

VITAM - Manuel Intégration Applicative

Version 10.0.0

VITAM

nov. 15, 2026

Table des matières

1 VITAM **1**
- 1.1 Architecture générale . . . 1
- 1.2 Architecture des flux . . . 2

2 API **3**
- 2.1 Formation générale des API externes . . . 3
  - 2.1.1 Services . . . 3
  - 2.1.2 Quelques Ressources . . . 3
  - 2.1.3 Format . . . 3
- 2.2 Clients d'appels Java . . . 3

3 Exemples **5**
- 3.1 Recherche d'unités archivistiques par ArchivalAgencyArchiveUnitIdentifier . . . 5
- 3.2 Recherche de registre de fonds par producteur (FRAN_NP_005568) . . . 6
- 3.3 Recherche d'unités archivistiques par titre AND description AND dates . . . 7
- 3.4 Recherche d'unités archivistiques par libre titre OR description . . . 8

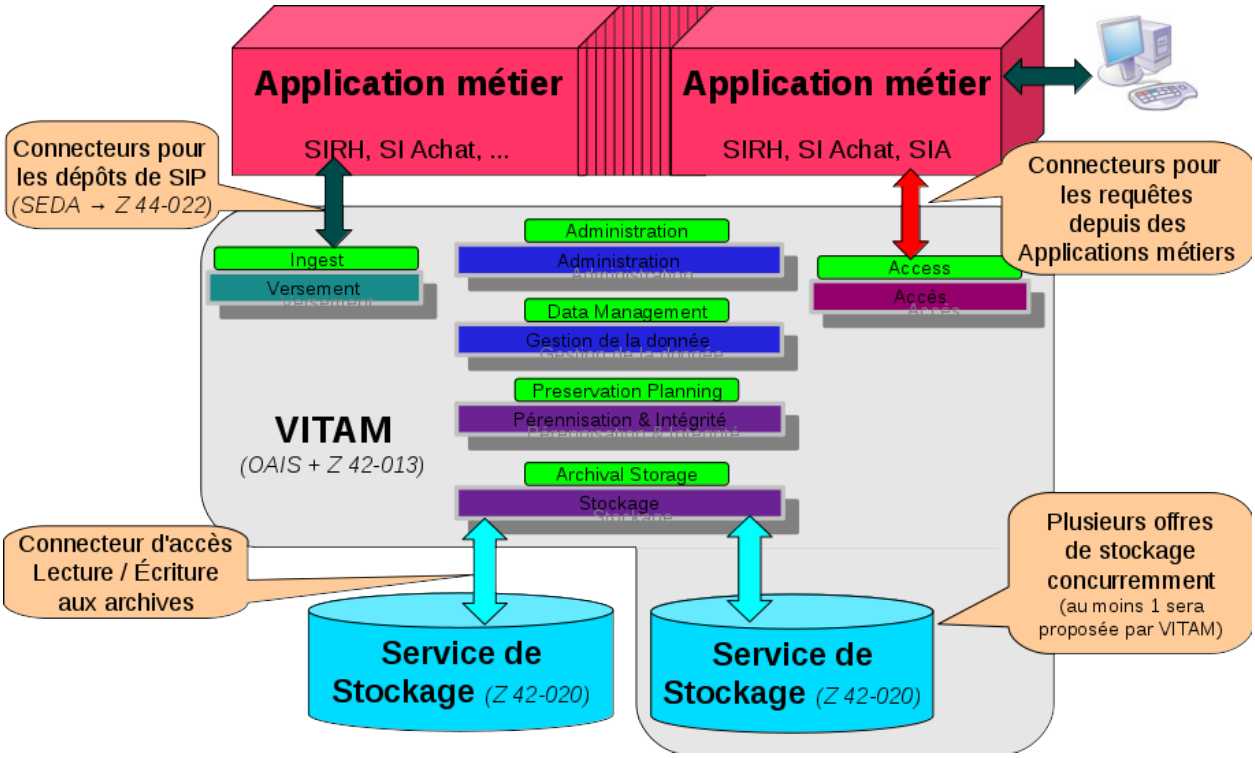
4 DSL Java Vitam **10**
- 4.1 Génération de requêtes DSL en Java . . . 10
- 4.2 Exemples d'usages du DSL . . . 12
  - 4.2.1 Partie $query . . . 12
  - 4.2.2 Partie $action dans la fonction Update . . . 13

