VITAM - Manuel de développement

Version 0.20.0

VITAM

juil. 21, 2017

Table des matières

CHAPITRE 1

Détails par composant

Les sections qui suivent donnent une description plus fine de l'architecture interne des services VITAM.

1.1 Access

1.1.1 Introduction

1.1.1.1 But de cette documentation

L'objectif de cette documentation est de compléter la Javadoc pour ce module.

1.1.2 Composant Access

1.1.3 Utilisation

1.1.3.1 Configuration

Le module d'access est configuré par un POM qui contient les informations nécessaires (nom du projet, numéro de version, identifiant du module parent, les sous modules (common, api, core, rest, client) de sous module d'access, etc..). Ces informations sont contenues dans le fichier pom.xml présent dans le répertoire de base du module Access.

```
  <parent>
      <groupId>fr.gouv.vitam</groupId>
      <artifactId>parent</artifactId>
      <version>0.5.0-SNAPSHOT</version>
  </parent>

  <artifactId>access</artifactId>
  <packaging>pom</packaging>

  <modules>
      <module>access-common</module>
      <module>access-api</module>
      <module>access-core</module>
      <module>access-rest</module>
      <module>access-client</module>
</modules>
```

1.1.3.2 La factory

Afin de récupérer le client une factory a été mise en place.

```
// Récupération du client
final AccessClient client = AccessClientFactory.getInstance().
↪getAccessOperationClient();
```

1.1.3.2.1 Le Mock

Par défaut, le client est en mode Mock. Il est possible de récupérer directement le mock. Si les paramètres de productions sont introuvables, le client passe en mode Mock par défaut. Il est possible de récupérer directement le mock :

```
    // Changer la configuration du Factory client
    AccessClientFactory.setConfiguration(AccessClientType.MOCK);

    // Récupération explicite du client mock
      final AccessClient client = AccessClientFactory.getInstance().
↪getAccessOperationClient();

- Pour instancier son client en mode Production :
```

```
// Changer la configuration du Factory
 AccessClientFactory.setConfiguration(AccessClientType.PRODUCTION);
// Récupération explicite du client
AccessClient client = AccessClientFactory.getInstance().getAccessOperationClient();
```

1.1.3.3 L'application rest

La méthode run avec l'argument de port permet aux tests unitaires de démarrer sur un port spécifique. Le premier argument contient le nom du fichier de configuration access.conf (il est templatiser avec ansible)

1.1.3.4 Le client

Le client propose actuellement plusieurs méthodes : selectUnits(String dslQuery) ; selectUnitbyId(String sqlQuery, String id) ; updateUnitbyId(String updateQuery, String unitId) ;selectObjectbyId(String selectObjectQuery, String objectId) ; getObjectAsInputStream(String selectObjectQuery, String objectGroupId, String usage, int version) ;

Paramètre de la fonction : String dsl, selectUnitbyId(String sqlQuery, String id) //TODO (Itérations futures : ajouter méthode modification des métadonnées ?)

Le client récupère une réponse au format Json ou au format InputStream.

1.1.3.4.1 Exemple d'usage générique

```
// Récupération du client dans le module ihm-demo
 AccessClient client = AccessClientFactory.getInstance().getAccessOperationClient();

// Récupération du dsl ( cf ihm-demo documentation)
```

```
// Recherche des Archives Units
JsonNode selectUnits(String dsl)

// Recherche des Units par Identification
 JsonNode selectUnitbyId(String sqlQuery, String id)

 //Recherche d'object par ID + un DSL selectObjectQuery
 JsonNode jsonObject = client.selectObjectbyId(String selectObjectQuery, String id);

 //Récupération d'un objet au format input stream
 InputStream stream = client.getObjectAsInputStream(String selectObjectQuery, String␣
↪objectGroupId, String usage, int version);
```

1.1.3.4.2 Exemple d'usage générique

```
 // Récupération du client
private static final AccessClient ACCESS_CLIENT = AccessClientFactory.getInstance().
↪getAccessOperationClient();

  ...

 // Autres Opérations

 public static JsonNode searchUnits(String parameters)
         throws AccessClientServerException, AccessClientNotFoundException,␣
↪InvalidParseOperationException {
     return ACCESS_CLIENT.selectUnits(parameters);
 }
```

1.1.4 Présentation

Un filter passée dans les headers, a été ajouté pour pouvoir interdire toute requête n'indiquant pas de headerAccess-ContratId ou son contrat est inconnu sur ce tenant ou son contrat est invalide.

1.1.4.1 Classe de filtre

Une classe de filtre a été ajoutée :

AccessContratIdContainerFilter : On vérifie la présence du header X_ACCESS_CONTRAT_ID dans la requête, sinon, une réponse UNAUTHORIZED (code 401) sera retournée. Ensuite, on vérifie l'existence et la validité du contrat avec id de X_ACCESS_CONTRAT_ID, sinon, une réponse une réponse UNAUTHORIZED (code 401) sera retournée.

1.1.4.2 Implémenter des filters

Le filtre sera ajouté dans registerInResourceConfig de serveur application de access internal

```
resourceConfig.register(new AccessInternalResourceImpl(getConfiguration()))
.register(new LogbookInternalResourceImpl())
.register(AccessContratIdContainerFilter.class);
```

1.1.5 Access-rest

1.1.5.1 Présentation

API REST EXT appelées par le client access external. Il y a un controle des paramètres (SanityChecker.checkJsonAll) transmis avec ESAPI.

1.1.5.2 fr.gouv.vitam.access.external.rest

– AccessExternalRessourceImpl

1.1.5.2.1 Rest API

-Unit

GET https://vitam/access-external/v1/units récupérer la liste des units avec la filtre (le contenu de la requête)

POST https://vitam/access-external/v1/units (with X-HTTP-METHOD-OVERRIDE GET) récupérer la liste des units avec la filtre (le contenu de la requête)

PUT https://vitam/access-external/v1/units Mettre à jour la liste des units (non implémenté)

PUT https://vitam/access-external/v1/units/unit_id Mettre à jour l'unit avec avec le contenu de la requête

HEAD https://vitam/access-external/v1/units Vérifier l'existence d'un unit (non implémenté)

GET https://vitam/access-external/v1/units/unit_id récupérer l'units avec la filtre (le contenu de la requête)

POST https://vitam/access-external/v1/units/unit_id (avec X-HTTP-METHOD-OVERRIDE GET) récupérer l'units avec la filtre (le contenu de la requête)

GET https://vitam/access-external/v1/units/unit_id/object récupérer le group d'objet par un unit (le contenu de la requête)

POST https://vitam/access-external/v1/units/unit_id/object (avec X-HTTP-METHOD-OVERRIDE GET) récupérer le group d'objet par un unit (le contenu de la requête)

-ObjectGroup

GET https://vitam/access-external/v1/objects récupérer la liste des object group (non implémenté)

POST https://vitam/access-external/v1/objects (avec X-HTTP-METHOD-OVERRIDE GET) récupérer la liste des object group (non implémenté)

GET https://vitam/access-external/v1/objects/object_id récupérer une groupe d'objet avec la filtre (le contenu de la requête) et id

POST https://vitam/access-external/v1/objects/objet_id (avec X-HTTP-METHOD-OVERRIDE GET) récupérer une groupe d'objet avec la filtre (le contenu de la requête) et id

-Accession Register

POST https://vitam/admin-external/v1/accession-registers récupérer le registre de fond

POST https://vitam/admin-external/v1/accession-registers/document_id récupérer le registre de fond avec la filtre (le contenu de la requête) et id

POST https://vitam/admin-external/v1/accession-registers/document_id/accession-register-detail récupérer le détail du registre de fond avec la filtre (le contenu de la requête) et id

– LogbookRessourceImpl

1.1.5.2.2 Rest API

-Operation

GET https://vitam/access-external/v1/operations récupérer tous les journaux de l'opéraion

POST https://vitam/access-external/v1/operations (with X-HTTP-METHOD-OVERRIDE GET) récupérer tous les journaux de l'opéraion

GET https://vitam/access-external/v1/operations/{id_op} récupérer le journal de l'opéraion avec la filtre (le contenu de la requête) et id

POST https://vitam/access-external/v1/operations/{id_op} (with X-HTTP-METHOD-OVERRIDE GET) récupérer le journal de l'opéraion avec la filtre (le contenu de la requête) et id

-Cycle de vie

GET https://vitam/access-external/v1/unitlifecycles/{id_lc} récupérer le journal sur le cycle de vie d'un unit avec la filtre (le contenu de la requête) et id

GET https://vitam/access-external/v1/objectgrouplifecycles/{id_lc} récupérer le journal sur le cycle de vie d'un groupe d'objet avec la filtre (le contenu de la requête) et id

– AdminManagementResourceImpl

1.1.5.2.3 Rest API

-Format&Rule

PUT https://vitam/admin-external/v1/collection_id vérifier le format ou la règle

POST https://vitam/admin-external/v1/collection_id importer le fichier du format ou de la règle

POST https://vitam/admin-external/v1/collection_id récupérer le format ou la règle

POST https://vitam/admin-external/v1/collection_id/document_id récupérer le format ou la règle avec la filtre (le contenu de la requête) et id

– AdminManagementExternalResourceImpl

1.1.5.2.4 Rest API

-Contrat d'accès

PUT https://vitam/admin-external/v1/accesscontracts Mise à jour du contrat d'accès

-Contrat d'entrée

|PUT https://vitam/admin-external/v1/entrycontracts Mise à jour du contrat d'entrès

- Profiles

POST https://vitam/admin-external/v1/profiles Créer ou rechercher des profiles au format json (métadata). Le header X-Http-Method-Override pilote la décision entre la recherche et la création.

PUT https://vitam/admin-external/v1/profiles Importer le profile au format rng ou xsd

GET https://vitam/admin-external/v1/profiles Télécharger le profile au format rng ou xsd si le accept est un octet-stream sinon c'est une recherche de profiles au format json (métadata)

GET https://vitam/admin-external/v1/profiles/profile_id Rechercher un profile avec son id (profile_id)

POST https://vitam/admin-external/v1/profiles/profile_id Si X-Http-Method-Override égale à GET alors rechercher un profile avec son id (profile_id)

1.1.6 contrôle des flux d'accèss

Le module access-external a besoin de disposer d'une brique frontale effectuant les contrôles de sécurité pour les flux d'accès à la plateforme.

