : valeur } }
```

```
Read = GET
 filter : { limit, offset, orderby }, projection : { field : 0/1, ... }
```

```
Update = PUT (avec forme ~ POST) / PATCH
 action : { set : { field : value, ...}, inc : { field : value }, ... }
```

```
Delete = DELETE
 filter : { mult : true/false }
 roots = liste des Id de départ (sommet de l'arbre de classement)
```

```
HTTP POST/PUT /ressources
{
  roots : [ liste Id ],
  queries : [
          { query1 },
          { query2 }
  ],
  filter : { filter },
  data : { data }
}
```

```
HTTP GET /ressources
{
  roots : [ liste Id ],
  queries : [
          { query1 },
          { query2 }
  ],
  filter : { filter },
  projection : { projection }
}
```

```
HTTP PATCH /ressources
{
  roots : [ liste Id ],
  queries : [
          { query1 },
          { query2 }
  ],
  filter : { filter },
  action : { action }
}
```

```
HTTP DELETE /ressources
{
  roots : [ liste Id ],
  queries : [
          { query1 },
          { query2 }
  ],
  filter : { filter }
}
```

Une query est exprimée avec des opérateurs (inspirés de MongoDB / Elastic)

Catégorie	Opérateurs	Arguments	Commentaire
Accès direct	$path	identifiants	Accès direct à un noeud
Booléens	$and, $or, $not	opérateurs	Combinaison logique d'opérateurs
Comparaison	$eq, $ne, $lt, $lte, $gt, $gte	Champ et valeur	Comparaison de la valeur d'un champ et la valeur passée en argument
	$range	Champ, $lt, $lte, $gt, $gte et valeurs	Comparaison de la valeur d'un champ avec l'intervalle passé en argument
Existence	$exists, $missing, $isNull	Champ	Existence d'un champ
Tableau	$in, $nin	Champ et valeurs	Présence de valeurs dans un tableau
	$size	Champ et taille	Comparaison (égale) de la taille d'un tableau
	[n] **UNSUPPORTED**	Position (n >= 0)	Élément d'un tableau
Textuel	$term, $wildcard	Champ, mot clef	Comparaison de champs mots-clefs à valeur exacte
	$match, $matchPhrase, $matchPhrasePrefix	Champ, phrase, $max_expansions (optionnel)	Recherche plein texte soit sur des mots, des phrases ou un préfixe de phrase
	$regex	Champ, Expression régulière	Recherche via une expression régulière
	$search	Champ, valeur	Recherche du type moteur de recherche
	$flt, $mlt	Champ, valeur	Recherche « More Like This », soit par valeurs approchées
Géomatique	$geometry, $box, $polygon, $center	Positions	Définition d'une position géographique
UNSUPPORTED	$geoWithin, $geoIntersects, $near	Une forme	Recherche par rapport à une forme géométrique

Chaque Query dispose éventuellement d'arguments additionnels pour gérer l'arborescence :

Catégorie	Opérateur	Arguments	Commentaire
Profondeur	$depth, $exactdepth	+ ou - n	Permet de spécifier si la query effectue une recherche vers les racines (-) ou vers les feuilles (+) et de quelle profondeur (n), avec une profondeur relative ($depth) ou exacte ($exactdepth) - $depth = 0 signifie que l'on ne change pas de profondeur (mêmes objets concernés) - $depth > 0 indique une recherche vers les fils uniquement - $depth < 0 indique une recherche vers les pères uniquements (cf. schéma sur les multiples queries)
Collection	$source	units / objects	Permet dans une succession de Query de changer de collection. Attention, la dernière Query doit respecter la collection associée à la requête

4.3 API Java et documentation

Documentation :

- Ressource documentaire : raml/externe/introduction.html [8]

API java :

- Dans common/common-database-vitam/common-database-public

8. http://www.programmevitam.fr/ressources/Doc0.20.0/raml/externe/introduction.html

- fr.gouv.vitam.common.database.builder.query ; notamment **VitamFieldsHelper** et **QueryHelper**
- fr.gouv.vitam.common.database.builder.query.action ; dont **UpdateActionHelper**
- fr.gouv.vitam.common.database.builder.request.multiple ; dont **DeleteMultiQuery**, **SelectMultiQuery**, **InsertMultiQuery**, **UpdateMultiQuery**
- fr.gouv.vitam.common.database.builder.request.single ; dont **Delete**, **Insert**, **Select**, **Update**

CHAPITRE 5

DSL Vitam

Le DSL (Domain Specific Language) Vitam est composé de deux parties :

- Request : Contient la structure du Body contenant la requête au format Json. Le DSL permet d'exprimer un grand nombre de possibilités de requêtes. Dans le cadre des *Units* et *Objects*, cette requête peut être arborescente et multiples.
- Response : Contient la structure du Body contenant le résultat au format Json. Il contient différentes informations utiles ou demandées.

5.1 Request

Une requête est composée de plusieurs parties, et en particulier le Body qui contient un Json exprimant la requête.

Elle peut être complétée par quelques valeurs dans le *Header* :

- **X-Application-Id** : (**UNSUPPORTED**) pour conserver la session (valeur non signifiante) dans les journaux et logs du SAE associés à l'opération demandée
- **X-Valid : true** : (**UNSUPPORTED**) pour une requête HEAD sur un **Object** pour vérifier la validité (check d'empreinte)
- **X-Tenant-Id** : pour chaque requête, le tenant sur lequel doit être exécutée la requête
- **X-Qualifier** et **X-Version** : pour une requête GET sur un **Object** pour récupérer un usage et une version particulière
- **X-Callback** (**UNSUPPORTED**) : pour les opérations de longue durée et donc asynchrones pour indiquer l'URL de Callback
- **X-Cursor : true** et **X-Cursor-Id** (**UNSUPPORTED**) : pour la gestion d'une requête en mode "curseur"
- **X-Http-Method-Override** : pour permettre aux clients HTTP ne supportant pas tous les modes de la RFC 7231 (GET/PUT/DELETE avec Body)

5.1.1 BuilderRequest

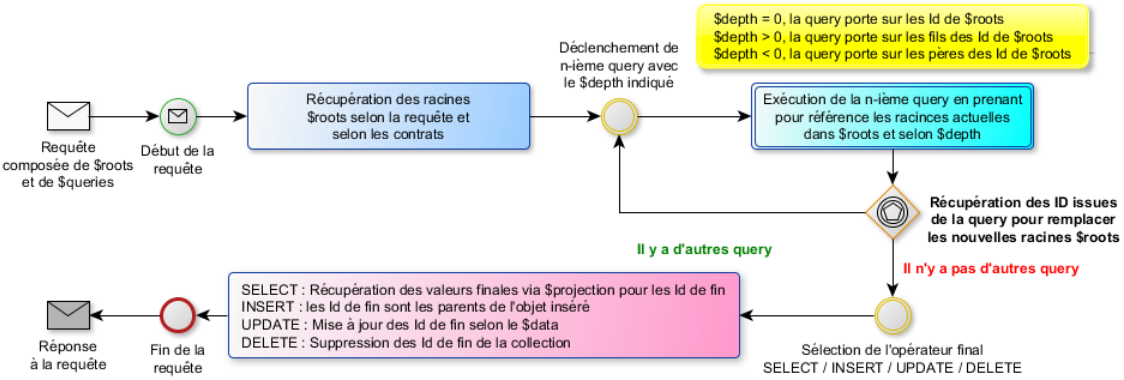
Il existe des Helpers en Java pour construire les requêtes au bon format (hors headers) dans le package **common-database-public.**

- **Request**
  - Multiple : fr.gouv.vitam.common.database.builder.request.multiple avec MultipleDelete, MultipleInsert, MultipleSelect et MultipleUpdate (pour Units et Objects)
  - Single : fr.gouv.vitam.common.database.builder.request.single avec Select, Insert, Delete, Update (pour toutes les autres collections, en fonction des droits et des possibilités)

- **Query** qui sont des arguments d'une Request
  - fr.gouv.vitam.common.database.builder.query avec BooleanQuery, CompareQuery, ... et surtout le **QueryHelper** et le **VitamFieldsHelper** pour les champs protégés (commençant par un '#')
  - dans le cas de update fr.gouv.vitam.common.database.builder.action avec AddAction, IncAction, ... et surtout le **UpdateActionHelper**

5.1.2 Collections Units et Objects uniquement

- **$roots** :
  - Il s'agit des racines (Units) à partir desquelles la requête est exprimée (toutes les recherches sur les Units et Objects sont en mode arborescente). Il correspond d'une certaine façon à "*FROM x*" dans le langage SQL étendu au cas des arborescences.
  - Autres collections : ce champ n'existe pas car il n'y a pas d'arborescence.
- **$query** :
  - $query peut contenir plusieurs Query, qui seront exécutées successivement (tableau de Query).
  - Une Query correspond à la formulation "*WHERE xxx*" dans le langage SQL, c'est à dire les critères de sélection.
  - La succession est exécutée avec la signification suivante :
    - Depuis $roots, chercher les Units/Objects tel que Query[1], conduisant à obtenir une liste d'identifiants[1]
    - Cette liste d'identifiants[1] devient le nouveau $roots, chercher les Units/Objects tel que Query[2], conduisant à obtenir une liste d'identifiants[2]
    - Et ainsi de suite, la liste d'identifiants[n] de la dernière Query[n] est la liste de résultat définitive sur laquelle l'opération effective sera réalisée (SELECT, UPDATE, INSERT, DELETE) selon ce que l'API supporte (GET, PUT, POST, DELETE).
    - Chaque query peut spécifier une profondeur où appliquer la recherche :
      - $depth = 0 : sur les items spécifiés (filtre sur les mêmes items, à savoir pour la première requête ceux de $roots, pour les suivantes, le résultat de la requête précédente, c'est à dire le nouveau $roots)
      - $depth < 0 : sur les items parents (hors les items spécifiés dans le $roots courant)
      - $depth > 0 : sur les items enfants (hors les items spécifiés dans le $roots courant)
      - par défaut, $depth vaut 1 (enfants immédiats dans le $roots courant)
    - Le principe est résumé dans le graphe d'états suivant :



    - $source (**UNSUPPORTED**) permet de changer de collections entre deux query (unit ou object)
- **$filter** :

    - Il permet de spécifier des filtres additionnels :
        - Pour *GET* :
            - **$limit** : le nombre maximum d'items retournés (limité à 1000 par défaut, maximum à 100000)
            - **$per_page** (**UNSUPPORTED**) : le nombre maximum des premiers items retournés (limité à 100 par défaut, maximum à 100)
            - **$offset** : la position de démarrage dans la liste retournée (positionné à 0 par défaut, maximum à 100000)
            - **$orderby : { fieldname : 1, fieldname : -1 }** : permet de définir un tri ascendant ou descendant