5 Utilisation des clients externes **14**
- 5.1 Client Ingest . . . 14
- 5.2 Client Access . . . 15
  - 5.2.1 Access . . . 15
  - 5.2.2 Admin . . . 16
- 5.3 Configuration d'un client externe . . . 16

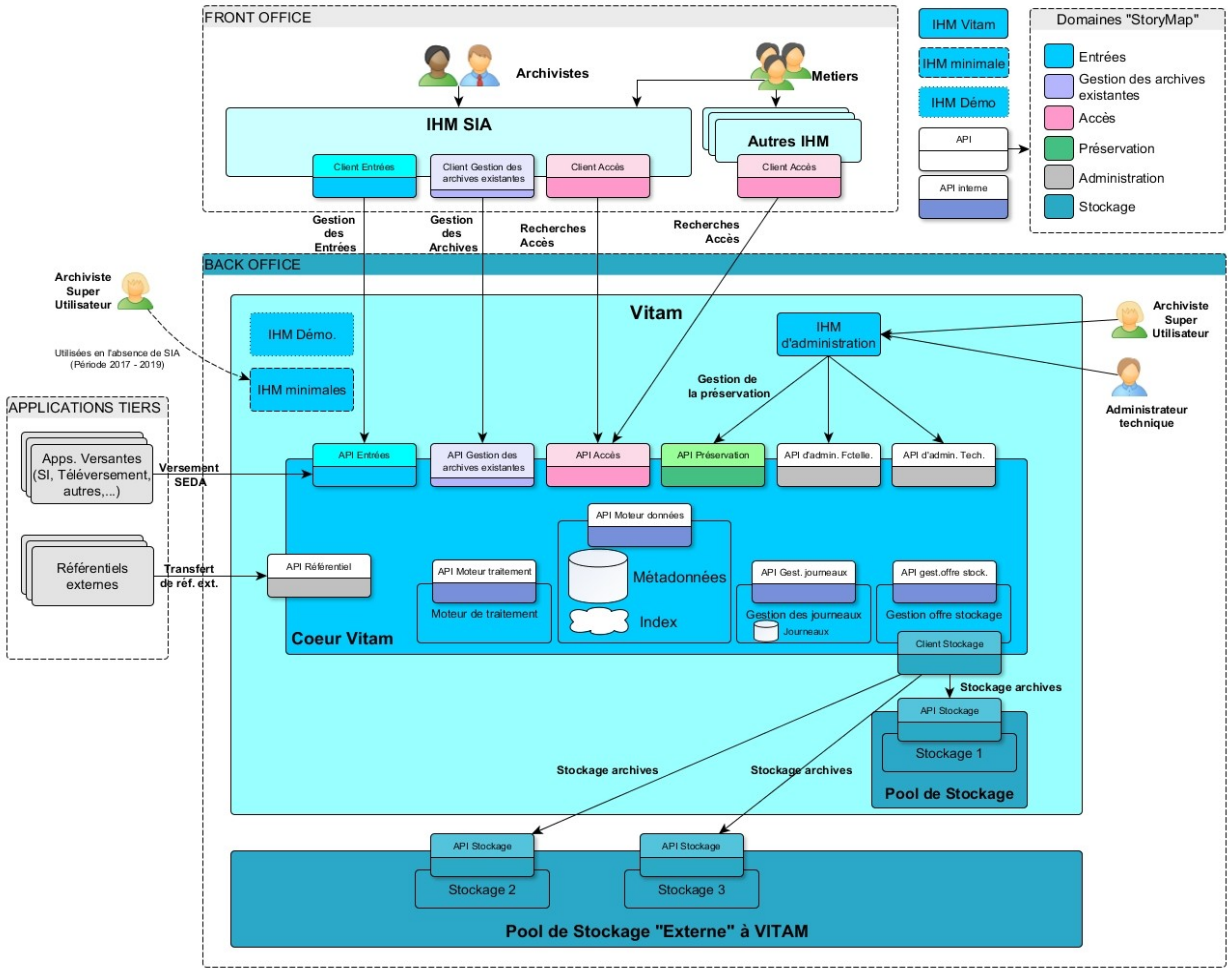
CHAPITRE 1

VITAM

1.1 Architecture générale



1.2 Architecture des flux



CHAPITRE 2

API

2.1 Formation générale des API externes

2.1.1 Services

- ingest-external : Opérations d'entrées
- access-external : Opérations d'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 Format

```
POST   /access-external    /v1       /units
VERB   Endpoint            Version   Ressource
```