– Fournissant la terminaison TLS – Fournissant l'authentification par certificat – Un WAF applicatif permettant le filtrage d'entrée filtrant les entrées être une menace pour le système (ESAPI) – Un filtre permettant de vérifier l'existence et la cohérence du header X-Tenant-Id

```
protected void setFilter(ServletContextHandler context) throws␣
↪VitamApplicationServerException {
    if (getConfiguration().isAuthentication()) {
        File shiroFile = null;
        try {
            shiroFile = PropertiesUtils.findFile(SHIRO_FILE);
        } catch (final FileNotFoundException e) {
            LOGGER.error(e.getMessage(), e);
            throw new VitamApplicationServerException(e.getMessage());
        }
        context.setInitParameter("shiroConfigLocations", "file:" + shiroFile.
↪getAbsolutePath());
        context.addEventListener(new EnvironmentLoaderListener());
        context.addFilter(ShiroFilter.class, "/*", EnumSet.of(
            DispatcherType.INCLUDE, DispatcherType.REQUEST,
            DispatcherType.FORWARD, DispatcherType.ERROR, DispatcherType.ASYNC));
    }
    // chargemenet de la liste des tenants de l'application
    JsonNode node = JsonHandler.toJsonNode(getConfiguration().getTenants());
    context.setInitParameter(GlobalDataRest.TENANT_LIST, JsonHandler.
↪unprettyPrint(node));
    context.addFilter(TenantFilter.class, "/*", EnumSet.of(
        DispatcherType.INCLUDE, DispatcherType.REQUEST,
        DispatcherType.FORWARD, DispatcherType.ERROR, DispatcherType.ASYNC));
}
protected void registerInResourceConfig(ResourceConfig resourceConfig) {
    setServiceRegistry(new VitamServiceRegistry());
    serviceRegistry.register(AccessInternalClientFactory.getInstance())
        .register(AdminManagementClientFactory.getInstance());
    resourceConfig.register(new AccessExternalResourceImpl())
        .register(new LogbookExternalResourceImpl())
        .register(new AdminManagementExternalResourceImpl())
        .register(new AdminStatusResource(serviceRegistry))
        .register(SanityCheckerCommonFilter.class)
        .register(SanityDynamicFeature.class);
}
```

1.2 Common

1.2.1 Introduction

1.2.1.1 But de cette documentation

L'objectif de cette documentation est de compléter la Javadoc pour ce module.

1.2.1.2 Utilitaires Commons

1.2.1.2.1 FileUtil

Cet utilitaire propose quelques méthodes pour manipuler des fichiers.

Attention : les méthodes "readFile" doivent être limitées en termes d'usage au strict minimum et pour des fichiers de petites tailles.

1.2.1.2.2 LocalDateUtil

Cet utilitaire propose quelques méthodes pour manipuler des dates avec la nouvelle classe LocalDateTime.

1.2.1.2.3 ServerIdentity

Cet utilitaire propose une implémentation de la carte d'identité de chaque service/serveur.

ServerIdentity contient le ServerName, le ServerRole, le siteId, serverId et le PlatformGlobalID

Pour une JVM, un seul ServerIdentity existe.

C'est un Common Private.

Par défaut cette classe est initialisée avec les valeurs suivantes : * ServerName (String) : hostname ou UnknownHostname si introuvable * ServerRole (String) : UnknownRole * ServerId (int) : MAC adresse partielle comme entier (31 derniers bits de la MAC) * SiteId (int) : Id du site (entier entre 0 et 15) . Les serveurs de 2 régions informatiques (sites/salles) doivent avoir des Id différents * GlobalPlatformID (int) : nombre aggrégé du siteId et d'une partie du ServerId

Il est important que chaque server à son démarrage initialise les valeurs correctement.

```
ServerIdentity serverIdentity = ServerIdentity.getInstance();
serverIdentity.setName(name).setRole(role).setPlatformId(platformId);
// or
ServerIdentity.getInstance().setFromMap(map);
// or
ServerIdentity.getInstance().setFromPropertyFile(file);
```

Où name, role et platformID viennent d'un fichier de configuration par exemple.

1.2.1.2.3.1 Usage

```
ServerIdentity serverIdentity = ServerIdentity.getInstance();
String name = serverIdentity.getName();
String role = serverIdentity.getRole();
int platformId = serverIdentity.getGlobalPlatformId();
```

1.2.1.2.3.2 Les usages principaux

- GUID pour PlatformId
- Logger and Logbook pour tous les champs

1.2.1.2.4 SystemPropertyUtil

Cet utilitaire propose quelques méthodes pour manipuler les Propriétés héritées du Système, notamment celle déduites de "-Dxxxx" dans la ligne de commande Java.

Il intègre notamment : - String getVitamConfigFolder() - String getVitamDataFolder() - String getVitamLogFolder() - String getVitamTmpFolder()

Les répertoires sont par défaut : - Config = /vitam/conf - Data = /vitam/data - Log = /vitam/log - Tmp = /vitam/data/tmp

Ils peuvent être dynamiquement surchargés par une option au lancement du programme Java : - -Dvitam.config.folder=/path - -Dvitam.data.folder=/path - -Dvitam.log.folder=/path - -Dvitam.tmp.folder=/path

1.2.1.2.5 PropertiesUtils

Cet utilitaire propose quelques méthodes pour manipuler des fichiers de propriétés et notamment dans le répertoire Resources.

Il intègre notamment : - File getResourcesFile(String resourcesFile) qui retourne un File se situant dans "resources (classpath) /resourcesFile" - File findFile(String filename) qui retourne un File se situant dans l'ordre

- Chemin complet donné par resourcesFile
- Chemin complet donné par ConfigFolder + resourcesFile
- Chemin complet dans resources (classpath) /resourcesFile

- File fileFromConfigFolder(String subpath) qui retourne un File se situant dans "ConfigFolder + subpath" (non checké)
- File fileFromDataFolder(String subpath) qui retourne un File se situant dans "DataFolder + subpath" (non checké)
- File fileFromLogFolder(String subpath) qui retourne un File se situant dans "LogFolder + subpath" (non checké)
- File fileFromTmpFolder(String subpath) qui retourne un File se situant dans "TmpFolder + subpath" (non checké)

1.2.1.2.6 BaseXXX

Cet utilitaire propose quelques méthodes pour manipuler des Base16, Base32 et Base64.

1.2.1.2.7 CharsetUtils

Cet utilitaire propose quelques méthodes pour la gestion des Charset.

1.2.1.2.8 ParametersChecker

Cet utilitaire propose quelques méthodes pour gérer la validité des arguments dans les méthodes.

1.2.1.2.9 SingletonUtil

Cet utilitaire propose quelques méthodes pour obtenir des Singletons.

1.2.1.2.10 StringUtils

Cet utilitaire propose quelques méthodes pour manipuler des String.

1.2.1.3 GUID

Cf chapitre dédié.

1.2.1.4 Logging

Cf chapitre dédié.

1.2.1.5 LRU

Cet utilitaire propose une implémentation en mémoire de Cache Last Recent Used.

Il est notamment utilisé dans la partie Metadata.

Son usage doit positionner une dimension maximale et un délai avant retrait :

- Les plus anciens sont supprimés lorsque la place manque
- Les plus anciens sont supprimés lorsque la méthode **forceClearOldest**() est appelé

1.2.1.6 Digest

Cet utilitaire propose les fonctionnalités de calculs d'empreintes selon différents formats.

Cf chapitre dédié.

1.2.1.7 Json

Cet utilitaire propose les fonctionnalités de manipulation de Json en utilisant Jackson.

Ce module propose une configuration par défaut pour Vitam.

1.2.1.8 Exception

L'exception parente Vitam VitamException s'y trouve. Toutes les exceptions Vitam en héritent.

1.2.1.9 Client

Le client parent Vitam BasicClient et son implémentation des méthodes commune AbstractClient s'y trouvent. Une configuration commune SSLClientConfiguration complète le client Vitam.

1.2.2 Global Unique Identifier (GUID) pour Vitam

1.2.2.1 Spécifier ProcessId

Pour surcharger/spécifier le processId, qui par défaut prend la valeur du PID du processus Java, il faut utiliser la property suivante :

```
-Dfr.gouv.vitam.processId=nnnnn
```

Où nnnnn est un nombre entre 0 et 2^22 (4194304).

1.2.2.2 GUID Factory

Usage :

Il faut utiliser le helper approprié en fonction du type d'objet pour lequel on souhaite créer un GUID.

1.2.2.2.1 Pour la partie interne Vitam

- Obligatoire en **Interne Vitam** : le ServerIdentity doit être initialisé (inutile en mode client Vitam)

```
ServerIdentity.getInstance.setFromMap(Map);
ServerIdentity.getInstance.setFromPropertyFile(File);
ServerIdentity.getInstance.setName(String).setRole(String).setPlatformId(int);
```

- Pour un Unit et son Unit Logbook associé :

```
GUID unitGuid = GUIDFactory.newUnitGUID(tenantId);
```

- Pour un ObjectGroup et son ObjectGroup Logbook associé :

```
GUID objectGroupGuid = GUIDFactory.newObjectGroupGUID(tenantId);
// or
GUID objectGroupGuid = GUIDFactory.newObjectGroupGUID(unitParentGUID);
```

- Pour un Object et son Binary object associé :

```
GUID objectGuid = GUIDFactory.newObjectGUID(tenantId);
// or
GUID objectGuid = GUIDFactory.newObjectGUID(objectGroupParentGUID);
```

- Pour une Opération (process) :

```
GUID operationGuid = GUIDFactory.newOperationIdGUID(tenantId);
```

- Pour un Request Id (X-Request-Id) :

```
GUID requestIdGuid = GUIDFactory.newRequestIdGUID(tenantId);
```

- Pour un SIP / Manifest / Seda like informations Id :

```
GUID manifestGuid = GUIDFactory.newManifestGUID(tenantId);
```

- Pour un Logbook daily Id (Operation, Write) :

```
GUID writeLogbookGuid = GUIDFactory.newWriteLogbookGUID(tenantId);
```

- Pour un storage operation Id :

```
GUID storageOperationGuid = GUIDFactory.newStorageOperationGUID(tenantId);
```

- Pour savoir si un GUID est par défaut associé à une Règle WORM :

```
GUID storageOperationGuid.isWorm();
```

1.2.2.2.2 Pour la partie interne et public Vitam

- Pour récupérer un GUID depuis sa réprésentation :

```
GUID guid = GUIDReader.getGUID(stringGuid);
GUID guid = GUIDReader.getGUID(byteArrayGuid);
```

Où le “stringGuid” peut être dans sa forme BASE16 / BASE32 / BASE64 ou ARK.

1.2.2.3 Attention

- Personne ne devrait utiliser les helpers constructeurs directs (newUuid). * Ces méthodes sont réservées à des usages spéciaux futurs non encore définis.

1.2.3 Digest

Ce package a pour objet de permettre les calculs d’empreintes au sein de Vitam.