                - **IMPORTANT** : pour un champ analysé (plein texte), le tri n'est pas lexicographique mais basé sur le score de correspondance
            - **$hint : "nocache"** (**UNSUPPORTED**) permet de spécifier si l'on ne veut pas bénéficier du cache (cache actif par défaut)
        - Pour *POST*, *PUT* et *DELETE*
            - **$mult** : booléen où *true* signifie que l'opération peut concerner de multiples items (par défaut, positionné à *false*)
                - *POST* : les pères sélectionnés peuvent être multiples
                - *PUT* : la mise à jour peut concerner de multiples items simultanément
                - *DELETE* : l'effacement peut concerner plusieurs items
- **$projection : { fieldname : 0, fieldname : 1 }** uniquement pour *GET* (SELECT)
    - Il permet de définir la forme du résultat que l'on souhaite. Il correspond au "*SELECT* *" dans le langage SQL.
    - Une valeur à 1 réclame le champ.
    - Une valeur à 0 exclut le champ.
    - Si rien n'est spécifié, cela correspond à tous les champs (équivalent à "SELECT *")
- **$data** : uniquement pour *POST* (INSERT)
    - Permet de définir le contenu à insérer dans la collection.
- **$action** : uniquement pour *PUT* (UPDATE)
    - Permet de définir le contenu à modifier dans la collection.
- Il n'y a pas d'argument complémentaire pour *DELETE* (DELETE) hormis la partie *$filter*
- **facetQuery** (**UNSUPPORTED**) : uniquement pour *GET* et optionnel
    - Permet de définir des sous-requêtes (sous la forme d'agrégats) correspondant généralement à des facettes dans l'application Front-Office

5.1.3 Autres collections

- **$query** :
    - Il s'agit d'une **Query** unique.
    - Une Query correspond à la formulation "*WHERE xxx*" dans le langage SQL, c'est à dire les critères de sélection.
- **$filter** :
    - Il permet de spécifier des filtres additionnels :
        - Pour *GET* :
            - **$limit** : le nombre maximum d'items retournés (limité à 1000 par défaut, maximum à 100000)
            - **$per_page** (**UNSUPPORTED**) : le nombre maximum des premiers items retournés (limité à 100 par défaut, maximum à 100)

        - **$offset** : la position de démarrage dans la liste retournée (positionné à 0 par défaut, maximum à 100000)
            - **IMPORTANT** : pour un champ analysé (plein texte), le tri n'est pas lexicographique mais basé sur le score de correspondance
        - **$orderby : { fieldname : 1, fieldname : -1 }** : permet de définir un tri ascendant ou descendant
        - **$hint : "nocache"** (**UNSUPPORTED**) permet de spécifier si l'on ne veut pas bénéficier du cache (cache actif par défaut)
    - Pour *POST*, *PUT* et *DELETE*
        - **$mult** (**UNSUPPORTED**) : booléen où *true* signifie que l'opération peut concerner de multiples items (par défaut, positionné à *false*)
            - *POST* : les pères sélectionnés peuvent être multiples
            - *PUT* : la mise à jour peut concerner de multiples items simultanément
            - *DELETE* : l'effacement peut concerner plusieurs items
- **$projection : { fieldname : 0, fieldname : 1 }** uniquement pour *GET*
    - Il permet de définir la forme du résultat que l'on souhaite. Il correspond au "*SELECT **" dans le langage SQL.
    - Une valeur à 1 réclame le champ.
    - Une valeur à 0 exclut le champ.
    - Si rien n'est spécifié, cela correspond à tous les champs (équivalent à "SELECT *")
- **$data** : uniquement pour *POST*
    - Permet de définir le contenu à insérer dans la collection.
- **$action** : uniquement pour *PUT*
    - Permet de définir le contenu à modifier dans la collection.
- **facetQuery** (**UNSUPPORTED**) : uniquement pour *GET* et optionnel
    - Permet de définir des sous-requêtes (sous la forme d'agrégats) correspondant généralement à des facettes dans l'application Front-Office

5.1.4 Query

Les commandes de la Query peuvent être :

Une query est exprimée avec des opérateurs (inspirés de MongoDB / Elastic)

Catégorie	Opérateurs	Arguments	Commentaire
Accès direct	$path	identifiants	Accès direct à un noeud
Booléens	$and, $or, $not	opérateurs	Combinaison logique d'opérateurs
Comparaison	$eq, $ne, $lt, $lte, $gt, $gte	Champ et valeur	Comparaison de la valeur d'un champ et la valeur passée en argument
	$range	Champ, $lt, $lte, $gt, $gte et valeurs	Comparaison de la valeur d'un champ avec l'intervalle passé en argument
Existence	$exists, $missing, $isNull	Champ	Existence d'un champ
Tableau	$in, $nin	Champ et valeurs	Présence de valeurs dans un tableau
	$size	Champ et taille	Comparaison (égale) de la taille d'un tableau
	[n] **UNSUPPORTED**	Position (n >= 0)	Élément d'un tableau
Textuel	$term, $wildcard	Champ, mot clef	Comparaison de champs mots-clefs à valeur exacte
	$match, $match_all, $match_phrase, $match_phrase_prefix	Champ, phrase, $max_expansions (optionnel)	Recherche plein texte soit sur des mots, des phrases ou un préfixe de phrase
	$regex	Champ, Expression régulière	Recherche via une expression régulière
	$search	Champ, valeur	Recherche du type moteur de recherche
	$flt, $mlt	Champ, valeur	Recherche « More Like This », soit par valeurs approchées
Géomatique	$geometry, $box, $polygon, $center	Positions	Définition d'une position géographique
UNSUPPORTED	$geoWithin, $geoIntersects, $near	Une forme	Recherche par rapport à une forme géométrique

Chaque Query dispose éventuellement d'arguments additionnels pour gérer l'arborescence :

Catégorie	Opérateur	Arguments	Commentaire
Profondeur	$depth, $exactdepth	+ ou - n	Permet de spécifier si la query effectue une recherche vers les racines (-) ou vers les feuilles (+) et de quelle profondeur (n), avec une profondeur relative ($depth) ou exacte ($exactdepth) - $depth = 0 signifie que l'on ne change pas de profondeur (mêmes objets concernés) - $depth > 0 indique une recherche vers les fils uniquement - $depth < 0 indique une recherche vers les pères uniquements (cf. schéma sur les multiples queries)
Collection	$source	units / objects	Permet dans une succession de Query de changer de collection. Attention, la dernière Query doit respecter la collection associée à la requête

5.1.5 Actions

Dans la commande PUT (Update) :

Opérateur	Arguments	Commentaire
$set	nom de champ, valeur	change la valeur du champ
$unset	liste de noms de champ	enlève le champ
$min, $max	nom de champ, valeur	change la valeur du champ à la valeur minimale/maximale si elle est supérieure/inférieure à la valeur précisée
$inc	nom de champ, valeur	incrémente/décremente la valeur du champ selon la valeur indiquée
$rename	nom de champ, nouveau nom	change le nom du champ
$push, $pull	nom de champ, liste de valeurs	ajoute en fin ou retire les éléments de la liste du champ (qui est un tableau)
$add	nom de champ, liste de valeurs	ajoute les éléments de la liste du champ (qui est un "set" avec unicité des valeurs)
$pop	nom de champ, -1 ou 1	retire le premier (-1) ou le dernier (1) de la liste du champ

5.1.6 FacetQuery UNSUPPORTED

Lors d'une commande GET (Select), les possibilités envisagées sont :

Opérateur pour les facet	Arguments	Commentaire
$cardinality	nom de champ	indique le nombre de valeurs différentes pour ce champ
$avg, $max, $min, $stats	nom de champ numérique	indique la valeur moyenne, maximale, minimale ou l'ensemble des statistiques du champ
$percentile	nom de champ numérique, valeurs optionnelles	indique les percentiles de répartition des valeurs du champ, éventuellement selon la répartition des valeurs indiquées
$date_histogram	nom de champ, intervalle	indique la répartition selon les dates selon un intervalle définie sous la forme "nX" où n est un nombre et X une lettre parmi y (year), M (month), d(day), h(hour), m(minute), s(seconde) ou encore de la forme "year", "quarter", "month", "week", "day", "hour", "minute" ou "second"
$date_range	nom de champ, format, ranges	indique la répartition selon les dates selon un intervalle défini "ranges" : [{ "to" : "now-10M/M" }, { "from" : "now-10M/M" }] et "format" : "MM-yyyy"
$range	nom de champ, intervalles	indique la répartition selon des valeurs numériques par la forme "ranges" : [{ "to" : 50 }, { "from" : 50, "to" : 100 }, { "from" : 100 }]
$terms	nom de champ	indique la répartition selon des valeurs textuelles du champ
$significant_terms	nom de champ principal, nom de champ secondaire	indique la répartition selon des valeurs textuelles du champ principal et affiche pour chaque les termes significatifs pour le second champ

5.1.7 Exemples

5.1.7.1 GET

- La query sélectionne les Units qui vont être retournées. - Le contenu est :
  - Pour **Units/Objects** :
    - **$roots**
    - **$query**
    - **$filter**
    - **$projection : { fieldname : 0, fieldname : 1 }**
    - **facetQuery** optionnel (**UNSUPPORTED**)
  - Pour les autres collections :
    - **$query**
    - **$filter**
    - **$projection : { fieldname : 0, fieldname : 1 }**
    - **facetQuery** optionnel (**UNSUPPORTED**)

Exemple :