La documentation des API REST décrit en détail les endpoints, les conventions d'appels ainsi que le language de requêtes DSL.

2.2 Clients d'appels Java

Vitam est livré avec des clients d'appels externes en Java. Ils sont notamment accessibles depuis les packages des clients suivants :

- Ingest External Client : `fr.gouv.vitam.ingest.external.client`
- Access External Client : `fr.gouv.vitam.access.external.client`

De plus, plusieurs helpers sont disponibles pour la construction des requêtes DSL dans `common/common-database-vitam/common-database-public` :

- 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**

La documentation JavaDoc décrit en détail les API clientes Java.

CHAPITRE 3

Exemples

3.1 Recherche d'unités archivistiques par ArchivalAgencyArchiveUnitIdentifier

EndPoint : /access-external/v1/units

Client Java

```
try (AccessExternalClient client = AccessExternalClientFactory.getInstance().
↪getClient()) {
      Integer tenantId = 0; // à titre d'exemple
      String contract = "myContract"; // à titre d'exemple
      final String selectQuery = "{\"$query\": [{\"$eq\": {\
↪"ArchivalAgencyArchiveUnitIdentifier\" : \"20130456/3\"}}]}";
      final JsonNode queryJson = JsonHandler.getFromString(selectQuery);
      client.selectUnits(new VitamContext(tenantId).setAccessContract(contract),␣
↪queryJson);
  } catch (InvalidParseOperationException | VitamClientException e) {
      ///Log ...
}
```

Client Java avec construction DSL

EndPoint : access-external/v1/units

```
JsonNode queryDsql = null;
Integer tenantId = 0; // à titre d'exemple
String contract = "myContract"; // à titre d'exemple
try (AccessExternalClient client = AccessExternalClientFactory.getInstance().
↪getClient()) {
    Query query = QueryHelper.eq("ArchivalAgencyArchiveUnitIdentifier", "20130456/3");
    SelectMultiQuery select = new SelectMultiQuery()
      .addQueries(query)
      .setLimitFilter(0, 100);
```

(suite sur la page suivante)

(suite de la page précédente)

```
    client.selectUnits(new VitamContext(tenantId).setAccessContract(contract), select.
↪getFinalSelect());
} catch (InvalidCreateOperationException | InvalidParseOperationException |␣
↪VitamClientException e) {
    ///Log ...
}
```

Postman

POST /access-external/v1/units

Indiquer pour la requête POST :

- Header :
- X-Http-Method-Override : GET
- X-Tenant-Id : 0
- X-Access-Contract-Id : myContract
- Accept : application/json
- Content-Type : application/json
- Body :

```
{
  "$roots": [],
  "$query": [
    {
      "$eq": {
        "ArchivalAgencyArchiveUnitIdentifier": "20130456/3"
      }
    }
  ],
  "$filter": {},
  "$projection": {}
}
```

3.2 Recherche de registre de fonds par producteur (FRAN_NP_005568)

Client Java

Endpoint : /admin-external/v1/accessionregisters

```
Integer tenantId = 0; // à titre d'exemple
String contract = "myContract"; // à titre d'exemple
final String queryDsl = "{\"$query\": [{\"$eq\": {\"OriginatingAgency\" : \"FRAN_NP_
↪005568\"}}]}";
try (AdminExternalClient client = AdminExternalClientFactory.getInstance().
↪getClient()) {
    final JsonNode queryJson = JsonHandler.getFromString(queryDsl);
    client.findAccessionRegister(new VitamContext(tenantId).
↪setAccessContract(contract), queryJson);
} catch (InvalidParseOperationException | VitamClientException e) {
    // LOG
}
```