Les formats supportés sont :

- MD5
- SHA-1
- SHA-256
- SHA-384
- SHA-512

1.2.3.1 Usage

```
Digest digest = new Digest(DigestType.MD5);
// One of
digest.update(File);
digest.update(byte []);
digest.update(ByteBuffer);
digest.update(String);
digest.update(InputStream);
digest.update(FileChannel);

// Or using helpers
Digest digest = Digest.digest(InputStream, DigestType);
Digest digest = Digest.digest(File, DigestType);

// Get the result
byte[] bresult = digest.digest();
String sresult = digest.digestHex(); // in Hexa format
String sresult = digest.toString(); // in Hexa format

// Compare the result: Note that only same DigestType can be used
boolean same = digest.equals(digest2);
boolean same = digest.equals(bresult);
boolean same = digest.equals(sresult);
boolean same = digest.equalsWithType(bresult, DigestType); // same as equals(bresult)
boolean same = digest.equalsWithType(sresult, DigestType); // same as equals(sresult)
```

1.2.4 Logging

Tous les logiciels Vitam utilisent le logger VitamLogger instantié via VitamLoggerFactory.

1.2.4.1 Initialisation

Dans la Classe contenant la méthode **main** : si Logback n'est pas l'implémentation choisie, il faut changer le Factory.

```
// Out of **main** method
private static VitamLogger logger;


// In the **main** method
VitamLoggerFactory.setDefaultFactory(another VitamLoggerFactory);
  // Could be JdkLoggerFactory, Log4JLoggerFactory, LogbackLoggerFactory
logger = VitamLoggerFactory.getInstance(Class);
// or
logger = VitamLoggerFactory.getInstance(String);
```

Si l'implémentation est bien celle de Logback, cette initialisation peut être ignorée.

1.2.4.2 Usage

```
private static final VitamLogger LOGGER = VitamLoggerFactory.getInstance(Class);

LOGGER.debug(String messageFormat, args...);
// Note messageFormat supports argument replacement using '{}'
LOGGER.debug("Valeurs: {}, {}, {}", "value", 10, true);
// => "Valeur: value, 10, true"
```

Il est possible de changer le niveau de log :

```
VitamLoggerFactory.setLogLevel(VitamLogLevel);
```

5 niveaux de logs existent :

- TRACE : le plus bas niveau, ne devrait pas être activé en général
- DEBUG : le plus bas niveau usuel
- INFO : pour des informations explicatives ou contextuelles
- WARN : pour les points d'attentions (warning)
- ERROR : pour les erreurs

1.2.4.3 Pour l'usage interne Vitam

```
private static final VitamLogger LOGGER = VitamLoggerFactory.getInstance(Class);
static final VitamLoggerHelper LOGGER_HELPER = VitamLoggerHelper.newInstance();

LOGGER.debug(LOGGER_HELPER.format(message), args...);
// Allow special formatting and extra information to be set
```

1.2.5 JunitHelper

1.2.5.1 MongoDb or Web Server Junit Support

Si dans un Web Server Junit, il est nécessaire d'activer un service utilisant un port, et ceci afin de favoriser un parallélisme maximal des tests unitaires, il est demandé de procéder comme suit :

```
private static JunitHelper junitHelper;
private static int databasePort;
private static int serverPort;

// dans le @BeforeClass
// Créer un objet JunitHelper
junitHelper = new JunitHelper();

// Pour MongoDB (exemple)
databasePort = junitHelper.findAvailablePort();
final MongodStarter starter = MongodStarter.getDefaultInstance();
// On utilise le port
mongodExecutable = starter.prepare(new MongodConfigBuilder()
    .version(Version.Main.PRODUCTION)
    .net(new Net(databasePort, Network.localhostIsIPv6()))
    .build());
mongod = mongodExecutable.start();

// Pour le serveur web (ici Logbook)
// On initialise le mongoDbAccess pour le service
mongoDbAccess =
    MongoDbAccessFactory.create(
        new DbConfigurationImpl(DATABASE_HOST, databasePort,
            "vitam-test"));
// On alloue un port pour le serveur Web
serverPort = junitHelper.findAvailablePort();
```

```
 // On lit le fichier de configuration par défaut présent dans le src/test/resources
 File logbook = PropertiesUtils.findFile(LOGBOOK_CONF);
 // On extraie la configuration
 LogbookConfiguration realLogbook = PropertiesUtils.readYaml(logbook,␣
↪LogbookConfiguration.class);
 // On change le port
 realLogbook.setDbPort(databasePort);
 // On sauvegarde le fichier (dans un nouveau fichier différent) (static File)
 newLogbookConf = File.createTempFile("test", LOGBOOK_CONF, logbook.getParentFile());
 PropertiesUtils.writeYaml(newLogbookConf, realLogbook);

 // On utilise le port pour RestAssured
 RestAssured.port = serverPort;
 RestAssured.basePath = REST_URI;

 // On démarre le serveur
 try {
    vitamServer = LogbookApplication.startApplication(new String[] {
       // On utilise le fichier de configuration ainsi créé
        newLogbookConf.getAbsolutePath(),
        Integer.toString(serverPort)});
    ((BasicVitamServer) vitamServer).start();
 } catch (FileNotFoundException | VitamApplicationServerException e) {
    LOGGER.error(e);
    throw new IllegalStateException(
        "Cannot start the Logbook Application Server", e);
}

// Dans le @AfterClass
// On arrête le serveur
try {
    ((BasicVitamServer) vitamServer).stop();
} catch (final VitamApplicationServerException e) {
    LOGGER.error(e);
}
mongoDbAccess.close();
junitHelper.releasePort(serverPort);
// On arrête MongoDb
mongod.stop();
mongodExecutable.stop();
junitHelper.releasePort(databasePort);
// On efface le fichier temporaire
newLogbookConf.delete();
```

1.2.6 Client

1.2.6.1 But de cette documentation

Cette documentation indique comment utiliser le code commun du client Vitam pour créer son propre client Vitam.

1.2.6.2 Client Vitam

L'interface commune du client Vitam est : *fr.gouv.vitam.common.client.BasicClient*.

Elle mets à disposition les méthodes suivantes :

- la récupération du status du serveur distant auquel le client se connecte
- la récupération du chemin du serveur distant auquel le client se connecte
- l'arrêt du client

Une implémentation par défaut de ces méthodes est fournie dans la classe abstraite associée *fr.gouv.vitam.common.AbstractClient*.

Chaque client Vitam doit créer sa propre interface qui hérite de l'interface BasicClient

```
public interface MyModuleClient extends BasicClient {
        ....
}
```

Chaque client Vitam doit créer au moins deux implémentations :

- le client production

```
class MyModuleClientRest extends AbstractClient implements MyModuleClient {
        ....
}
```

- le client bouchonné (Mock)

```
class MyModuleClientMock extends AbstractClient implements MyModuleClient {
        ....
}
```

Une factory doit être mise en place pour récupérer l'instance du client adaptée. Par défaut, le client attends un fichier de configuration mymodule-client.conf. S'il n'est pas présent, le client bouchonnée est renvoyé.

```
public class MyModuleClientFactory {
        ....
}
```

Elle doit pouvoir être utilisée de la manière suivante :

```
// Retrieve the default mymodule client
MyModuleClient client = MyModuleClientFactory.getInstance().getMyModuleClient();
```

1.2.6.3 Configuration

Une classe de configuration par défaut est fournie : *fr.gouv.vitam.common.clientSSLClientConfiguration* . Elle contient les propriétés suivantes :

- **serverHost** : le nom d'hôte du serveur distant auquel le client va se connecter (Exemple : localhost)
- **serverPort** : le port du serveur distant auquel le client va se connecter (Exemple : 8082)
- **serverContextPath** : le context sur lequel est exposé le serveur distant auquel le client va se connecter (Exemple : /)
- **useSSL** : booléen permettant de spécifier si le client doit utiliser le protocole HTTP (false) ou HTTPS (true)

Un fichier de configuration nommé **mymodule-client.conf** doit être présent dans le classpath de l'application utilisant le client. Ce fichier de configuration est au format YAML et il doit contenir les propriétés définies par la classe de configuration.

Note : Actuellement le mode HTTPS n'est pas encore implémenté. Ainsi une runtime exception est lancée si le client est instancié avec une configuration dont le **useSSL** vaut true.

1.2.7 DirectedCycle

Vitam utilise DirectedCycle pour verifier la structure des arbres et de s'assurer qu'on n' a pas un cycle dans le graphe.

1.2.7.1 Initialisation

Pour initialiser un objet DirectedCycle, il faut instancier un objet DirectedGraph à partir d'un fichier Json (vous trouvrez ci-dessous un exemple).

```
File file = PropertiesUtils.getResourcesFile("ingest_acyc.json");
JsonNode json = JsonHandler.getFromFile(file);
DirectedGraph g = new DirectedGraph(json);
DirectedCycle graphe = new DirectedCycle(g);
```

1.2.7.2 Usage

Pour vérifier la présence d'un cycle dans le graphe

```
graphe..isCyclic() ;
```

La méthode isCyclic return true si on a un cycle.

Exemple de fichier json : ingest_acyc.json

{ "ID027" : { }, "ID028" : { "_up" : ["ID027"]}, "ID029" : {"_up" : ["ID028"]}, "ID030" : {"_up" : ["ID027"]},"ID032" : {"_up" : ["ID030", "ID029"] }, "ID031" : {"_up" : ["ID027"]}

}

1.2.7.3 Remarque

Pour Vitam, fonctionnellement il ne faut pas trouver des cycles au niveau des arbres des units. (au niveau du bordereau)

1.2.8 Graph

Vitam utilise le Graphe pour determiner l'ordre d'indexation en se basant sur la notion de chemin le plus long (longest path)

1.2.8.1 Initialisation

Pour initialiser un objet Graph :

```
File file = PropertiesUtils.getResourcesFile("ingest_tree.json");
JsonNode json = JsonHandler.getFromFile(file);
Graph graph = new Graph(json);
```

1.2.8.2 Usage

Pour determiner l'ordre il faut avoir le chemin le plus long par rapport aux différentes racines :

```
graph.getGraphWithLongestPaths()
```

La méthode getGraphWithLongestPaths return un map qui contient l'ordre on key et la liste (Set) des units id en valeur

Exemple de resultat :

{0=[ID027], 1=[ID030, ID031, ID028], 2=[ID029], 3=[ID032]}

1.2.9 Code d'erreur Vitam

Afin d'harmoniser les erreurs un code d'erreur commun aux différents modules Vitam a été défini. Il est composé de trois éléments aplhanumériques à deux caractères.

Exemple : 0A7E08 où 0A est le code service, 7E est le code domaine et 08 l'item.

1.2.9.1 Les codes

1.2.9.1.1 Code service

Le code service identifie le service concerné par l'erreur. Tous les services sont listés dans l'énum ServiceName. On y retrouve son code et sa description.

Attention, le code 00 est utilisé dans le cas où le service concerné ne se trouve pas dans l'énum. Il sert également aux différents test, il ne faut pas le supprimer.

L'énum offre également la possibilité de retrouver un service via son code (getFromCode(String code)).