```
{
  "$roots": [ "id0" ],
  "$query": [
    { "$match": { "Title": "titre" }, "$depth": 4 }
  ],
  "$filter": { "$limit": 100 },
  "$projection": { "$fields": { "#id": 1, "Title": 1, "#type": 1, "#parents": 1, "
↪#object": 1 } },
  "$facetQuery": { "$terms": "#object.#type" } //(**UNSUPPORTED**)
}
```

5.1.7.2 POST

- La query sélectionne le ou les Units parents de celle qui va être créée. - Le contenu est :
  - Pour **Units/Objects** :
    - **$roots**
    - **$query**
    - **$filter**
    - **$data**
  - Pour les autres collections :
    - **$query**
    - **$filter**
    - **$data**

```
{
 "$roots": [ "id0" ],
 "$query": [
   { "$match": { "Title": "titre" }, "$depth": 4 }
 ],
 "$filter": {  },
 "$data": { "Title": "mytitle", "description": "my description", "value": 1 }
}
```

5.1.7.3 PUT

- **La query sélectionne les Units sur lesquelles l'update va être réalisé.**
    - **Le contenu est :**
        - **Pour Units/Objects :**
            - **$roots**
            - **$query**
            - **$filter**
            - **$action**
        - **Pour les autres collections :**
            - **$query**
            - **$filter**
            - **$action**

```
{
 "$roots": [ "id0" ],
 "$query": [
   { "$eq": { "Title": "mytitle" }, "$depth": 5 }
 ],
 "$filter": {  },
 "$action": [{ "$inc": { "value": 10 } }]
}
```

5.2 Response

Une réponse est composée de plusieurs parties :

- **$hits** :
    - **limit** : le nombre maximum d'items retournés (limité à 1000 par défaut)
    - **offset** : la position de démarrage dans la liste retournée (positionné à 0 par défaut)
    - **total** : le nombre total potentiel (estimation) des résultats possibles
    - **size** : le nombre réel d'items retournés
    - **time_out** : Vrai si la requête a durée trop longtemps et donc avec un résultat potentiellement partiel
- **$context** : rapelle la requête exprimée
- **$results** : contient le résultat de la requête sous forme d'une liste d'items
- **$facets** : contient le résultat de la partie $facetQuery.

Des champs sont protégés dans les requêtes :

- Il est interdit d'exprimer un champ qui démarre par un '_'
- La plupart de ces champs protégés sont interdits à la modification. Ils ne sont utilisables que dans la partie *$projection* ou *$query* mais pas dans la partie *$data*
- Communs Units et Objects
    - **#id** est l'identifiant de l'item
    - **#all** est l'équivalent de "SELECT *"
    - **#unitups** est la liste des Units parents
    - **#tenant** est le tenant associé
    - **#operations** est la liste des opérations qui ont opéré sur cet élément

    - **#originating_agency** est l'OriginatingAgency su SIP d'origine
    - **#originating_agencies** est l'ensemble des OriginatingAgencies issues du SIP et des rattachements (héritage)
    - **#storage** est l'état de stockage
    - **#score** (**UNSUPORTED**) contiendra en cas de requête avec plein texte le score de pertinence
- Spécifiques pour les Units
    - **#unittype** est la typologie du Unit (Arbre **HOLLDING_UNIT**, Plan **FILING_UNIT** ou ArchiveUnit **INGEST**)
    - **#nbunits** est le nombre de fils immédiats à un Unit donné
    - **#object** est l'objet associé à un Unit (s'il existe)
    - **#type** est le type d'item (Document Type)
    - **#allunitups** est l'ensemble des Units parents (depuis les racines)
    - **#management** est la partie règles de gestion associées au Unit (ce champ est autorisée à être modifiée et donc dans *$data*)
- Spécifiques pour les Objects
        - **#type** est le type d'item (Type d'Objet : **Document**, **Audio**, **Video**, **Image**, **Text**, ...)
        - **#nbobjects** est le nombre d'objets binaires (usages/version) associé à cet objet
        - **#qualifiers** est la liste des qualifiers disponibles
            - Les "qualifiers" disponibles pour les objets :
                - **PhysicalMaster** pour original physique
                - **BinaryMaster** pour conservation
                - **Dissemination** pour la version de diffusion compatible avec un accès rapide et via navigateur
                - **Thumbnail** pour les vignettes pour les affichages en qualité très réduite et très rapide en "prévue"
                - **TextContent** pour la partie native texte (ASCII UTF8)
        - Un raccourci exite : **#usage**
            - **#size** est la taille d'un objet
            - **#format** est le format (PUID) d'un objet

La réponse dispose également de champs dans le *Header* :

- **FullApiVersion** : (**UNSUPPORTED**) retourne le numéro précis de la version de l'API en cours d'exécution
- **X-Request-Id** : pour chaque requête, un unique identifiant est fourni en réponse
- **X-Tenant-Id** : pour chaque requête, le tenant sur lequel a été exécutée l'opération demandée
- **X-Application-Id** : (**UNSUPPORTED**) pour conserver la session (valeur non signifiante) dans les journaux et logs associés à l'opération demandée
- **X-Qualifier** et **X-Version** : pour une requête GET sur un **Object** pour indiquer un usage et une version particulière
- **X-Callback** (**UNSUPPORTED**) : pour les opérations de longue durée et donc asynchrones pour indiquer l'URL de Callback
- (**UNSUPPORTED**) Si **X-Cursor : true** a été spécifié et si la réponse nécessite l'usage d'un curseur (nombre de réponses > *$per_page*), le SAE retourne **X-Cursor-Id** et **X-Cursor-Timeout** (date de fin de validité du curseur) : pour la gestion d'une requête en mode "curseur" par le client

5.2.1 Exemples

5.2.1.1 Réponse pour Units

```
{
 "$hits": {
   "total": 3,
   "size": 3,
   "offset": 0,
   "limit": 100,
   "time_out": false
 },
 "$context": {
   "$roots": [ "id0" ],
   "$query": [
     { "$match": { "Title": "titre" }, "$depth": 4 }
   ],
   "$filter": { "$limit": 100 },
   "$projection": { "$fields": { "#id": 1, "Title": 1, "#type": 1, "#unitups": 1, "
↪#object": 1 } },
   "$facetQuery": { "$terms": "#object.#type" }
 },
 "$results": [
   {
     "#id": "id1", "Title": "titre 1", "#type": "DemandeCongés",
     "#unitups": [ { "#id": "id4", "#type": "DossierCongés" } ],
     "#object": { "#id": "id101", "#type": "Document",
       "#qualifiers": { "BinaryMaster": 5, "Dissemination": 1, "Thumbnail": 1,
↪"TextContent": 1 } }
   },
   {
     "#id": "id2", "Title": "titre 2", "#type": "DemandeCongés",
     "#unitups": [ { "#id": "id4", "#type": "DossierCongés" } ],
     "#object": { "#id": "id102", "#type": "Document",
       "#qualifiers": { "BinaryMaster": 5, "Dissemination": 1, "Thumbnail": 1,
↪"TextContent": 1 } }
   },
   {
     "#id": "id3", "Title": "titre 3", "#type": "DemandeCongés",
     "#unitups": [ { "#id": "id4", "#type": "DossierCongés" } ],
     "#object": { "#id": "id103", "#type": "Image",
       "#qualifiers": { "BinaryMaster": 3, "Dissemination": 1, "Thumbnail": 1,
↪"TextContent": 1 } }
   }
 ],
 "$facet": { // **UNSUPPORTED**
   "#object.#type": { "Document": 2, "Image": 1 }
 }
}
```

5.2.1.2 Réponse pour Objects

```
{
 "$hits": {
   "total": 3,
   "size": 3,
```

```
    "offset": 0,
    "limit": 100,
    "time_out": false
  },
  "$context": {
    "$roots": [ "id0" ],
    "$query": [
      { "$match": { "Title": "titre" }, "$depth": 4, "$source": "units" },
      { "$eq": { "#type": "Document" }, "$source": "objects" }
    ],
    "$filter": { "$limit": 100 },
    "$projection": { "$fields": { "#id": 1, "#qualifiers": 1, "#type": 1, "#unitups":␣
↪1 } }
  },
  "$results": [
    {
      "#id": "id101", "#type": "Document",
      "#qualifiers": { "BinaryMaster": 5, "Dissemination": 1, "Thumbnail": 1,
↪"TextContent": 1 },
      "#unitups": [ { "#id": "id1", "#type": "DemandeCongés" } ]
    },
    {
      "#id": "id102", "#type": "Document",
      "#qualifiers": { "BinaryMaster": 5, "Dissemination": 1, "Thumbnail": 1,
↪"TextContent": 1 },
      "#unitups": [ { "#id": "id2", "#type": "DemandeCongés" } ]
    },
    {
      "#id": "id103", "#type": "Document",
      "#qualifiers": { "BinaryMaster": 3, "Dissemination": 1, "Thumbnail": 1,
↪"TextContent": 1 },
      "#unitups": [ { "#id": "id3", "#type": "DemandeCongés" } ]
    }
  ]
}
```


5.2.2 Réponse en cas d'erreurs

En cas d'erreur, Vitam retourne un message d'erreur dont le format est :

- **httpCode** : code erreur Http
- **code** : code erreur Vitam
- **context** : contexte de l'erreur
- **state** : statut en format de message court sous forme de code
- **message** : statut en format de message court
- **description** : statut détaillé
- **errors** : le cas échéant des sous-erreurs associées avec le même format

5.2.2.1 Exemple de retour en erreur