Client Java avec construction DSL

Endpoint : /admin-external/v1/accessionregisters

```
Integer tenantId = 0; // à titre d'exemple
String contract = "myContract"; // à titre d'exemple=
Select select = new Select();
try (AdminExternalClient client = AdminExternalClientFactory.getInstance().
↪getClient()) {
    select.setQuery(QueryHelper.eq("OriginatingAgency", "FRAN_NP_005568"));
    client.findAccessionRegister(new VitamContext(tenantId).
↪setAccessContract(contract),
        select.getFinalSelect());
} catch (VitamClientException | InvalidCreateOperationException e) {
    // LOG
}
```

Postman

POST /admin-external/v1/accessionregisters

Indiquer pour la requête POST :

- Header :
- X-Http-Method-Override : GET
- X-Tenant-Id : 0
- X-Access-Contract-Id : myContract
- Accept : application/json
- Content-Type : application/json
- Body

```
{
  "$query" : {
      "$eq" : { "OriginatingAgency" : "FRAN_NP_005568" }
  },
  "$filter":{},
  "$projection":{}
}
```

3.3 Recherche d'unités archivistiques par titre AND description AND dates

Client Java

Endpoint : /access-external/v1/units

```
Integer tenantId = 0; // à titre d'exemple
String contract = "myContract"; // à titre d'exemple
Select select = new Select();
try (AccessExternalClient client = AccessExternalClientFactory.getInstance().
↪getClient()) {
    MatchQuery titleQ = QueryHelper.match("Title", "myTitle");
    CompareQuery dateQ = QueryHelper.eq("StartDate", "2015-07-24T02:15:28.28Z");
```

(suite sur la page suivante)

(suite de la page précédente)

```
    MatchQuery descQ = QueryHelper.match("Description", "myDescription");
    select.setQuery(QueryHelper.and().add(titleQ, dateQ, descQ));
    client.selectUnits(new VitamContext(tenantId).setAccessContract(contract), select.
↪getFinalSelect());
} catch (InvalidCreateOperationException | VitamClientException e) {
    ///Log ...
}
```

Postman

GET /access-external/v1/units Indiquer pour la requête POST :

- Header :
- X-Http-Method-Override : GET
- X-Tenant-Id : 0
- X-Access-Contract-Id : myContract
- Accept : application/json
- Content-Type : application/json
- Body :

```
{
  "$roots": [],
  "$query": [
    {
      "$and": [
        {
          "$match": {
            "Title" : "myTitle"
          }
        },
        {
          "$match": {
            "Description" : "myDescription"
          }
        },
        {
          "$eq" : {
            "StartDate" : "2015-07-24T02:15:28.28Z"
          }
        }
      ]
    }
  ],
  "$filter": {},
  "$projection": {}
}
```

3.4 Recherche d'unités archivistiques par libre titre OR description

Client Java

Endpoint : /access-external/v1/units

```
Integer tenantId = 0; // à titre d'exemple
String contract = "myContract"; // à titre d'exemple
Select select = new Select();
try (AccessExternalClient client = AccessExternalClientFactory.getInstance().
↪getClient()) {
    MatchQuery titleQ = QueryHelper.match("Title", "myTitle");
    MatchQuery descQ = QueryHelper.match("Description", "myDescription");
    select.setQuery(QueryHelper.or().add(titleQ, descQ));
    client.selectUnits(new VitamContext(tenantId).setAccessContract(contract), select.
↪getFinalSelect());
} catch (InvalidCreateOperationException | VitamClientException e) {
    ///Log ...
}
```

Postman

GET /access-external/v1/units

Indiquer pour la requête POST :

- Header :
- X-Http-Method-Override : GET
- X-Tenant-Id : 0
- X-Access-Contract-Id : myContract
- Accept : application/json
- Content-Type : application/json
- Body :

```
{
  "$roots": [],
  "$query": [
    {
      "$or": [
        {
          "$match": {
            "Title" : "myTitle"
          }
        },
        {
          "$match": {
            "Description" : "myDescription"
          }
        }
      ]
    }
  ],
  "$filter": {},
  "$projection": {}
}
```

CHAPITRE 4

DSL Java Vitam

Cette partie va essayer de montrer quelques exemples d'usages du DSL à l'aide de la librairie DSL Java Vitam dans différentes conditions.