1.2.9.1.2 Code domaine

Le code domaine identifie le domaine concerné par l'erreur. Tous les domaines actuellement identifiés sont listés dans l'énum DomainName. On y retrouve son code et sa description.

Attention, le code 00 est uniquement utilisé dans les tests. Il ne doit pas être utilisé dans le code de Vitam. Il ne doit pas êre supprimé.

L'énum offre également la possiblité de retrouver un domaine via son code (getFromCode(String code)).

1.2.9.1.3 Code Vitam

Le code Vitam est donc composé du service, du domaine et d'un item. On retrouve les erreurs Vitam dans l'énum CodeVitam. On y voit le service, le domaine, l'item, le status HTTP associé à cette erreur ainsi qu'un message.

A terme, le message sera une clef de traduction afin d'internationaliser les messages d'erreur.

Le code 000000 (service 00, domaine 00, item 00) est un code de test. Il ne faut pas l'utiliser dans le code Vitam ni le supprimer.

1.2.9.1.4 Ajout d'élement dans les énums

Au fur et à mesure des développements, chaque développeur va être amené à ajouter une erreur. Il n'aura principalement qu'à ajouter une ligne dans VitamCode. Cependant, le triplet service, domain, item est unique.

Pour garantir cette unicité, un test unitaire se charge de vérifier les trois énums : CodeTest.

Dans un premier temps sont validés les codes (2 caractères alphanumériques en majuscule) pour chaque énum. Ensuite est vérifié l'unicité des codes pour chacune.

Ces tests n'ont pas à être modifiés ! S'ils ne passent plus après l'ajout d'une entrée, c'est que celle ci est incorrecte, le test ne le sera jamais. Dans les logs de CodeTest vous trouverez la raison de l'erreur (code dupliqué et avec lequel ou erreur dans le code).

1.2.9.2 Utilisation

Afin de récupérer un VitamCode, il suffit de passer par l'énum :

```
VitamCode vitamCode = VitamCode.TEST;
```

Il est également possible de le récupérer directement via son code à l'aide du helper VitamCodeHelper :

```
VitamCode vitamCode = VitamCodeHelper.getFrom("012AE5");
```

A partir des getter de l'énum VitamCode, il est possible de récupérer les différentes informations :

```
VitamCode vitamCode = VitamCode.TEST;
ServiceName service = vitamCode.getService();
DomainName domain = vitamCode.getDomain();
String item = vitamCode.getItem();
Status status = vitamCode.getStatus();
String message = vitamCode.getMessage();
```

Concernant le message, il est possible de lui mettre des paramètres (String.format()). Ainsi, via le helper, il est possible de récupérer le message avec les paramètres insérés :

```
VitamCode vitamCode = VitamCode.TEST;
String message = VitamCodeHelper.getParametrizedMessage(vitamCode, "monParametre",
↪"monAutreParametre");
```

Il est possible de récupérer un "log" formaté et paramétré telque "[codeVitam] message paramétré" :

```
String log = VitamCodeHelper.getLogMessage(VitamCode.TEST, param1, param2);
```

1.2.10 Common format identification

1.2.10.1 But de cette documentation

Cette documentation indique comment utiliser le code commun du format identifier pour éventuellement ajouter un client pour un nouvel outil.

1.2.10.2 Outil Format Identifier

L'interface du format identifier est *fr.gouv.vitam.common.format.identification.FormatIdentifier*. Elle met à disposition 2 méthodes :

- status() qui renvoie le statut du format identifer
- analysePath(Path) qui renvoie une liste de formats potentiellement identifiés par l'outil.

Une implémentation Mock est présente : *fr.gouv.vitam.common.format.identification.FormatIdentifierMock*
Chaque nouvel outil doit implémenter l'interface :

```
public class FormatIdentifierSiegfried implements FormatIdentifier {
  @Override
  public FormatIdentifierInfo status() { //CALL THE TOOL AND GET THE STATUS }

  @Override
  public List<FormatIdentifierResponse> analysePath(Path path) { //CALL THE TOOL AND␣
↪ANALYSE}
}
```

De plus, pour pouvoir être utilisé, l'outil doit être ajouté dans l'enum FormatIdentifierType :

```
public enum FormatIdentifierType {
   MOCK,
   SIEGFRIED
}
```

Une factory a été mise en place pour récupérer l'instance du client adaptée. En cas de nouvel outil, il faut la mettre à jour :

```
public class FormatIdentifierFactory {
  ......
  private FormatIdentifier instanciate(String formatIdentifierId){
    ...
    switch (infos.getType()) {
      case MOCK:
        return new FormatIdentifierMock();
      case SIEGFRIED:
        return new FormatIdentifierSiegfried(infos.getConfigurationProperties());
      .....
    }
  }
}
```

1.2.10.3 Configuration

Dans **/vitam/conf** du serveur applicatif où sont déployés les services d'identification de formats, il faut un fichier **format-identifiers.conf**. C'est un fichier YAML de configuration des services d'identification de format. Il possède les configurations des services que l'on souhaite déployer sur le serveur.

Le code suivant contient un exemple de toutes les configurations possibles :

```
siegfried-local:
  type: SIEGFRIED
  client: http
  host: localhost
  port: 55800
  rootPath: /root/path
  versionPath: /root/path/version/folder
  createVersionPath: false
mock:
  type: MOCK
```

Pour plus d'informations sur le sujet, voir la documentation sur l'exploitation.

1.2.11 Common-storage

Le common storage est un module commun pour plusieurs modules qui consiste à gérer des objets stockés dans un container et/ou dans dans un repertoire, ce module propose plusieurs offres de stockage (Jclouds), par exemple file Systeme et swift (open stack et ceph) configurabe par code (java) ou par fichier de configuration. Dans les chapitres suivants, on présentra les 2 modes du configuration

1.2.11.1 1- Présentation des APIs Java :

1.1 - Introduction :

Le Module common storage expose un ensemble des methodes qui gèrent la creation, la mise à jour , la supprission des contenaire, des repertoires et des objets, Vous trouverez ci-dessous la liste des methodes avec leur fonctions attendus.

L'API principale est l'interface ContentAddressableStorage. Celle-ci a la hiérarchie de classe suivante :

- ContentAddressableStorageAbstract : classe abstraite implémentant quelques méthodes communes
  - HashFileSystem : implémentation d'un CAS sur FileSystem (via java.nio.*) avec une répertoire par sous-répertoire permettant un stockage d'un grand nombre d'objets (jusqu'à 500e6 objets)
  - ContentAddressableStorageJcloudsAbstract : classe abstraite implémentant la plupart des méthodes pour une implémentation jclouds sous-jacente
    - FileSystem : implémentation d'un CAS sur FileSystem (via jclouds) avec une répertoire à plat sous les container
    - OpenstackSwift : classe d'implémentation permettant le stockage sur Swift (via jclouds)

1.2 - Liste des méthodes :

- getContainerInformation : consulter les information d'un contenaire (pour la version 0.14.0-SNAPSHOT)
  - Paramètres :
  - containerName : :String
  - Retourner : (pour la version 0.14.0-SNAPSHOT) l'espace utilisés et l'espace disponible par région
- CreateContainer : creer un contenaire
  - Paramètres :
  - containerName : :String
  - Retourner :
- getUriListDigitalObjectFromFolder :
  - Paramètres :
    - containerName : :String (le nom de contenaire à consulter)
    - folderName : :String (le nom de repertoire à consulter pour lister les URIs des objets)
  - Retourner :
    - List<URI> : La liste des URIs des objets dans le repertoire cité ci-dessus.
- getObjectMetadatas : lire et récupérer les métadonnées d'un objet (le fichier ou le répertoire)
  - Paramètres :
  - containerName : :String (le nom de contenaire dans lequel qu'on stock l'object)
  - objectId : :String (Id de l'object. S'il est null, c'est-à-dire, il est un repertoire)
  - Retourner :
    - MetadatasObject : La classe qui contient les informations de metadata

- objectName : l'ID du fichier
- type : le type (dossier comme Units, Binary, ObjectGroup, Reports, ...)
- digest : l'empreinte
- fileOwner : propriétaire
- fileSize : taille du fichier
- lastAccessDate : date de dernier accès
- lastModifiedDate : date de modification des données

Dans le cas echéant la method retourne une immuatable empty list.

- uncompressObject : cette méthode extracte des fichiers compressés toute en indiquant le type de l'archive, pour cette version (v0.14.0) supporte 4 types : zip, tar, tar.gz et tar.bz2.

  -Paramètres :

  - containerName : :String : c'est le nom de container dans lequel on stocke les objets
  - folderName : :String : c'est le repertoire racine .
  - archiveType : : String : c'est le nom ou le type de l'archive (exemple : application/zip , application/x-tar)
  - compressedInputStream : :InputStream c'est le stream des objets compressés

- retourner :

Dans le cas echéant (uncompress KO) la methode génère une exception avec un message internal server.

- getObjectInformation :cette méthode retourne des informations utils sur un objet
  - Paramètres :
  - containerName : :String (le nom de contenaire à consulter)
  - objectName : :String
  - Retourner :
  - JsonNode

  La méthode retourne un Json contenant des informations sur un objet présent dans un contenaire prédéfinit (et des exceptions en cas d'erreur : objet non existant, erreur server).

1.2.11.2 2 - Configuration

La première chose que nous devons faire est d'ajouter la dépendance maven dans le pom.xml du projet. Après il faut configurer le contexte de stockage souhaités (filesystem/swift ceph/ swift openStack), (on traitera cette problématique au chapitre 2.1 et 2.2)

```
<dependency>
     <groupId>fr.gouv.vitam</groupId>
     <artifactId>common-storage<artifactId>
     <version>x.x.x</version>
</dependency>
```

La configuration de l'offre de stockage est basé sur plusieurs paramètres :

- provider : : String : le type de l'offre de stockage (valeur par defaut : filesystem, valeur possibles : openstack-swift , filesystem ou chaîne vide)
- keystoneEndPoint* : : String : URL d'authentification keystone
- swiftSubUser* : : String : le nom de l'utilisateur (sur rados, il prend la forme <tenant>$<user>)
- cephMode* : : boolean : l'implementation swift (true pour ceph, false pour openstack)

- storagePath : : String : path de stockage pour l'offre FileSystem

2.1 - Configuration par code :

2.1.a Exemple file systeme :

```
StorageConfiguration storeConfiguration = new StorageConfiguration().
↪setProvider(StorageProvider.FILESYSTEM.getValue())
  .setStoragePath("/");
```

2.1.b Exemple SWIFT CEPH

```
StorageConfiguration storeConfiguration = new StorageConfiguration().
↪setProvider(StorageProvider.SWIFT.getValue())
     .setKeystoneEndPoint("http://10.10.10.10:5000/auth/v1.0)
     .setSwiftUid(swiftUID)
     .setSwiftSubUser(user)
     .setCredential(passwd)
     .setCephMode(true);
```

2.1.c Exemple SWIFT OpenStack

```
StorageConfiguration storeConfiguration = new StorageConfiguration().
↪setProvider(StorageProvider.SWIFT.getValue())
     .setKeystoneEndPoint("http://10.10.10.10:5000/auth/v1.0)
     .setSwiftUid(swift)
     .setSwiftSubUser(user)
     .setCredential(passwd)
     .setCephMode(false);
```

2.2 - Configuration par fichier

Exemple d'un fichier de configuration :

```
provider: openstack-swift
keystoneEndPoint : http://10.10.10.10:5000/auth/v1.0
swiftUid : vitam
swiftSubUser : swift
credential : password
cephMode : true
```

Dans ce cas, on peut utiliser un Builder qui permet de fournir le context associé au provider.

```
ContentAddressableStorage storage=StoreContextBuilder.
↪newStoreContext(configuration)
```

1.2.11.3 3- Présentation des méthodes dans SWIFT & FileSystem :

3.1 - Introduction :

Il y a deux classes qui héritent les APIs. l'une utilise SWIFT et l'autre utilise FileSystem.