```
{
  "httpCode": 404,
  "code" : "codeVitam1",
```

```
 "context": "ingest",
 "state": "Item_Not_Found",
 "message": "Item is not found",
 "description": "Operation on item xxx cannot be done since item is not found in <
↪<resourcePathName>>",
 "errors": [
   { "httpCode": 415,
     "code" : "codevitam2",
     "context": "ingest",
     "state": "Unsupported_Media_Type",
     "message": "Unsupported media type detected",
     "description": "File xxx has an unsupported media type yyy" },
   { "httpCode": 412,
     "code": "codevitam3",
     "context": "ingest",
     "state": "Precondition_Failed",
     "message": "Precondition in error",
     "description": "Operation on file xxx cannot continue since precondition is in␣
↪error" }
 ]
}
```

5.2.3 Cas particulier : HEAD pour test d'existence et validation (UNSUPPORTED)

La commande *HEAD* permet de savoir pour un item donné s'il existe (retour **204**) ou pas (retour **404**).

(**UNSUPPORTED**) Si dans le Header est ajoutée la commande **X-Valid : true**, la commande *HEAD* vérifie si l'item (Unit ou Object) existe et s'il est conforme (audit de l'item sur la base de son empreinte). S'il n'est pas conforme mais qu'il existe, le retour est **417** (Expectation Failed).

CHAPITRE 6

Exemples

6.1 Recherche par ArchivalAgencyArchiveUnitIdentifier

EndPoint : /access-external/v1/units

Client Java

```
try (AccessExternalClient client = AccessExternalClientFactory.getInstance().getClient()) {
   Integer  tenantId= 0;  //à titre d'exemple
   final String selectQuery = "{ \"$query\" : [ { \"$eq\": { \"#ArchivalAgencyArchiveUnitIdentifier\" : "20130456/3\" } } ]    "
}";

   final JsonNode queryJson = JsonHandler.getFromString(selectQuery);

   client.selectUnits(queryJson, tenantId);

} catch (InvalidCreateOperationException e) {
LOG ...
}
```

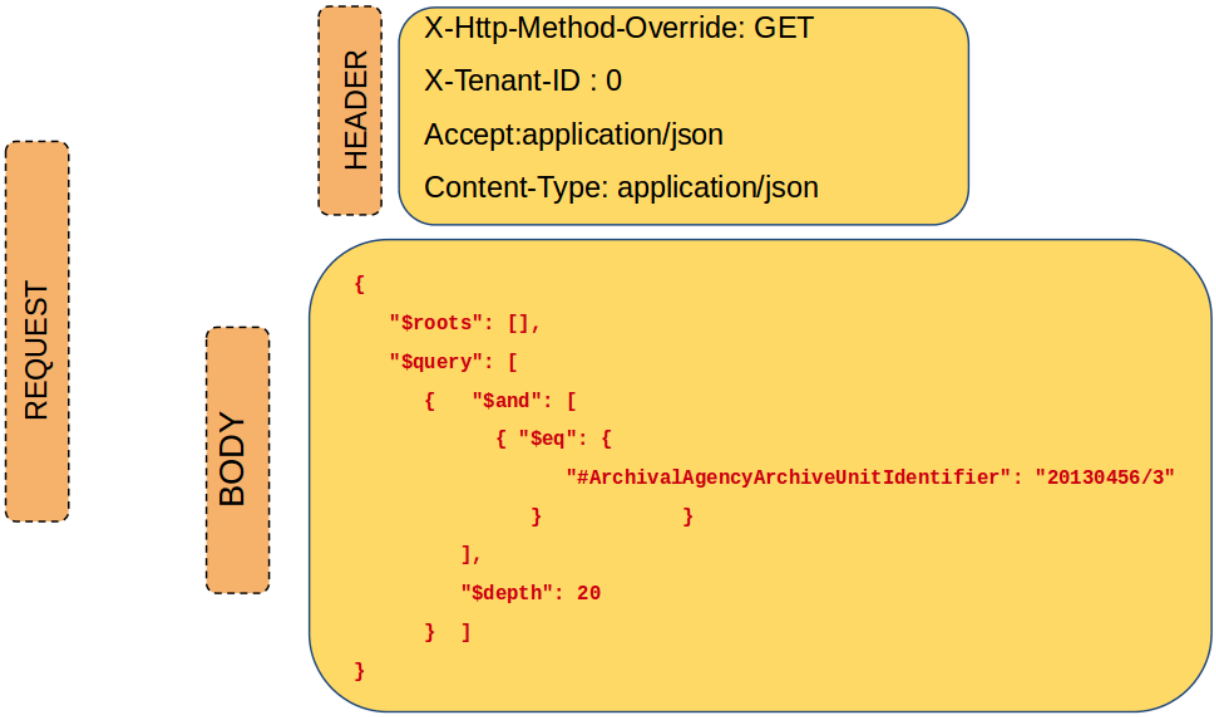
Client Java avec construction DSL

EndPoint : access-external/v1/units

```
try (AccessExternalClient client = AccessExternalClientFactory.getInstance().getClient()) {

Select select = new Select() ;
Integer  tenantId= 0;  //à titre d'exemple
JsonNode   queryDsl = null;
final SelectParserSingle parser = new SelectParserSingle();
            parser.parse(select.getFinalSelect());
            parser.addCondition(QueryHelper.eq("ArchivalAgencyArchiveUnitIdentifier", "20130456/3"));
            queryDsl = parser.getRequest().getFinalSelect();
            client.selectUnits(queryJson, tenantId);
} catch (InvalidCreateOperationException e) {
LOG ...
}
```

Postman

POST /access-external/v1/units



6.2 Recherche par producteur (FRAN_NP_005568)

Client Java

Endpoint : /admin-external/v1/accession-registers

```
final AccessExternalClient client =
  AccessExternalClientFactory.getInstance().getClient();
final String options =

  "{ \"$query\" : [ { \"$eq\": { \"OriginatingAgency\" : \ "FRAN_NP_005568",\" } } ]   " }";
Integer  tenantId= 0 ;

try (AccessExternalClient client = AccessExternalClientFactory.getInstance().getClient()) {
  final Map<String, String> optionsMap = JsonHandler.getMapStringFromString(options);
  final JsonNode query = Dsl QueryHelper.createSingleQueryDSL(optionsMap);
  return client.getAccessionRegisterSummary(query, tenantId);
}
```

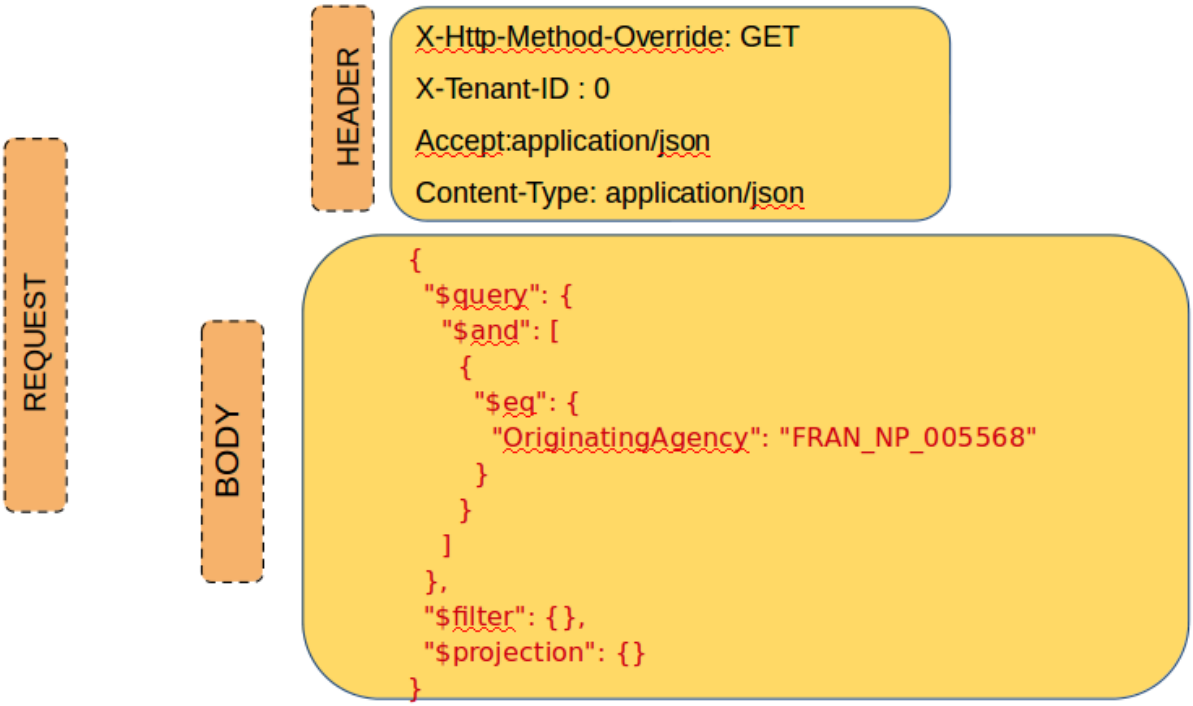
Client Java avec construction DSL

Endpoint : /admin-external/v1/accession-registers

```
try (AccessExternalClient client = AccessExternalClientFactory.getInstance().getClient()) {

final Select select = new Select();

select.setQuery(eq("OriginatingAgency", "FRAN_NP_005568"));

   client.getAccessionRegisterSummary(select.getFinalSelect(), tenantId);

} catch (InvalidCreateOperationException e) {
LOG ...
}
```

Postman

POST /admin-external/v1/accession-registers



6.3 Recherche par titre AND description AND dates

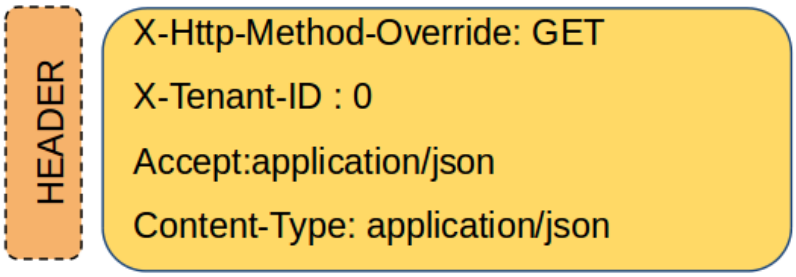
Client Java

Endpoint : /access-external/v1/units

```
try (AccessExternalClient client =
AccessExternalClientFactory.getInstance().getClient()) {

final Select select = new Select();

select.setQuery(or(eq("title", "test"),eq("Description", "test"));
....
.......
.........



  client.getAccessionRegisterSummary(select.getFinalSelect(), tenantId);