4.1 Génération de requêtes DSL en Java

Les clients externes java Vitam offrent la possibilité de créer les requêtes DSL à partir des librairies DSL. Il existent 4 types de requêtes DSL au format Json :

- requêtes DSL de recherche (SELECT SINGLE)
- requêtes DSL de recherche de type graphe (SELECT MULTIPLE) **EXPERIMENTAL**
- requête DSL d'accès unitaire (GET BY ID) qui peut se générer de deux manières différentes
- requête DSL de modification unitaire (UPDATE BY ID) qui peut se générer de deux manières différentes

Pour le choix de la requête nécessaire, se référer à la document de l'API rest Vitam. Exemples de code de génération :

- requête DSL graphe pour recherche sur métadonnées : Select Multi Query (collections multi-query : Unit et Objects)

```
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 = exists("FilePlanPosition").setDepthLimit(3);
SelectMultiQuery select = new SelectMultiQuery().addRoots("id0")
      .addQueries(query1, query2)
      .setLimitFilter(0, 100)
      .addProjection(id(), "Title", type(), parents(), object());
JsonNode json = select.getFinalSelect();
```

- requête DSL unitaire d'accès pour les métadonnées : Select By Id (collections multi-query : Unit et Objects)

```
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.*;

SelectMultiQuery select = new SelectMultiQuery()
      .addProjection(id(), "Title");
JsonNode json = select.getFinalSelectById();
```

- requête DSL graphe pour recherche sur les données référentiel et logbook : Select Single Query

```
include fr.gouv.vitam.common.database.builder.request.single.Select;
static include fr.gouv.vitam.common.database.builder.query.VitamFieldsHelper.*;
static include fr.gouv.vitam.common.database.builder.query.QueryHelper.*;

Query query = eq("Identifier", "ID");
Select select = new Select()
      .setQuery(query)
      .setLimitFilter(0, 100)
      .addProjection();
JsonNode json = select.getFinalSelect();
```

- requête DSL unitaire d'accès pour les données référentiel et logbook : Select By Id

```
include fr.gouv.vitam.common.database.builder.request.single.Select;
static include fr.gouv.vitam.common.database.builder.query.VitamFieldsHelper.*;
static include fr.gouv.vitam.common.database.builder.query.QueryHelper.*;

Select select = new Select()
      .addProjection(id(), "Name");
JsonNode json = select.getFinalSelectById();
```

- requête DSL de modification unitaire pour les métadonnées : Update By Id (collection multi-query : Unit et Objects)

```
include fr.gouv.vitam.common.database.builder.request.multiple.UpdateMultiQuery;
static include fr.gouv.vitam.common.database.builder.query.VitamFieldsHelper.*;
static include fr.gouv.vitam.common.database.builder.query.action.UpdateActionHelper.
↪*;

Action action = set("Description", "Ma nouvelle description");
UpdateMultiQuery update = new UpdateMultiQuery()
        .addAction(action);
JsonNode json = update.getFinalUpdateById();
```

- requête DSL de modification unitaire pour les données référentiel et logbook : Update By Id (collection single)

```
include fr.gouv.vitam.common.database.builder.request.single.Update;
static include fr.gouv.vitam.common.database.builder.query.VitamFieldsHelper.*;
static include fr.gouv.vitam.common.database.builder.query.action.UpdateActionHelper.
↪*;

Action action = set("Name", "Mon nouveau nom");
Update update = new Update().addActions(action);
JsonNode json = update.getFinalUpdateById();
```

4.2 Exemples d'usages du DSL

4.2.1 Partie $query

- $and, $or, $not

```
{ "$and" : [ { "$gte" : { "StartDate" : "2014-03-23T00:00:00" } }, { "$lt" : {
↪"StartDate" : "2014-04-23T00:00:00" } } ] }

static include fr.gouv.vitam.common.database.builder.query.QueryHelper.*;
Query query = and().add(gte("StartDate", dateFormat.parse("2014-03-23T00:00:00")),
         lt("StartDate", dateFormat.parse("2014-04-23T00:00:00"));
```