3.2 - Liste des méthodes :

3.2.1 getObjectInformation :

- SWIFT : Obtenir l'objet par les APIs swift

```
        result.setFileOwner("Vitam_" + containerName.split("_")[0]);
result.setType(containerName.split("_")[1]);
result.setLastAccessDate(null);
if (objectId != null) {
    SwiftObject swiftobject = getSwiftAPi()
        .getObjectApi(swiftApi.getConfiguredRegions().iterator().next(),␣
↪containerName).get(objectId);

    result.setObjectName(objectId);
    result.setDigest(computeObjectDigest(containerName, objectId, VitamConfiguration.
↪getDefaultDigestType()));
    result.setFileSize(swiftobject.getPayload().getContentMetadata().
↪getContentLength());
    result.setLastModifiedDate(swiftobject.getLastModified().toString());
} else {
    Container container = getContainerApi().get(containerName);
    result.setObjectName(containerName);
    result.setDigest(null);
    result.setFileSize(container.getBytesUsed());
    result.setLastModifiedDate(null);
}
```

- FileSystem : Obtenir le fichier de jclouds par le nom du conteneur et le nom du dossier

```
        File file = getFileFromJClouds(containerName, objectId);
BasicFileAttributes basicAttribs = getFileAttributes(file);
long size = Files.size(Paths.get(file.getPath()));
if (null != file) {
    if (objectId != null) {
        result.setObjectName(objectId);
        result.setDigest(computeObjectDigest(containerName, objectId,␣
↪VitamConfiguration.getDefaultDigestType()));
        result.setFileSize(size);
    } else {
        result.setObjectName(containerName);
        result.setDigest(null);
        result.setFileSize(getFolderUsedSize(file));
    }
    result.setType(containerName.split("_")[1]);
    result.setFileOwner("Vitam_" + containerName.split("_")[0]);
    result.setLastAccessDate(basicAttribs.lastAccessTime().toString());
    result.setLastModifiedDate(basicAttribs.lastModifiedTime().toString());
}
```

1.2.11.4 4- Détail de l'implémentation HashFileSystem

Logique d'implémentation

- /<storage-path> : défini par configuration
  - /container-name : sur les offres de stockage, cela est construit dans le CAS Manager par concaténation du type d'objet et du tenant . Cette configuration n'est pas la configuration cible (notamment par rapport à l'offre froide)
    - /0/a/b/c/<fichier> : avec 0abc les 4 premiers hexdigits du SHA-256 du nom du fichier stocké

1.2.12 Métriques dans VITAM

1.2.12.1 Fonctionnement

Les métriques dans VITAM sont stockées dans le package :

fr.gouv.common.metrics

Les registres de métriques et les reporters de métriques sont tous les deux contenus dans une classe *VitamMetrics*. Cette classe doit être instanciée avec un *VitamMetricsType* qui peut être **JERSEY**, **JVM** ou **BUSINESS**. Le type définira les métriques enregistrées dans le registre interne de la classe.

La classe **AbstractVitamApplication** contient une *Map* statique de *VitamMetrics* qui est initialisée à chaque configuration d'une application VITAM. Cette *Map* contient obligatoirement un *VitamMetrics* de type BUSINESS et peut accessoirement contenir les *VitamMetrics* de types JVM et JERSEY. La fonction en question est :

```
protected static final void clearAndconfigureMetrics()
```

Cette fonction vide et recharge les métriques à chaque création d'une application VITAM. Les reporters de métriques quant à eux sont démarrés lors d'un appel à la fonction :

```
public final void run() throws VitamApplicationServerException
```

qui elle même appelle la fonction :

```
public final void startMetrics()
```

de la classe *AbstractVitamApplication*.

Note : Les *VitamMetrics* de type JERSEY ou JVM n'ont pas à être modifiés pendant l'execution d'une application VITAM.

1.2.12.2 Configuration

Les métriques sont configurées dans le fichier `/vitam/conf/<service_id>/vitam.metrics.conf`. Ce fichier contient la documentation nécessaire pour configurer correctement les métriques.

1.2.12.3 Métriques métier

Les métriques métiers permettent aux développeurs d'enregistrer des métriques n'importe où dans le code, pour par exemple suivre une variable ou bien chronométrer une fonction. Pour cela il suffit d'appeler la fonction statique *getBusinessMetricRegistry* dans la classe *AbstractVitamApplication*, puis d'enregistrer une métrique.

```
AbstractVitamApplication.getBusinessMetricsRegistry().register("Running workflows",
    new Gauge<Long>() {
        @Override
        public Long getValue() {
            return runningWorkflows.get();
        }
    });
```

Avertissement : Avec la fonction *register*, si une métrique avec un nom identique est déjà enregistrée, alors l'ancienne métrique sera ecrasée par la nouvelle avec un avertissement dans les logs. En revanche, avec les fonctions de création de métrique comme *timer*, *meter*..., une exception sera soulevée.

1.2.12.4 Reporters

2 reporters sont disponibles, un reporter Logback (toutes les métriques sont dumpées dans Logback) ou bien un reporter ElasticSearch (toutes les métriques sont dumpées dans la base ElasticSearch Log). Le reporter est configurable avec un interval de temps entre chaque reporting.

Avertissement : Les index ElasticSearch ne sont pas configurables pour les métriques. Ils se nomment respectivement : * `metrics-vitam-jersey-YYYY.MM.dd` pour les métriques JERSEY * `metrics-vitam-jvm-YYYY.MM.dd` pour les métriques JVM * `metrics-vitam-business-YYYY.MM.dd` pour les métriques métier

1.2.12.5 Legacy

Pour celui ou celle qui souhaiterais continuer le développement du système de métriques au sein de VITAM, voici quelques points qui peuvent être intéressants à développer :

- Pour un reporter ElasticSearch, vérifier l'état de la connexion à chaque reporting et augmenter progressivement le temps de reporting si la base de données n'est pas accessible.
- Permettre le chargement de reporters de manière générique, en se passant du *switch* dans *VitamMetrics* et abstraire tout ce qui concerne le reporting.

1.2.13 Common-private

1.2.13.1 Génération de certificats et de keystore

1.2.13.1.1 Présentation

Nous avons besoins de certificats & keystore pour la procédure d'authentification client-serveur. Ce document présente comment nous les créeons

1. Pour rappel, nous avons besoins de différents keystore :

- keystore.jks : contient le certificat de la clé privé du serveur
- truststore.jks : contient la chaîne des CAs qui génère ce certificat de clients & serveurs
- granted_certs.jks : list de certificats du client qui sont autorisés à faire des requêtes vers le serveur
- le client qui doit présenter sa clé privée & le certificat,lors d'une requête d'authentification.

2.Création des certificats Comme il n'y a pas de PKI, nous utilisons le xca pour générer des certificats et pour les tests. Nous créons l'ensemble des certificats suivants en utilisant le xca.

- VitamRootCA : certificat auto-signé, modèle de certificat : CA, X509v3 Basic Constraints Extensions : Autorité de Certification
- VitamIntermediateCA : certificat signé par VitamRootCA, modèle de certificat : CA, X509v3 Basic Constraints Extensions : Autorité de Certification

- IngestExtServer : certificat signé par VitamIntermediateCA , modèle de certificat : https_server, X509v3 Basic Constraints Extensions : Entité Finale
- client : certificat signé par VitamIntermediateCA , modèle de certificat : https_client, X509v3 Basic Constraints Extensions : Entité Finale
- client_expired : certificat signé par VitamIntermediateCA , modèle de certificat : https_client, X509v3 Basic Constraints Extensions : Entité Finale
- client_notgranted : certificat signé par VitamIntermediateCA , modèle de certificat : https_client, X509v3 Basic Constraints Extensions : Entité Finale

Une fois qu'on a créé ces certificats, nous exportons ces certificats soit en format crt, pem ou p12 pour des utilisations différentes

3. Création des keystores vides Nous utilisons le keytool pour créer les keystores

keytool -genkey -alias mydomain -keystore keystore.jks keytool -delete -alias mydomain -keystore keystore.jks

keytool -genkey -alias mydomain -keystore truststore.jks keytool -delete -alias mydomain -keystore truststore.jks

keytool -genkey -alias mydomain -keystore granted_certs.jks keytool -delete -alias mydomain -keystore granted_certs.jks

4. Import des certificats

- **truststore.jks** [importer VitamIntermediateCA.crt, VitamRootCA.crt] keytool -import -trustcacerts -alias VitamRootCA -file VitamRootCA.crt -keystore truststore.jks keytool -import -trustcacerts -alias VitamIntermediateCA -file VitamIntermediateCA.crt -keystore truststore.jks
- **keystore.jks** importer la clé privée et le certificat du serveur keytool -v -importkeystore -srckeystore IngestExtServer.p12 -srcstoretype PKCS12 -destkeystore keystore.jks -deststoretype JKS keytool -import -trustcacerts -alias IngestExtServer -file IngestExtServer.crt -keystore truststore.jks
- **granted_certs.jks** importer des certificats client.crt et client_expired.crt

5. Utilisation des certificats client. exporter en format p12 ou pem selon des buts d'utilisations.

1.2.13.2 esapi utilisation

```
<?xml version="1.0" encoding="UTF-8"?>
<policy>
      <settings>
              <mode>redirect</mode>
              <error-handling>
                      <default-redirect-page>/security/error.jsp</default-redirect-
↪page>
                      <block-status>403</block-status>
              </error-handling>
      </settings>
      <outbound-rules>
              <add-header name="FOO" value="BAR" path="/.*">
                      <path-exception type="regex">/marketing/.*</path-exception>
              </add-header>
      </outbound-rules>
</policy>
```

1.2.13.3 Format Identifiers

1.2.13.3.1 But de cette documentation

Cette documentation indique comment utiliser les services d'identification de format et comment créer sa propre implémentation.