} catch (InvalidCreateOperationException e) {
LOG ...
}
```

Postman

GET /access-external/v1/units



6.4 Recherche libre titre OR description

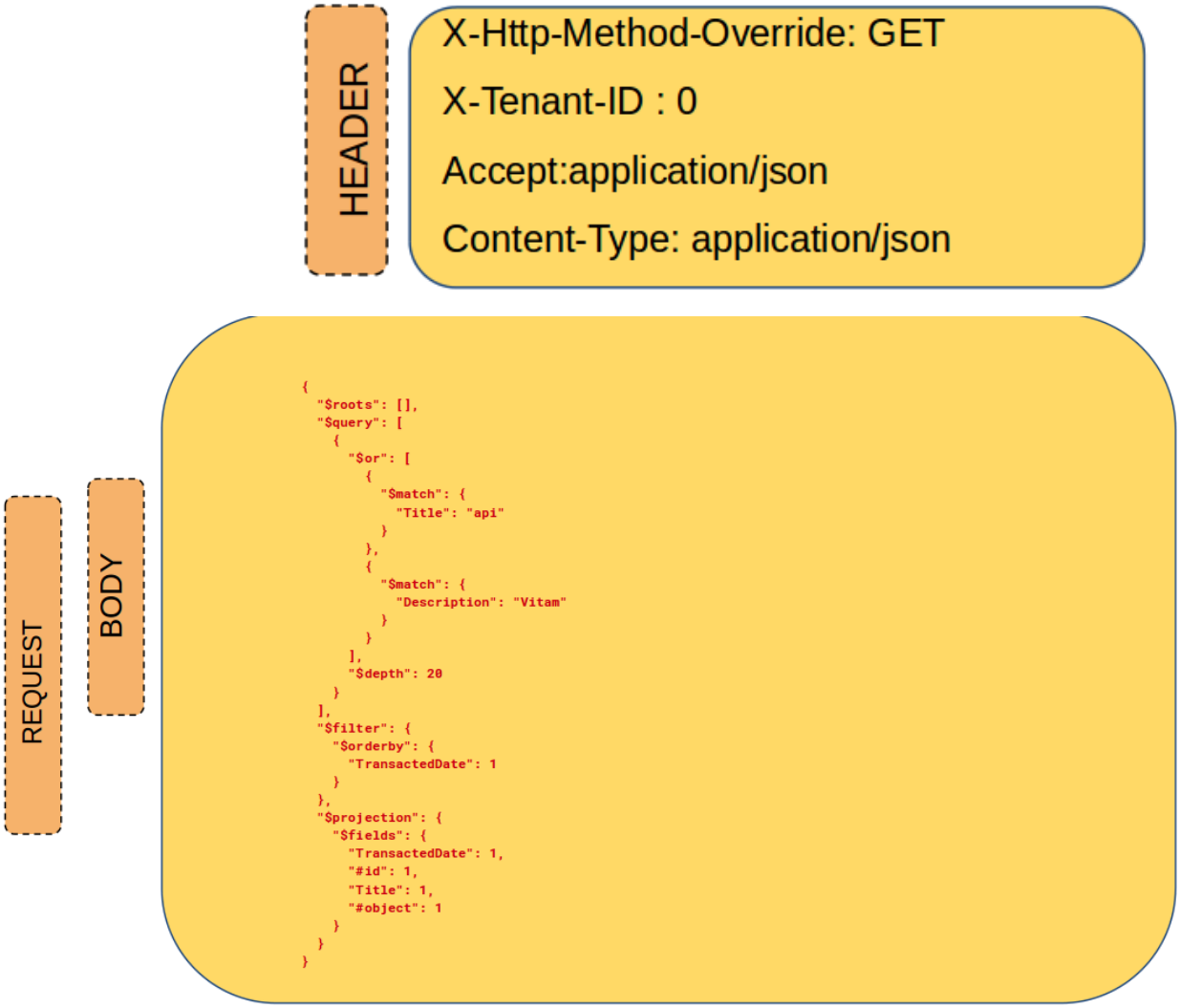
Client Java

Endpoint : /access-external/v1/units

```
try (AccessExternalClient client = AccessExternalClientFactory.getInstance().getClient())
{

final Select select = new Select();

select.setQuery(or(eq("title", "test"),eq("Description", "test"));

client.getAccessionRegisterSummary(select.getFinalSelect(), tenantId);

} catch (InvalidCreateOperationException e) {
LOG ...
}
```

Postman

GET /access-external/v1/units


HEADER
X-Http-Method-Override: GET
X-Tenant-ID : 0
Accept:application/json
Content-Type: application/json
REQUEST
BODY
{
"$roots": [],
"$query": [
{
"$or": [
{
"$match": {
"Title": "api"
}
},
{
"$match": {
"Description": "Vitam"
}
}
],
"$depth": 20
}
],
"$filter": {
"$orderby": {
"TransactedDate": 1
}
},
"$projection": {
"$fields": {
"TransactedDate": 1,
"#id": 1,
"Title": 1,
"#object": 1
}
}
}

CHAPITRE 7

Exemple DSL Vitam

Cette partie va essayer de montrer quelques exemples d'usages du DSL dans différentes conditions.

7.1 Collection Units

Points particuliers sur les end points

- **/units** : il s'agit ici de requêter un ensemble d'archives (Units) sur leurs métadonnées. Bien que non encore supportée, il sera possible de réaliser des UPDATE massifs sur cet end-point.
  - **$root** peut être vide ou renseigné : il sera contrôlé via les contrats d'accès associés à l'application.
    - S'il est vide, il prendra les valeurs renseignées par les contrats.
    - S'il contient des identifiants, il sera vérifié que ces identifiants sont bien soient ceux des contrats, soient fils de ceux spécifiés dans ces contrats.
  - Le résultat doit être une liste de Units (vide ou pas)
- **/units/id** : il s'agit ici de requêter depuis un Unit donné. Le résultat peut être multiple selon les query spécifiées (et notamment le *$depth*).
  - **$roots** est implicite à la valeur de id (si une valeur est spécifiée, elle sera ignorée)
    - Cet Id sera toujours contrôlé par rapport aux contrats d'accès associés à l'application.
  - Le résultat doit être une liste de Units (vide ou pas)
- **/units/id/object** : il s'agit ici d'accéder, s'il existe, à l'objet (ObjectGroup) associé à cet Unit.
  - Les query peuvent remonter les métadonnées (Header **Accept : application/json**)
  - Les query peuvent remonter un des objets binaires (Headers **Accept : application/octet-stream** et **X-Qualifier** et **X-Version**)
  - Le résultat doit être une liste de Objects (pour application/json) ou d'un seul objet binaire (pour application/octet-stream)

7.2 Collection Objects

Points particuliers sur les end points Cette collection est DEPRECATED et va disparaître car elle est contraire aux règles d'accès aux objets à partir d'une ArchiveUnit (/units/id/object).

- **/objects** : il s'agit ici de requêter un ensemble d'objets sur leurs métadonnées uniquement.
  - **$root** peut être vide ou renseigné : il sera contrôlé via les contrats d'accès associés à l'application et ne concerne que des Id de Units.
    - S'il est vide, il prendra les valeurs renseignées par les contrats (Id de Units parentes).

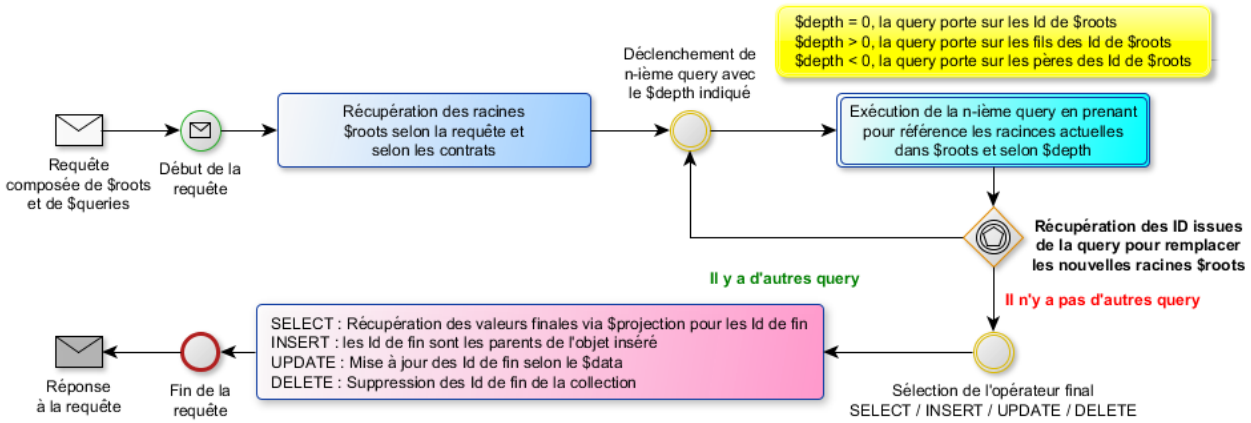
    - S'il contient des identifiants, il sera vérifié que ces identifiants sont bien soient ceux des contrats, soient fils de ceux spécifiés dans ces contrats (toujours des Id de Units).
  - Le résultat doit être une liste de Objects (vide ou pas)
  - Cette fonction est surtout utile pour des données statistiques sur les objets dans leur ensemble.
- **/objects/id** : **(susceptible d'être dépréciée dans une prochaine version)** il s'agit ici d'accéder à un objet (ObjectGroup).
  - Les query peuvent remonter les métadonnées (Header **Accept : application/json**)
  - Les query peuvent remonter un des objets binaires (Headers **Accept : application/octet-stream** et **X-Qualifier** et **X-Version**)
  - Le résultat doit être une liste de Objects (pour application/json) ou d'un seul objet binaire (pour application/octet-stream)

7.3 Exemples d'usages du DSL

7.3.1 Partie $query

7.3.1.1 Rappel sur l'usage de $depth

- $query peut contenir plusieurs Query, qui seront exécutées successivement (tableau de Query).
- Une Query correspond à la formulation "*WHERE xxx*" dans le langage SQL, c'est à dire les critères de sélection.
- La succession est exécutée avec la signification suivante :
  - Depuis $roots, chercher les Units/Objects tel que Query[1], conduisant à obtenir une liste d'identifiants[1]
  - Cette liste d'identifiants[1] devient le nouveau $roots, chercher les Units/Objects tel que Query[2], conduisant à obtenir une liste d'identifiants[2]
  - Et ainsi de suite, la liste d'identifiants[n] de la dernière Query[n] est la liste de résultat définitive sur laquelle l'opération effective sera réalisée (SELECT, UPDATE, INSERT, DELETE) selon ce que l'API supporte (GET, PUT, POST, DELETE).
  - Chaque query peut spécifier une profondeur où appliquer la recherche :
    - *$depth = 0* : sur les items spécifiés (filtre sur les mêmes items, à savoir pour la première requête ceux de $roots, pour les suivantes, le résultat de la requête précédente, c'est à dire le nouveau $roots)
    - *$depth < 0* : sur les items parents (hors les items spécifiés dans le $roots courant)
    - *$depth > 0* : sur les items enfants (hors les items spécifiés dans le $roots courant) - **par défaut, $depth vaut 1** (enfants immédiats dans le $roots courant)
  - Le principe est résumé dans le graphe d'états suivant :



- $source (**UNSUPPORTED**) permet de changer de collections entre deux query (unit ou object)

7.3.1.2 Détails sur la partie $filter $orderby

Il y a une différence entre un tri sur un champ non analysé (date, nombre, code) et un champ analysé :

- Pour un champ non analysé : l'ordre est lexicographique pour un texte, l'ordre est naturel pour un champ date ou nombre
- **IMPORTANT** : Pour un champ analysé (plein texte), le tri n'est pas lexicographique mais basé sur le score de correspondance
- l'ordre de déclaration des tris est respectés dans la réponse

7.3.1.3 Détails sur chaque commande de la partie $query

- $path : [id1, id2, ...]
  - Accès direct à un ou plusieurs noeuds
  - **$path : ["id1", "id2"]** est l'équivalent implicite de *$in : { #id : [id1, id2] }* mais sur le champ #id uniquement
  - **Important** : cette commande n'est autorisée qu'en première position. Elle implique une vérification que les *$roots* sont compatibles avec ces Ids qui deviennent les nouveaux $roots implicitement