- $eq, $ne, $lt, $lte, $gt, $gte

```
{ "$gte" : { "StartDate" : "2014-03-23T00:00:00" } }

static include fr.gouv.vitam.common.database.builder.query.QueryHelper.*;
Query query = gt("StartDate", dateFormat.parse("2014-03-23T00:00:00"));
```

- $range

```
{ "$range" : { "StartDate" : { "$gte" : "2014-03-23T00:00:00", "$lt" : "2014-04-
↪23T00:00:00" } } }

static include fr.gouv.vitam.common.database.builder.query.QueryHelper.*;
Query query = range("StartDate", dateFormat.parse("2014-03-23T00:00:00"), true,
      dateFormat.parse("2014-04-23T00:00:00"), true);
```

- $exists

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

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

- $in, $nin

```
{ "$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");
```

- $wildcard

```
{ "$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");
```

- $match, $match_all, $match_phrase, $match_phrase_prefix

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

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

```
{ "$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");
```

- $regex

```
{ "$regex" : { "Identifier" : "AC*" } }

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

- $search

```
{ "$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");
```

4.2.2 Partie $action dans la fonction Update

- $set

```
{ "$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");
```

- $unset

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

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

CHAPITRE 5

Utilisation des 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.

5.1 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 16) ; le fichier de configuration dédié à l'API d'ingest externe est le fichier `ingest-external-client.conf` :

```
serverHost: {{ vitam.ingest_external.host }}
serverPort: {{ vitam.ingest_external.port_service }}
secure: true
sslConfiguration:
  keystore:
    - keyPath: {{ vitam_folder_conf }}/keystore_{{ vitam_struct.vitam_component }}.p12
      keyPassword: {{ keystores.client_external.ihm_demo }}
  truststore:
    - keyPath: {{ vitam_folder_conf }}/truststore_{{ vitam_struct.vitam_component }}.
↪jks
      keyPassword: {{ truststores.client_external }}
hostnameVerification: true
```

Le fichier définitif doit s'appeler `ingest-external-client.conf` et doit être placé dans le répertoire `/vitam/conf` ou le répertoire défini par la surconfiguration du chemin de configuration par l'argument passé à la JVM `-Dvitam.config.folder=/monchemin` où `monchemin` est le lieu où se trouve ce fichier de configuration.

Une instance de client se récupère grâce au code suivant :

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

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

5.2 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 16) ; le fichier de configuration dédié à l'API d'access externe est le fichier `access-external-client.conf` :

```
serverHost: {{ vitam.access_external.host }}
serverPort: {{ vitam.access_external.port_service }}
secure: true
sslConfiguration:
  keystore:
    - keyPath: {{ vitam_folder_conf }}/keystore_{{ vitam_struct.vitam_component }}.p12
      keyPassword: {{ keystores.client_external.ihm_demo }}
  truststore:
    - keyPath: {{ vitam_folder_conf }}/truststore_{{ vitam_struct.vitam_component }}.
↪jks
      keyPassword: {{ truststores.client_external }}
hostnameVerification: true
```

Le fichier définitif doit s'appeler `access-external-client.conf` et placé dans le répertoire par défaut `/vitam/conf` ou le répertoire définit par la surconfiguration du chemin de configuration par l'argument passé à la JVM `-Dvitam.config.folder=/monchemin` où `monchemin` est le lieu où se trouve ce fichier de configuration.

5.2.1 Access

Une instance de client se récupère grâce au code suivant :

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

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

5.2.2 Admin

Une instance de client se récupère grâce au code suivant :

```
fr.gouv.vitam.access.external.client
AdminExternalClient client = AdminExternalClientFactory.getInstance().getClient()
```

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

5.3 Configuration d'un client externe

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

- La propriété système Java `vitam.config.folder` : 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`) : 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).

Le client externe peut necessiter un header pour l'authentification « X-Personal-Certificate » pour certaines resources sensibles. Ces resources sont listées dans la collection certificate de la base de données identity.