1.2.13.3.2 Format Identifier

L'interface commune du service d'identification des formats est : *fr.gouv.vitam.common.format.identification.FormatIdentifier*.

Elle mets à disposition les méthodes suivantes :

- la récupération du status du logiciel auquel le service se connecte
- l'identification du format d'un fichier par le logiciel auquel le service se connecte

Les implémentations de l'interface sont :

- pour l'implémentation Mock : *fr.gouv.vitam.common.format.identification.FormatIdentifierMock*
- pour l'implémentation du logiciel Siegfried : *fr.gouv.vitam.common.format.identification.FormatIdentifierSiegfried*

Il sera possible d'en implémenter d'autres.

1.2.13.3.2.1 Implémentation Mock

Implémentation simple renvoyant des réponses statiques.

1.2.13.3.2.2 Implémentation Siegried

Implémentation basique utilisant un client HTTP.

1.2.13.3.3 Format Identifier Factory

Afin de récupérer l'implémentation configurée une factory a été mise en place.

1.2.13.3.3.1 Configuration

Cette factory charge un fichier de configuration "format-identifiers.conf". Ce fichier contient les configurations des services d'identificaton de format identifiées par un id :

```
siegfried-local:
        type: SIEGFRIED
        client: http
        host: localhost
        port: 55800
        rootPath: /root/path
        versionPath: /root/path/version/folder
        createVersionPath: false
mock:
        type: MOCK
```

Le type est obligatoire et doit correspondre à l'enum *fr.gouv.vitam.common.format.identification.model.FormatIdentifierType*.

Les autres données sont spécifiques à chaque implémentation du service d'identification de format.

Si le fichier n'est pas présent au démarrage du serveur, aucune configuration n'est chargée par la factory.

1.2.13.3.3.2 Méthodes

Pour récupérer un service d'identification de formats :

```
FormatIdentifier siegfried = FormatIdentifierFactory.getInstance().
↪getFormatIdentifierFor("siegfried-local");
```

Pour ajouter une configuration mock :

```
FormatIdentifierConfiguration mock = new FormatIdentifierConfiguration();
siegfried.setType(FormatIdentifierType.MOCK);
FormatIdentifierFactory.getInstance().addFormatIdentifier("mock", mock);
```

Pour ajouter une configuration siegfried :

```
siegfried-local:
        type: SIEGFRIED
        client: http
        host: localhost
        port: 55800
        rootPath: /root/path
        versionPath: /root/path/version/folder
        createVersionPath: false
```

client : *http* correspond au client HTTP à lancer (ce dernier effectue des requêtes HTTP pour analyser les fichiers) *host/port* correspond au le serveur sur lequel Siegfried est installé. *rootPath* correspond au chemin vers les fichiers analysables par Siegfried. *versionPath* correspond au chemin vers un dossier vide utilisé pour requêter la version de Siegfried. *createVersionPath* : Si *false* le dossier doit pré-existant sur le server sur lequel tourne Siegfried. Sinon, le client siegfried tente de créer automatiquement le dossier en local.

```
FormatIdentifierConfiguration siegfried = new FormatIdentifierConfiguration();
siegfried.setType(FormatIdentifierType.SIEGFRIED);
FormatIdentifierFactory.getInstance().addFormatIdentifier("siegfried-local",␣
↪siegfried);
```

Pour supprimer une configuration :

```
FormatIdentifierFactory.getInstance().removeFormatIdentifier("siegfried-local");
```

1.2.13.4 Introduction

1.2.13.4.1 But de cette documentation

L'ensemble de ces documents est le manuel de développement du module Graph, qui représente le métier fonctionnel de US story #510 de projet VITAM, dont le but et de définir un niveau d'indexation de chaque Unit aprés avoir créer un arbre à partir de fichier SEDA.

Le manuel se compose de : - DAT présente l'architecture technique du module au niveau des packages, classes.

1.2.13.5 DAT : module Graph

Ce document présente l'ensemble de manuel développement concernant l'algorithme de graph qui représente le story #510, qui contient :

1.2.13.5.1 modules & packages

1. Modules et packages

Au présent : nous proposons le schéma ci-dessous représentant le module principal et ses sous modules.

graph

|—DirectedCycle : Directed cycle detection : un graphe orienté donné a un cycle dirigé ? Si oui, trouver un tel cycle. DirectedCycle.java résout ce problème en utilisant la recherche en profondeur d'abord.

Depth-first orders : fait de recherche en profondeur d'abord sur chaque sommet exactement une fois. Trois ordres de sommet sont d'un intérêt dans des applications typiques :

Preorder : Mettre le sommet(vertex) sur une file d'attente avant les appels récursifs. Postorder : Mettre le sommet(vertex) sur une file d'attente après les appels récursifs. Reverse postorder : Mettre le sommet(vertex) sur une pile après les appels récursifs.

Le Depth-first search est l'algorithme de recherche des composantes fortement connexes. L'algorithme consiste à démarrer d'un sommet et à avancer dans le graphe en ne repassant pas deux fois par le même sommet. Lorsque l'on est bloqué, on ''revient sur ses pas" jusqu'à pouvoir repartir vers un sommet non visité. Cette opération de ''retour sur ses pas" est très élégamment prise en charge par l'écriture d'une procédure récursive.

Après la parse de Unit recursive et la creation d'arbre orienté.Le choix de la racine de départ de l'arbre orienté se fait en faisant le test récursive si l'élément ne possède pas un up alors c'est un racine .

|—DirectedGraph : Un graphe orienté (ou digraphe) est un ensemble de sommets et une collection de bords orientés qui relie chacun une paire ordonnée de sommets.

Un bord dirigé pointe du premier sommet de la paire et les points au deuxième sommet de la paire.

|—Graph Un graphe est composé d'un ensemble de sommets et un ensemble d'arêtes . Chaque arête représente une liaison entre deux sommets.

Deux sommets sont voisins s'ils sont reliés par un bord , et le degré d'un sommet est le nombre de ses voisins. Graph data type. Graph-processing algorithms généralement d'abord construit une représentation interne d'un graphe en ajoutant des arêtes (edges), puis le traiter par itération sur les sommets et sur les sommets adjacents à un sommet donné.

L'algorithme de chemin le plus long est utilisé pour trouver la longueur maximale d'un graph donné. La longueur maximale peut être mesuré par le nombre maximal d'arêtes ou de la somme des poids dans un graph pondéré.

L'algorithme de chemin le plus long permet de définir dans notre cas le niveau d'indexation de chaque Unit .

***L'algorithme de parcours en profondeur (ou DFS, pour Depth First Search) est un algorithme de parcours d'arbre, et plus généralement de parcours de graphe, qui se décrit naturellement de manière récursive. Son application la plus simple consiste à déterminer s'il existe un chemin d'un sommet à un autre.

1.2.13.6 Paramètres

1.2.13.6.1 Présentation

Dans tout le projet Vitam sont utilisés différents paramètres transmis aux différentes classes ou au différentes méthodes. Afin de ne pas bloquer toute évolution, il est recommandé d'utiliser une classe de paramètres (afin d'éviter de modifier le nombre de paramètres en signature de méthodes) ou d'utiliser une Map.

1.2.13.6.2 Principe

L'idée ici est de mettre en place une mécanique de paramètres commune à tous les modules Vitam. Pour se faire, une interface VitamParameter a été créée. Afin de créer une nouvelle classe de paramètre, il faut alors implémenter cette interface qui retourne une Map de paramètre et un Set de noms de paramètre obligatoires. Cette interface est générique et prend comme typage une énum qui dispose du nom des paramètres.

Une classe utilitaire, ParameterHelper a été mise en place afin de vérifier les champs obligatoires. Elle s'appuie sur les deux méthodes définies dans l'interface VitamParameter.

1.2.13.6.3 Mise en place

1.2.13.6.3.1 Nom des paramètres

Nous souhaitons mettre en place une classe de paramètre pour le module storage, StorageParameter. Il faut dans un premier temps une énum disposant des noms de paramètre.

```
public enum StorageParameterName {
    /**
    * Nom du premier paramètre
    **/
    field1,
    /**
    * Nom du deuxième paramètre
    **/
    field2,
    /**
    * Nom du troisième paramètre
    **/
    field3;
}
```

1.2.13.6.3.2 Interface

Ensuite, une interface va définir les différentes methodes nécéssaires à la classe de paramètre ("définition du contrat") tout en héritant de l'interface VitamParameter (afin que la classe implémentant cette nouvelle interface implémente les deux méthodes getMapParameters et getMandatoriesParameters.

```
/**
* Exemple d'interface de paramètres
**/
public interface StorageParameters extends VitamParameter<StorageParameterName> {
    /**
     * Put parameterValue on mapParameters with parameterName key <br />
     * <br />
     * If parameterKey already exists, then override it (no check)
     *
     * @param parameterName the key of the parameter to put on the parameter map
     * @param parameterValue the value to put on the parameter map
     * @return actual instance of WorkerParameter (fluent like)
     * @throws IllegalArgumentException if the parameterName is null or if␣
↪parameterValue is null or empty
     **/
    StorageParameters putParameterValue(StorageParameterName parameterName, String␣
↪parameterValue);
```

```
    /**
     * Get the parameter according to the parameterName
     *
     * @param parameterName the wanted parameter
     * @return the value or null if not found
     * @throws IllegalArgumentException throws if parameterName is null
     **/
    String getParameterValue(StorageParameterName parameterName);
    /**
     * Set from map using String as Key
     *
     * @param map the map parameters to set
     * @return the current instance of WorkerParameters
     * @throws IllegalArgumentException if parameter key is unknown or if the map is␣
↪null
     **/
    StorageParameters setMap(Map<String, String> map);
    /**
    * Get the field1 value
    *
    * @return the field1's value
    **/
    String getStorageParameterField1();
}
```