```
{ "$path" : [ "id1", "id2" ] }

static include fr.gouv.vitam.common.database.builder.query.QueryHelper.*;
Query query = path("id1", "id2");
```

- $and, $or, $not
  - Combinaison logique d'opérateurs
  - **$and : [expression1, expression2, ...]** où chaque expression est une commande et chaque commande doit être vérifiée
  - **$or** où chaque expression est une commande et au moins une commande doit être vérifiée
  - **$not** où chaque expression est une commande et aucune ne doit être vérifiée
  - Exemple :

```
{ "$and" : [ { "$gt" : { "StartDate" : "2014-03-25" } }, { "$lte" : { "StartDate" :
↪"2014-04-25" } } ] }

static include fr.gouv.vitam.common.database.builder.query.QueryHelper.*;
Query query = and().add(gt("StartDate", dateFormat.parse("2014-03-25")),
         lte("StartDate", dateFormat.parse("2014-04-25"));
```

pour toute StartDate plus grande que le 25 mars 2014 et inférieure ou égale au 25 avril 2014 (équivalent à un $range dans ce cas)

- $eq, $ne, $lt, $lte, $gt, $gte
  - Comparaison de la valeur d'un champ et la valeur passée en argument
  - **$gt : { name : value }** où *name* est le nom du champ et *value* la valeur avec laquelle on compare le champ
    - $eq : égalité, marche également avec les champs non analysés (codes). **Attention** : pour les champs analysés, il s'agit d'un $match_all.
    - $ne : le champ n'a pas la valeur dournie
    - $lt, $lte : le champs a une valeur inférieure ou égale avec la valeur fournie
    - $gt, $gte : le champs a une valeur supérieure ou égale avec la valeur fournie
- Exemple :

```
{ "$gt" : { "StartDate" : "2014-03-25" } }

static include fr.gouv.vitam.common.database.builder.query.QueryHelper.*;
Query query = gt("StartDate", dateFormat.parse("2014-03-25"));
```

pour toute StartDate plus grande que le 25 mars 2014

- $range
  - Comparaison de la valeur d'un champ avec l'intervalle passé en argument
  - **$range : { name : { $gte : value, $lte : value } }** est un raccourci pour chercher sur un seul champ nommé *name* les Units dont la valeur est comprise entre la partie *$gt* ou *$gte* et la partie *$lt* ou *$lte*
  - Exemple :

```
{ "$range" : { "StartDate" : { "$gte" : "2014-03-25", "$lte" : "2014-04-25" } } }

static include fr.gouv.vitam.common.database.builder.query.QueryHelper.*;
Query query = range("StartDate", dateFormat.parse("2014-03-25"), true,
      dateFormat.parse("2014-04-25"), true);
```

pour toute StartDate plus grande ou égale au 25 mars 2014 mais inférieure ou égale au 25 avril 2014

- **$exists, $missing, $isNull**
  - Existence d'un champ
  - **$exists : name** où *name* est le nom du champ qui doit exister
  - **$missing** : le champ ne doit pas exister
  - **$isNull** : le champ existe mais vide
  - Exemple :

```
{ "$exists" : "StartDate" }

static include fr.gouv.vitam.common.database.builder.query.QueryHelper.*;
Query query = exists("StartDate");
```

pour tout Unit contenant le champ StartDate

- **$in, $nin**
  - Présence de valeurs dans un champ (ce champ peut être un tableau ou un simple champ avec une seule valeur)
  - **$in : { name : [value1, value2, ...] }** où *name* est le nom du tableau et le tableau de valeurs ce que peut contenir le tableau. Il suffit d'une seule valeur présente dans le tableau pour qu'il soit sélectionné.
    - **Attention** : pour les champs analysés, il s'agit d'un $match multiple via $or.
  - **$nin** est l'opérateur inverse, le tableau ne doit contenir aucune des valeurs spécifiées
  - Exemple :

```
{ "$in" : { ""#unitups" : ["id1", "id2"] } }

static include fr.gouv.vitam.common.database.builder.query.VitamFieldsHelper.*;
static include fr.gouv.vitam.common.database.builder.query.QueryHelper.*;
Query query = in(unitups(), "id1", "id2");
```

pour rechercher les Units qui ont pour parents immédiats au moins l'un des deux Id spécifiés

- **$size**
  - Taille d'un tableau

- **$size : { name : length }** où *name* est le nom du tableau et *length* la taille attendue (égalité)
- Exemple :

```
{ "$size" : { ""#unitups" : 2 } }

static include fr.gouv.vitam.common.database.builder.query.VitamFieldsHelper.*;
static include fr.gouv.vitam.common.database.builder.query.QueryHelper.*;
Query query = size(unitups(), 2);
```

pour rechercher les Units qui ont 2 parents immédiats exactement

- $term
  - Comparaison de champs avec une valeur exacte (non analysé)
  - **$term : { name : term, name : term }** où l'on fait une recherche exacte sur les différents champs indiqués
  - **Attention** : pour les champs analysés, il s'agit d'un $match_all.
  - Exemple :

```
{ "$term" : { "#id" : "guid" } }

static include fr.gouv.vitam.common.database.builder.query.VitamFieldsHelper.*;
static include fr.gouv.vitam.common.database.builder.query.QueryHelper.*;
Query query = term(id(), guid);
```

qui cherchera le Unit ayant pour Id celui précisé (équivalent dans ce cas à $eq) (non analysé, donc pour les codes uniquement)

- $wildcard
  - Comparaison de champs mots-clefs à valeur
  - **$wildcard : { name : term }** où l'on fait une recherche exacte sur le champ indiqué mais avec une possibilité d'introduire un '*' dans le contenu
  - **NOTA BENE** : cette requête est coûteuse.
  - Exemple :

```
{ "$wildcard" : { "#type" : "FAC*01" } }

static include fr.gouv.vitam.common.database.builder.query.VitamFieldsHelper.*;
static include fr.gouv.vitam.common.database.builder.query.QueryHelper.*;
Query query = wildcard(type(), "FAC*01");
```

qui cherchera les Units qui contiennent dans le type (Document Type) une valeur commençant par FAC et terminant par 01 (non analysé, donc pour les codes uniquement)

- $match, $match_all, $match_phrase, $match_phrase_prefix
  - Recherche plein texte soit sur des mots, des phrases ou un préfixe de phrase
  - **$match : { name : words, $max_expansions : n }** où *name* est le nom du champ, *words* les mots que l'on cherche, dans n'importe quel ordre, et optionnellement *n* indiquant une extension des mots recherchés ("seul" avec n=5 permet de trouver "seulement")
  - **$match_all : { name : words, $max_expansions : n }** où *name* est le nom du champ, *words* les mots que l'on cherche (tous), dans n'importe quel ordre, et optionnellement *n* indiquant une extension des mots recherchés ("seul" avec n=5 permet de trouver "seulement")
  - **$match_phrase** permet de définir une phrase (*words* constitue une phrase à trouver exactement dans cet ordre)
  - **$match_phrase_prefix** permet de définir que le champ *name* doit commencer par cette phrase

- **NOTA BENE** : dans le cas de champs non analysés, cette requête est remplacé par une requête de type "prefix".
- Exemple :

```
{ "$match" : { "Title" : "Napoléon Waterloo" } }

static include fr.gouv.vitam.common.database.builder.query.QueryHelper.*;
Query query = match("Title", "Napoléon Waterloo");
```

qui cherchera les Units qui contiennent les deux mots dans n'importe quel ordre dans le titre

```
{ "$match_phrase" : { "Description" : "le petit chat est mort" } }

static include fr.gouv.vitam.common.database.builder.query.QueryHelper.*;
Query query = matchPhrase("Description", "le petit chat est mort");
```

qui cherchera les Units qui contiennent la phrase n'importe où dans la description

- $regex
  - Recherche via une expression régulière
  - **NOTA BENE** : cette requête est très lenbte et très coûteuse.
  - **$regex : { name : regex }** où *name* est le nom du champ et *regex* l'expression au format expression régulière du contenu du champ
  - Exemple :