1.2.13.6.3.3 Possibilité d'avoir une classe abstraite

Le but est d'implémenter cette interface. Cependant, il est possible de vouloir plusieurs classes de paramètres en fonction des besoins. Il est alors possible de mettre en place une classe abstraite qui implémente les méthodes communes aux différentes classes de paramètre (par exemple les getters / setters).

```
abstract class AbstractStorageParameters implements StorageParameters {
    @JsonIgnore
    private final Map<StorageParameterName, String> mapParameters = new TreeMap<>();
    @JsonIgnore
    private Set<StorageParameterName> mandatoryParameters;
    AbstractStorageParameters(final Set<StorageParameterName> mandatory) {
        mandatoryParameters = mandatory;
    }
    @JsonCreator
    protected AbstractStorageParameters(Map<String, String> map) {
        mandatoryParameters = StorageParametersFactory.getDefaultMandatory();
        setMap(map);
    }
    @JsonIgnore
    @Override
    public Set<StorageParameterName> getMandatoriesParameters() {
        return Collections.unmodifiableSet(new HashSet<>(mandatoryParameters));
    }
    @JsonIgnore
    @Override
    public Map<StorageParameterName, String> getMapParameters() {
        return Collections.unmodifiableMap(new HashMap<>(mapParameters));
    }
    @JsonIgnore
    @Override
```

```
    public WorkerParameters putParameterValue(StorageParameterName parameterName,␣
↪String parameterValue) {
        ParameterHelper.checkNullOrEmptyParameter(parameterName, parameterValue,␣
↪getMandatoriesParameters());
        mapParameters.put(parameterName, parameterValue);
        return this;
    }
    @JsonIgnore
    @Override
    public String getParameterValue(StorageParameterName parameterName) {
        ParametersChecker.checkParameter(String.format(ERROR_MESSAGE, "parameterName
↪"), parameterName);
        return mapParameters.get(parameterName);
    }
    @JsonIgnore
    @Override
    public StorageParameters setMap(Map<String, String> map) {
        ParametersChecker.checkParameter(String.format(ERROR_MESSAGE, "map"), map);
        for (String key : map.keySet()) {
            mapParameters.put(WorkerParameterName.valueOf(key), map.get(key));
        }
        return this;
    }
    @JsonIgnore
    @Override
    public String getField1() {
        return mapParameters.get(StorageParameterName.field1);
    }
}
```

1.2.13.6.3.4 Possibilité d'avoir une factory

On voit dans le code d'exemple l'utilisation d'une factory qui permet d'obetnir la bonne implémentation de la classe de paramètres. En effet, au travers de la factory il est facilement possible de mettre en place les champs requis en fonction des besoins. Par exemple, certains paramètres peuvent être obligatoire pour toutes les implémentations alors que certains sont en plus requis pour certaines implémentations. Voir ici s'il n'est pas possible de faire une factory commune.

```
public class WorkerParametersFactory {
    private static final Set<StorageParameterName> genericMandatories = new HashSet<>
↪();
    static {
        genericMandatories.add(StorageParameterName.field1);
        genericMandatories.add(StorageParameterName.field2);
    }
    private StorageParametersFactory() {
        // do nothing
    }
    // Méthodes de la factory
    // ...
}
```

1.2.13.6.3.5 Code exemple

Ensuite, là où les paramètres sont nécéssaires, il suffit d'utiliser l'interface afin d'être le plus générique possible.

```
public void methode(StorageParameters parameters) {
    // Check des paramètres
    ParameterHelper.checkNullOrEmptyParameters(parameters);
    // Récupération des paramètres
    String value = parameters.getField1();
    String value 2 = parameters.get(StorageParameterName.field2);
    // etc...
}
// Exemple d'ajout de champs requis
public void methode2() {
    Set<StorageParameterName> mandatoryToAdd = new Set<>();
    mandatoryToAdd.put(StorageParameterName.field3);
    // Initialisation des paramètres
    StorageParameters parameters = StorageParameterFactory.
↪newStorageParameters(mandatoryToAdd);
    // etc..
}
```

1.2.13.6.4 Exemple d'utilisation dans le code Vitam

Il est possible de retrouver l'utilisation des paramètres génériques Vitam dans les modules suivants :

- Processing
- Logbook

1.2.13.7 Uniform Resource Identifier (URI) (vitam)

UriUtils Utilisé pour retirer le dossier racine du chemin d'un URI

Dans le cadre de vitam Dossier racine : sip Dossier des objets numériques : content

sip/content

1.2.13.7.1 fonctions

UriUtils.splitUri(String uriString)

1.2.13.8 Configuration de apache shiro

TODO : présentation de apache shiro, configuration, ...

1.2.13.9 Présentation authentification via certificats

Afin de pouvoir authentifier des clients via des certificats valides il suffit de bien configurer shiro. Pour ce faire vitam utilise le fichier shiro.ini qui a la forme suivante.

```
[main]
x509 = fr.gouv.vitam.common.auth.web.filter.X509AuthenticationFilter
x509.useHeader = false
x509credentialsMatcher = fr.gouv.vitam.common.auth.core.authc.
 ↪X509CredentialsSha256Matcher
x509Realm = fr.gouv.vitam.common.auth.core.realm.X509KeystoreFileRealm
x509Realm.grantedKeyStoreName = path/granted_certs.jks
x509Realm.grantedKeyStorePassphrase = password
x509Realm.trustedKeyStoreName = path/truststore.jks
x509Realm.trustedKeyStorePassphrase = password
x509Realm.credentialsMatcher = $x509credentialsMatcher
securityManager.realm = $x509Realm
securityManager.subjectDAO.sessionStorageEvaluator.sessionStorageEnabled = false
[urls]
/ingest-ext/v1/**= x509
```

1.2.13.10 Décryptage de shiro.ini

[main] Contient les déclaration de filters et classes comme par exemple X509AuthenticationFilter, X509CredentialsSha256Matcher, X509KeystoreFileReal, ... La clé (x509, x509Realm) sont custom et on peut donner le nom qu'on veut, par contre securityManager est un mot clé shiro. La ligne securityManager.realm = $x509Realm passe à shiro le Realm qu'on veut utiliser, ceci dit, les clé custom peut être passé à shiro de la même façon.

[urls] Pour une url donnée on dit quel filter utiliser, exemple : /ingest-ext/v1/**= x509 signifie que l'on veut utiliser le filter x509 pour toutes les urls de type /ingest-ext/v1/**

1.2.13.11 Utilisation des certificats

Vitam a une implémentation de filter pour utiliser des certificats x509 afin d'authentifier des clients.

X509AuthenticationFilter (filter par defaut)

- Activation du filter dans le fichier shiro.ini : x509 = fr.gouv.vitam.common.auth.web.filter.X509AuthenticationFilter
- Ce filter récupère les certificats fournis dans la requête :

```
X509Certificate[] clientCertChain = (X509Certificate[]) request.
 ↪getAttribute("javax.servlet.request.X509Certificate");
```

- Si des certificats sont trouvé alors un token est crée qui sera passé à la méthode qui s'occupe d'authentifier un client.

```
   new X509AuthenticationToken(clientCertChain, getHost(request));

- X509AuthenticationFilter peut aussi authentifier via un certificat passé dans le␣
 ↪header. La variable "useHeader" est égale à false par défaut. Donc cette option est␣
 ↪désactivé par défaut. Si useHeader= true (qu'on peut spécifier dans shiro.ini: x509.
 ↪useHeader = false dans l'exemple ci-dessus) et qu'aucun cetificat n'est fourni dans␣
 ↪l'attribute de la requête javax.servlet.request.X509Certificate alors il bascule␣
 ↪vers une authentification via le header. Le nom du header est X-SSL-CLIENT-CERT, et␣
 ↪il doit avoir comme valeur un certificat valide au format pem. Le certificat pem␣
 ↪est ensuite converti vers un X509Certificate qui sera utilisé pour créer le token d
 ↪'authentification. Ci-dessous une snipet de code qui permets de récupérer la valeur␣
 ↪du certificat depuis le header et le convertir au bon format.
- Attention, jetty n'accepte pas les retour à la ligne dans le header, d'ou la␣
 ↪nécessité d'encoder le pem en base 64. X509AuthenticationFilter s'occupe de␣
 ↪déterminer si le certificat passé dans le header est encodé ou non en base 64 et␣
 ↪fera en sorte d'accepter même les certificats non encodés.
```

```
   final HttpServletRequest httpRequest = (HttpServletRequest) request;
   String pem = httpRequest.getHeader(X_SSL_CLIENT_CERT);
   byte[] pemByte = null;
   if (null != pem) {
       try {
           try {
               pemByte = Base64.getDecoder().decode(pem);
           } catch (IllegalArgumentException ex) {
               // the pem is not base64 encoded
               pemByte = pem.getBytes();
           }
           final InputStream pemStream = new ByteArrayInputStream(pemByte);
           final CertificateFactory cf = CertificateFactory.getInstance("X.509");
           final X509Certificate cert = (X509Certificate) cf.
↪generateCertificate(pemStream);
           clientCertChain = new X509Certificate[] {cert};
       } catch (Exception ce) {
           throw new ShiroException(ce);
       }
   }


- Il faut noter que l'authentification via un certificat passé dans le header n'est␣
↪pas sécurisée (moins sécurisée que la solution via l'attribute de la requête). En␣
↪effet, il peut y avoir une injection lors de l'acheminement de la requête depuis un␣
↪client vers un serveur jetty. Nous recommendons donc l'utilisation de certificats␣
↪dans l'attribute de la requête.
```

1.2.13.12 Présentation

Nous proposons le filtre de sécurité qui permet de contrôler les requêtes vers vitam pour éviter les vulnaribilités et faille de sécurité. Le fitre sera contrôler : - le Header de la requête - le Parameter URI - le Body

1.2.13.13 Classes de filtres

Nous proposons trois filtres différents dans les classe comme suits :

- SanityCheckerCommonFilter.class : le filtre commmun pour contrôler le header, parametre URI et ceux de la requête. Ce filtre intègre aussi le contrôle XSS au niveau des header.
- SanityCheckerInputStreamFilter.class : filter body de type Inputstream
- SanityCheckerJsonFilter.class : filtre body de type Json

La logique est fait une contrôle et si c'est KO, une réponse de status 412 (PRECONDITION_FAILED) sera retourné.

1.2.13.14 Implémenter des filters

Le filtre sera ajouté dans registerInResourceConfig de chaque serveur application sous le syntaxe par exemple

```
serviceRegistry.register(AccessInternalClientFactory.getInstance())
    .register(SanityCheckerCommonFilter.class)
```

1.2.13.15 Appliquer le filtre pour Vitam

- le filtre commun SanityCheckerCommonFilter sera appliqué pour les modules suivants : AccessExternal, IngestExternal, Workspace, Metadata
- le filtre body Json SanityCheckerJsonFilter et body Inputstream SanityCheckerInputStreamFilter seront appliqué pour les modules AccessExternal, IngestExternal, Metadata

1.2.13.16 Présentation

Un filtre sur la valeur du tenant, passée dans les headers, a été ajouté pour pouvoir interdire toute requête n'indiquant pas de tenant, ou indiquant un tenant invalide.

1.2.13.17 Classe de filtre

Une classe de filtre a été ajoutée :

TenantFilter : on vérifie la présence du header X-Tenant-Id dans la requête. Ensuite, on s'assure que la valeur transmise est bien un Integer. Le contrôle est effectué, s'il est KO (tenant non valide), une réponse PRECONDITION_FAILED (code 412) sera retournée.