```
{ "$regex" : { "Title" : "Napoléon.\* [Waterloo | Leipzig]" } }

static include fr.gouv.vitam.common.database.builder.query.QueryHelper.*;
Query query = regex("Title", "Napoléon.\* [Waterloo | Leipzig]");
```

qui cherchera les Units qui contiennent exactement Napoléon suivi de n'importe quoi mais se terminant sur un choix parmi Waterloo ou Leipzig dans le titre

- $search
  - Recherche du type moteur de recherche
  - **$search : { name : searchParameter }** où *name* est le nom du champ, *searchParameter* est une expression de recherche
  - L'expression est formulée avec les opérateurs suivants :
    - **+** signifie AND
    - **|** signifie OR
    - **-** empêche le mot qui lui est accollé (tout sauf ce mot)
    - **"** permet d'exprimer un ensemble de mots en une phrase (l'ordre des mots est impératif dans la recherche)
    - ***** A la fin d'un mot signifie que l'on recherche tout ce qui contient un mot commençant par
    - **(** et **)** signifie une précédence dans les opérateurs (priorisation des recherches AND, OR)
    - **~N** après un mot est proche du ***** mais en limitant le nombre de caractères dans la complétion (fuzziness)
    - **~N** après une phrase (encadré par **"**) autorise des "trous" dans la phrase
    - **Attention** : pour les champs non analysés, il s'agit d'un $term multivalué (choix parmi plusieurs valeurs).
  - Exemple :

```
{ "$search" : { "Title" : "\"oeufs cuits\" +(tomate | patate) -frite" } }

static include fr.gouv.vitam.common.database.builder.query.QueryHelper.*;
Query query = search("Title", "\"oeufs cuits\" +(tomate | patate) -frite");
```

pour rechercher les Units qui ont dans le titre la phrase “oeufs cuits” et au moins un parmi tomate ou patate, mais pas frite

- $flt, $mlt
  - Recherche « More Like This », soit par valeurs approchées
  - **$mlt : { $fields : [name1, name2], $like : like_text }** où *name1*, *name2*, ... sont les noms des champs concernés, et *like_text* un champ texte avec lequel on va comparer les différents champs fournies pour trouver des éléments “ressemblant” à la valeur fournie (il s’agit d’une recherche permettant de chercher quelque chose qui ressemble à la valeur fournie, pas l’égalité, en mode plein texte)
    - $mlt : More like this, la méthode recommandée
    - $fmt : Fuzzy like this, une autre que fournie l’indexeur mais pouvant donner plus de faux positif et qui est un assemblage de $match avec une combinaison “$or”
  - Exemple :

```
{ "$mlt" : { "$fields" : ["Title", "Description"], "$like" : "Il était une fois" } }

static include fr.gouv.vitam.common.database.builder.query.QueryHelper.*;
Query query = mlt("Il était une fois", "Title", "Description");
```

pour chercher les Units qui ont dans le titre ou la description un contenu qui s’approche de la phrase spécifiée dans $like.

7.3.2 Partie $action dans la fonction Update

- $set
  - change la valeur des champs
  - **$set : { name1 : value1, name2 : value2, ... }** où *nameX* est le nom des champs à changer avec la valeur indiquée dans *valueX*
  - **NOTA BENE** : $set admet maintenant une liste de valeur pour un champ de type tableau.
  - Exemple :

```
{ "$set" : { "Title" : "Mon nouveau titre", "Description" : "Ma nouvelle description"␣
↪}" }

static include fr.gouv.vitam.common.database.builder.query.action.UpdateActionHelper.
↪*;
Action action = set("Title", "Mon nouveau titre").add("Description", "Ma nouvelle␣
↪description");
```

qui change les champs Title et Description avec les valeurs indiquées

- $unset
  - enlève la valeur des champs
  - **$unset : [name1, name2, ...]** où *nameX* est le nom des champs pour lesquels on va supprimer les valeurs
    - Exemple :

```
{ "$unset" : [ "StartDate", "EndDate" ]" }

static include fr.gouv.vitam.common.database.builder.query.action.UpdateActionHelper.
↪*;
Action action = unset("StartDate", "EndDate");
```

qui va vider les champs indiqués de toutes valeurs

- $min, $max
  - change la valeur du champ à la valeur minimale/maximale si elle est supérieure/inférieure à la valeur précisée
  - **$min : { name : value }** où *name* est le nom du champ où si sa valeur actuelle est inférieure à *value*, sa valeur sera remplacée par celle-ci
  - **$max** idem en sens inverse, la valeur sera remplacée si l'existante est supérieure à celle indiquée
  - Exemple :

```
{ "$min" : { "MonChamp" : 3 }" }

static include fr.gouv.vitam.common.database.builder.query.action.UpdateActionHelper.
↪*;
Action action = set("Title", "Mon nouveau titre").add("Description", "Ma nouvelle␣
↪description");
```

Si MonCompteur contient 2, MonCompteur vaudra 3, mais si MonCompteur contient 4, la valeur restera inchangée

- $inc
  - incrémente/décremente la valeur du champ selon la valeur indiquée
  - **$inc : { name : value }** où *name* est le nom du champ à incrémenter de la valeur *value* passée en paramètre (positive ou négative)
  - Exemple :

```
{ "$inc" : { "MonCompteur" : -2 }" }

static include fr.gouv.vitam.common.database.builder.query.action.UpdateActionHelper.
↪*;
Action action = inc("MonCompteur", -2);
```

décrémente de 2 la valeur initiale de MonCompteur

- $rename
  - change le nom du champ
  - **$rename : { name : newname }** où *name* est le nom du champ à renommer en *newname*
  - les champs préfixés par '#' ne peuvent pas être renommés.
  - Exemple :

```
{ "$rename" : { "MonChamp" : "MonNouveauChamp" }" }

static include fr.gouv.vitam.common.database.builder.query.action.UpdateActionHelper.
↪*;
Action action = rename("MonChamp", "MonNouveauChamp");
```

où le champ MonChamp va être renommé en MonNouveauChamp

- $push, $pull
  - ajoute en fin ou retire les éléments de la liste du champ (qui est un tableau)

- **$push : { name : { $each : [value, value, ...] } }** où *name* est le nom du champ de la forme d'un tableau (une valeur peut apparaître plus dune seule fois dans le tableau) et les valeurs sont ajoutées à la fin du tableau
- **$pull** a la même signification mais inverse, à savoir qu'elle enlève du tableau les valeurs précisées si elles existent
- Exemple :

```
{ "$push" : { "Tag" : { "$each" : [ "Poisson", "Oiseau" ] } } }

static include fr.gouv.vitam.common.database.builder.query.action.UpdateActionHelper.
↪*;
Action action = push("Tag", "Poisson", "Oiseau");
```

ajoute dans le champ Tag les valeurs précisées à la fin du tableau même si elles existent déjà dans le tableau

- $add
  - ajoute les éléments de la liste du champ (unicité des valeurs)
  - **$add : { name : { $each : [value, value, ...] } }** où *name* est le nom du champ de la forme d'une MAP ou SET (une valeur ne peut apparaître qu'une seule fois dans le tableau) et les valeurs sont ajoutées, si elles n'existent pas déjà
  - **$pull** peut être utilisé pour retirer une valeur
  - Exemple :

```
{ "$add" : { "Tag" : { "$each" : [ "Poisson", "Oiseau" ] } } }

static include fr.gouv.vitam.common.database.builder.query.action.UpdateActionHelper.
↪*;
Action action = add("Tag", "Poisson", "Oiseau");
```

ajoute dans le champ Tag les valeurs précisées sauf si elles existent déjà dans le tableau

- $pop
  - ajoute ou retire un élément du tableau en première ou dernière position selon la valeur -1 ou 1
  - **$pop : { name : value }** où *name* est le nom du champ et si *value* vaut -1, retire le premier, si *value* vaut 1, retire le dernier
  - Exemple :

```
{ "$pop" : { "Tag" : -1 } }

static include fr.gouv.vitam.common.database.builder.query.action.UpdateActionHelper.
↪*;
Action action = pop("Tag", -1);
```

retire dans le champ Tag la première valeur du tableau

7.4 Exemple d'un SELECT Multi-queries

```
{
 "$roots": [ "id0" ],
 "$query": [
   { "$match": { "Title": "titre" }, "$depth": 4 },
   { "$and" : [ { "$gt" : { "StartDate" : "2014-03-25" } },
     { "$lte" : { "EndDate" : "2014-04-25" } } ], "$depth" : 0},
```

```
   { "$exists" : "FilePlanPosition" }
 ],
 "$filter": { "$limit": 100 },
 "$projection": { "$fields": { "#id": 1, "title": 1, "#type": 1, "#parents": 1, "
↪#object": 1 } }
}

include fr.gouv.vitam.common.database.builder.request.multiple.SelectMultiQuery;
static include fr.gouv.vitam.common.database.builder.query.VitamFieldsHelper.*;
static include fr.gouv.vitam.common.database.builder.query.QueryHelper.*;

Query query1 = match("Title", "titre").setDepthLimit(4);
Query query2 = and(gt("StartDate", dateFormat.parse("2014-03-25")),
      lte("EndDate", dateFormat.parse("2014-04-25")))
      .setDepthLimit(0);
Query query3 = exists("FilePlanPosition");
SelectMultiQuery select = new SelectMultiQuery().addRoots("id0")
      .addQueries(query1, query2, query3)
      .setLimitFilter(0, 100)
      .addProjection(id(), "Title", type(), parents(), object());
JsonNode json = select.getFinalSelect();
```

1. Cette requête commence avec le Unit id0. A partir de ce Unit, on cherche des Units qui sont fils avec une distance d'au plus 4 du noeud id0 et où Title contient "titre", ce qui donne une nouvelle liste d'Ids.
2. La query suivante utilise la liste d'Ids précédemment obtenue pour effectuer un filtre sur celle-ci ($depth = 0) et vérifie une condition sur StartDate et EndDate, ce qui donne une nouvelle liste d'Ids, sous-ensemble de celle obtenue en étape 1.
3. La query suivante utilise la liste d'Ids précédemment obtenue comme point de départ et cherche les fils immédiats ($depth = 1 implicite) qui vérifie la condition que FilePlanPosition, ce qui donne une nouvelle d'Ids.
4. Sur la base de cette nouvelle liste d'Ids obtenue de l'étape 3, seuls les 100 premiers sont retournés, et le contenu de ce qui est retourné est précisé dans la projection.

A noter qu'il aurait été possible d'optimiser cette requête comme suit :

```
{
 "$roots": [ "id0" ],
 "$query": [
   { "$and" : [ { "$match": { "Title": "titre" } },
     { "$gt" : { "StartDate" : "2014-03-25" } },
     { "$lte" : { "EndDate" : "2014-04-25" } } ], "$depth" : 4},
   { "$exists" : "FilePlanPosition" }
 ],
 "$filter": { "$limit": 100 },
 "$projection": { "$fields": { "#id": 1, "title": 1, "#type": 1, "#parents": 1, "
↪#object": 1 } }
}

include fr.gouv.vitam.common.database.builder.request.multiple.SelectMultiQuery;
static include fr.gouv.vitam.common.database.builder.query.VitamFieldsHelper.*;
static include fr.gouv.vitam.common.database.builder.query.QueryHelper.*;

Query query2 = and(match("Title", "titre"), gt("StartDate", dateFormat.parse("2014-03-
↪25")),
      lte("EndDate", dateFormat.parse("2014-04-25"))).setDepthLimit(4);
Query query3 = exists("FilePlanPosition");
SelectMultiQuery select = new SelectMultiQuery().addRoots("id0")
```

```
      .addQueries(query2, query3)
      .setLimitFilter(0, 100)
      .addProjection(id(), "Title", type(), parents(), object());
JsonNode json = select.getFinalSelect();
```

Car la requête 1 et 2 sont unifiées en une seule.

7.5 Exemple de scénarios

7.5.1 Cas du SIP Mercier.zip

Etape 1

1. je cherche l'article 2 (ArchivalAgencyArchiveUnitIdentifier) = les discours prononcés devant l'Assemblée nationale

```
{
  "$roots": [],
  "$query": [
        {
          "$match": {
            "Title": "assemblée"
          },
          "$depth": 20
        },
        {
          "$match": {
            "Title": "discours"
          },
          "$depth": 20
        }
      ]
    }
  ],
  "$filter": {
    "$orderby": {
      "TransactedDate": 1
    }
  },
  "$projection": {
    "$fields": {

    }
  }
}
```

Etape 2

2. je cherche les discours prononcés lors de la préparation de la loi relative au défenseur des droits, que ce soit à l'Assemblée nationale ou le Sénat (Title = défenseur)

```
{
  "$roots": [],
  "$query": [
    {
      "$or": [
```

```
        {
          "$match": {
            "Title": "sénat"
          }
        },
        {
          "$match": {
            "Title": "assemblée"
          }
        }
      ],
      "$depth": 20
    },
    {
      "$and": [
        {
          "$match": {
            "Title": "défenseur"
          }
        }
      ],
      "$depth": 20
    }
  ],
  "$filter": {
    "$orderby": {
      "TransactedDate": 1
    }
  },
  "$projection": {
    "$fields": {
    }
  }
}
```

Etape 3

3. je cherche dans le dossier Sénat (Title = Sénat), les discours prononcés lors de la relative au défenseur des droits (Title = défenseur)

```
{
  "$roots": [],
  "$query": [
    {
      "$and": [
        {
          "$eq": {
            "Title": "Sénat"
          }
        }
      ],
      "$depth": 20
    },
    {
      "$and": [
        {
          "$match": {
            "Title": "défenseur"
          }
```

```
        }
      ],
      "$depth": 20
    }
  ],
  "$filter": {
    "$orderby": {
      "TransactedDate": 1
    }
  },
  "$projection": {
    "$fields": {
    }
  }
}
```

Etape 4

4. je cherche les discours prononcé sur telle intervalle de date (StartDate, EndDate)

```
{
  "$roots": [],
  "$query": [
      {
      "$or": [
        {
          "$match": {
            "Title": "discours"
          }
        }
      ],
      "$depth": 20
    },
    {
      "$and": [
        { "$range" : { "StartDate" : { "$gte" : "2012-10-22", "$lte" : "2012-11-07" } } },
        { "$range" : { "EndDate" : { "$gte" : "2012-11-07", "$lte" : "2012-11-08" } } }
      ],
      "$depth": 0
    }
  ],
  "$filter": {
    "$orderby": {
      "TransactedDate": 1
    }
  },
  "$projection": {
    "$fields": {

    }
  }
}
```

7.5.2 Cas du SIP 1069_OK_RULES_COMPLEXE_COMPLETE.zip

Etape 1

1. je cherche l'AU dont le titre est Botzaris (Title = Botzaris)

```
{
  "$roots": [],
  "$query": [
        {
          "$match": {
            "Title": "Botzaris"
          },
          "$depth": 20
        }
      ]
    }
  ],
  "$filter": {
    "$orderby": {
      "TransactedDate": 1
    }
  },
  "$projection": {
    "$fields": {

   }
  }
}
```

Etape 2

2. je cherche les AU qui ne seront pas communicables au 01/01/2018 (= les AU qui ont une AccesRule avec une EndDate postérieure au 01/01/2018)

```
{
  "$roots": [],
  "$query": [
    {
      "$or": [
        {
          "$gt": {
            "#management.AccessRule.EndDate": "2018-01-01"
          }
        }
      ],
      "$depth": 0
    }
  ],
  "$filter": {
    "$orderby": {
      "TransactedDate": 1
    }
  },
  "$projection": {
    "$fields": {
      "#rules" : 1, "Title" : 1
    }
  }
}
```

Etape 3

3. je cherche les AU qui ont une AppraisalRule avec sort final = Destroy

```
{
  "$roots": [],
  "$query": [
    {
      "$or": [
        {
          "$eq": {
            "#management.AppraisalRule.FinalAction": "Destroy"
          }
        }
      ],
      "$depth": 0
    }
  ],
  "$filter": {
    "$orderby": {
      "TransactedDate": 1
    }
  },
  "$projection": {
    "$fields": {
      "#rules" : 1, "Title" : 1
    }
  }
}
```

CHAPITRE 8

Utilisation des clients externes

8.1 Clients externes

Pour faciliter l'accès aux API externes, le projet VITAM met à disposition les clients externes Java correspondant.

Astuce : Le code d'ihm-demo est un bon exemple d'utilisation des clients présentés ci-dessous.

8.1.1 Client Ingest

Le client Java des API ingest externes a les coordonnées maven suivantes :

```
<dependency>
        <groupId>fr.gouv.vitam</groupId>
        <artifactId>ingest-external-client</artifactId>
        <version>${vitam.version}</version>
</dependency>
```

La configuration du client est à réaliser conformément au paragraphe *Configuration d'un client externe* (page 52) ; le fichier de configuration dédié à l'API d'ingest externe est le fichier `ingest-external-client.conf` :

```
serverHost: {{vitam_ingestexternal_host}}
serverPort: {{vitam_ingestexternal_port_https}}
secure: true
sslConfiguration :
 keystore :
  - keyPath: {{vitam_folder_conf}}/keystore_{{ vitam_component }}.p12
    keyPassword: {{keystores.client_external.ihm_demo}}
 truststore :
  - keyPath: {{vitam_folder_conf}}/truststore_{{ vitam_component }}.jks
    keyPassword: {{truststores.client_external}}
hostnameVerification: true
```

Une instance de client se récupère grâce au code suivant :

```
import fr.gouv.vitam.ingest.external.client
IngestExternalClient client = IngestExternalClientFactorty.getInstance().getClient()
```

Pour la suite, se référer à la javadoc de la classe `IngestExternalClient`.

8.1.2 Client Access

Le client Java des API access externes a les coordonnées maven suivantes :

```
<dependency>
        <groupId>fr.gouv.vitam</groupId>
        <artifactId>access-external-client</artifactId>
        <version>${vitam.version}</version>
</dependency>
```

La configuration du client est à réaliser conformément au paragraphe *Configuration d'un client externe* (page 52) ; le fichier de configuration dédié à l'API d'access externe est le fichier `access-external-client.conf` :

```
serverHost: {{vitam_accessexternal_host}}
serverPort: {{vitam_accessexternal_port_https}}
secure: true
sslConfiguration :
 keystore :
  - keyPath: {{vitam_folder_conf}}/keystore_{{ vitam_component }}.p12
    keyPassword: {{keystores.client_external.ihm_demo}}
 truststore :
  - keyPath: {{vitam_folder_conf}}/truststore_{{ vitam_component }}.jks
    keyPassword: {{truststores.client_external}}
hostnameVerification: true
```

Une instance de client se récupère grâce au code suivant :

```
fr.gouv.vitam.access.external.client
AccessExternalClient client = AccessExternalClientFactorty.getInstance().getClient()
```

Pour la suite, se référer à la javadoc de la classe `AccessExternalClient`.

8.1.3 Configuration d'un client externe

La configuration du client prend en compte les paramètres et fichiers suivants :

- La propriété système Java `vitam.conf.dir` : elle indique le répertoire dans laquelle les fichiers de configuration des clients seront recherchés (ex de déclaration en ligne de commande : `-Dvitam.config.folder=/vitam/conf/clientvitam`) ;
- Le fichier de configuration (`<api>-client.conf`) : il doit être présent dans le répertoire défini précédemment ; c'est un fichier de configuration qui contient notamment les éléments de configuration suivants :
  - `serverHost` et `serverPort` permettent d'indiquer l'hôte et le port du serveur hébergeant l'API externe ;
  - keystore : `keyPath` et `keyPassword` permettent d'indiquer le chemin et le mot de passe du magasin de certificats contenant le certificat client utilisé par le client externe pour s'authentifier auprès de l'API externe ;
  - trusstore : `keyPath` et `keyPassword` permettent d'indiquer le chemin et le mot de passe du magasin de certificats contenant les certificats des autorités de certification requise (i.e. AC des certificats client et serveur).