On vérifie ensuite la cohérence du X-Tenant-Id dans la requête, par rapport à la liste des tenants disponibles dans VITAM. Le contrôle est effectué, s'il est KO (tenant non présent dans la liste des tenants), une réponse UNAUTHORIZED (code 401) sera retournée.

1.2.13.18 Ajout du filtre

Le filtre est ajouté dans setFilter(ServletContextHandler context) de chaque serveur d'application :

```
// chargemenet de la liste des tenants de l'application
JsonNode node = JsonHandler.toJsonNode(getConfiguration().getTenants());
context.setInitParameter(GlobalDataRest.TENANT_LIST, JsonHandler.unprettyPrint(node));
context.addFilter(TenantFilter.class, "/*", EnumSet.of(
        DispatcherType.INCLUDE, DispatcherType.REQUEST,
        DispatcherType.FORWARD, DispatcherType.ERROR, DispatcherType.ASYNC));
```

1.2.13.19 Modules Vitam impactés

Le filtre sera appliqué pour les modules AccessExternal et IngestExternal.

1.2.13.20 Présentation

La configuration commune des serveurs de Vitam

1.2.13.20.1 Classe de configuration

DefaultVitamApplicationConfiguration contient 2 paramètres obligatoires :

_ Le nom du fichier de la configuration jetty

_ La liste des tenants applicatifs

1.2.13.20.2 Implémentation dans les serveurs de Vitam

- Les fichiers de configuration des serveurs doivent étendre cette configuration commune.

1.2.13.21 Implémentation de l'éxécution des requêtes mono-query DSL

1.2.13.21.1 Implémentation des query builder

Pour construire dynamiquement une requête mono-query, on peut utiliser les builders proposés ci-dessous :

Insert : [filter, data]

- Il élabore la requête d'insertion. Il contient le filtre et les données à insérer

Select : [query, filter, projection]

- Il élabore la requête de recherche. Contient le query, le filtre et la projection

Update : [query, filter, actions]

- Il élabore la requête de mise à jour. Contient le query, le filtre et les actions

Delete : [query, filter]

- Il élabore la requête de suppression. Contient le query, le filtre

```
Select selectQuery = new Select(requestInJson)
ou
Select selectQuery = new Select().setQuery(query).setfilter(filter).setData(data);
```

```
Update updateQuery = new Update(requestInJson)
ou
Update updateQuery = new Update().setQuery(query).setfilter(filter).addActions(data);
```

```
Insert insertQuery = new Insert(requestInJson)
```

ou

```
Insert insertQuery = new Insert().setData(data).setfilter(filter);
```

```
Delete deleteQuery = new Delete(requestInJson)
ou
Delete deleteQuery = new Delete().setQuery(query).setfilter(filter);
```

1.2.13.21.2 Implémentation de DbRequestSingle

DbRequestSingle est une classe pour éxécuter les requêtes DSL mono-query.

Pour l'initialiser, il faut utiliser le constructeur avec une collection de Vitam.

Le résultat de l'éxécution est un objet DbRequestResult qui contient les informations suivantes :

- boolean wasAcknowledged : l'information reconnue pour la suppression et la mise à jour
- long count : le nombre d'éléments insérés, trouvés, supprimés ou mis à jour
- Map<String, List<String>> diffs : la différence entre ancien et nouvelle valeur de l'action mise à jour
- MongoCursor<VitamDocument< ?>> cursor : le cursor mongo de l'opération de recherche

```
DbRequestSingle dbrequest = new DbRequestSingle(collection.getVitamCollection());
Insert insertquery = new Insert();
insertquery.setData(arrayNode);
DbRequestResult result = dbrequest.execute(insertquery);
```

L'implémentation du sort est disponible sur les requêtes MongoDB et ElasticSearch.

1.2.13.22 Implémentation de l'authentification

1.2.13.22.1 Implémentation de l'authentification (MongoDbAccess)

L'authentification est le processus de vérification de l'identité du client, donc vous avez besoin d'utiliser quatre paramètres dans la fichier de configuration

"dbAuthentication", "dbUserName", "dbName", "dbPassword"

La gestion de l'authentification doit être débrayable – Si "dbAuthentication" est égal à "false", il doit être possible de continuer à utiliser des bases de données Mongo sans authentification.

Si "dbAuthentication" est égal à "true", il faut créer le MongoClient contenant MongoCredential qui représente les informations d'identification pour l'authentification auprès d'un serveur mongo, ainsi que la source des informations d'identification et le mécanisme d'authentification à utiliser.

Ici, Les utilisateurs "dbUserName" se lier à une base de données spécifique "dbName". Il a besoin de mot de passe "dbPassword" pour entrer le base et CRUD.

```
   public static MongoClient createMongoClient(DbConfiguration configuration,␣
↪MongoClientOptions options) {
       List<MongoDbNode> nodes = configuration.getMongoDbNodes();
       List<ServerAddress> serverAddress = new ArrayList<ServerAddress>();
       for (MongoDbNode node : nodes){
           serverAddress.add(new ServerAddress(node.getDbHost(), node.getDbPort()));
       }
       if (configuration.isDbAuthentication()) {
           // create user with username, password and specify the database name
           MongoCredential credential = MongoCredential.createCredential(
               configuration.getDbUserName(), configuration.getDbName(),␣
↪configuration.getDbPassword().toCharArray());

           // create an instance of mongoclient
           return new MongoClient(serverAddress, Arrays.asList(credential), options);
       } else {
           return new MongoClient(serverAddress, options);
       }
   }
   ......................


--     List<ServerAddress> serverAddress:

       La liste des adresses du serveur qui permet la base de données mongodb de␣
↪connecter plusieurs nœuds
--     Arrays.asList(credential):

       La liste des informations d'identification que ce client authentifie toutes␣
↪les connexions avec
```

1.2.13.23 Implémentation du secret de la plateforme

1.2.13.23.1 Présentation

Le secret de plateforme permet de se protéger contre des erreurs de manipulation et de configuration en séparant les environnements de manière logique (secret partagé par l'ensemble de la plateforme mais différent entre plateforme).

1.2.13.23.2 Implémentation

- Un Header X-Request-Timestamp contenant le timestamp de la requête sous forme epoch (secondes depuis 1970)
- Un Header X-Platform-ID qui est SHA256("<methode>;<URL>;<Valeur du header X-Request-Timestamp>;<Secret partagé de plateforme>").

  Par contre, mettre le secret de plateforme à la fin permet de limite les attaques par extension.

```
// add Authorization Headers (X_TIMESTAMP, X_PLATFORM_ID)
Map<String,String> authorizationHeaders = AuthorizationFilterHelper.
↪getAuthorizationHeaders(httpMethod,baseUri);
if(authorizationHeaders.size()==2){
    builder.header(GlobalDataRest.X_TIMESTAMP,authorizationHeaders.get(GlobalDataRest.
↪X_TIMESTAMP));
    builder.header(GlobalDataRest.X_PLATFORM_ID,authorizationHeaders.
↪get(GlobalDataRest.X_PLATFORM_ID));
}
.....................
```

Si on veut assurer une sécurité additionnelle, il est possible de transmettre un hash des valeurs suivantes :

- URI + paramètres de l'URI
- Header Timestamp
- Secret de plateforme en clair non transmis (connus par les participants de la plateforme)

=> Hash (URI + paramètres (dans l'ordre alphabétique) + Header Timestamp + secret non transmis) Ce Hash est transmis dans le Header : X-Platform-Id

```
//encode URL using secret
public static String encodeURL(String httpMethod, String url, String timestamp,␣
↪String secret,
    DigestType digestType) {
    ParametersChecker.checkParameter(ARGUMENT_MUST_NOT_BE_NULL, httpMethod, url,␣
↪timestamp, secret, digestType);
    Digest digest = new Digest(digestType);
    return digest.update(httpMethod + DELEMITER_SEPARATED_VALUES + url + DELEMITER_
↪SEPARATED_VALUES + timestamp +
            DELEMITER_SEPARATED_VALUES + secret).toString();
}
.....................
```

Le contrôle est alors le suivant :

1. Existance de X-Platform-Id et Timestamp
2. Vérification que Timestamp est distant de l'heure actuelle sur le serveur requêté de moins de 10 secondes (**|Timestamp - temps local|** < 10 s)
3. Calcul d'un Hash2 = Hash(URI+paramètres (dans l'ordre alphabétique) + Header Timestamp + secret non transmis) et vérification avec la valeur Hash transmise

```
if ((Strings.isNullOrEmpty(platformId)) || (Strings.isNullOrEmpty(timestamp))) {
    return false;
} else {
    return (checkTimestamp(timestamp) && (checkPlatformId(platformId, timestamp)));
}
private boolean checkTimestamp(String timestamp) {
     ParametersChecker.checkParameter(ARGUMENT_MUST_NOT_BE_NULL, timestamp);
     long currentEpoch = System.currentTimeMillis() / 1000;
     long requestEpoch = Long.valueOf(timestamp).longValue();
     if (Math.abs(currentEpoch - requestEpoch) <= VitamConfiguration.
↪getAcceptableRequestTime()) {
         return true;
     }

     LOGGER.error("Timestamp check failed");
     return false;
}
private boolean checkPlatformId(String platformId, String timestamp) {
     ParametersChecker.checkParameter(ARGUMENT_MUST_NOT_BE_NULL, platformId,␣
↪timestamp);
     String uri = getRequestURI();
     String httpMethod = getMethod();
     String code = URLCodec.encodeURL(httpMethod, uri, timestamp, VitamConfiguration.
↪getSecret(),
         VitamConfiguration.getSecurityDigestType());
     if (code.equals(platformId)) {
         return true;
     }
     LOGGER.error("PlatformId check failed");
     return false;
}
```

1.3 Functional administration

1.3.1 Introduction

L'ensemble de ces documents est le manuel de développement du module functional-administration, qui représente le métier fonctionnel de l'user story #71 de projet VITAM, dont le but et de réaliser des opérations sur le format référentiels de fichier auprès de la base de données (insert/recherche (par id ou par condtion)/delete).

Le manuel se compose de deux parties - DAT présente l'architecture technique du module au niveau des packages, classes - REST-RAML explique comment on utitlise des différents service proprosés par module - détail d'utilisation du client

1.3.2 DAT : module functional-administration

Ce document présente l'ensemble du manuel développement concernant le développment du module functional-administration qui identifie par la user story #71, qui contient :

- modules & packages
- classes métiers

---

1. Modules et packages

functional-administration

|— functional-administration-common : contenant des classes pour des traitements communs concernant le format référentiels, l'opération auprès de la base de données |— functional-administration-format : fournir des traitements de base pour les formats référentiels de VITAM

|— functional-administration-format-api : définitions des APIs |— functional-administration-format-core : implémentations des APIs |— functional-administration-format-import

|— functional-administration-rule : fournir des traitements de base pour la gestion de règles administratives

|— functional-administration-rule-api : Définition des APIs |— functional-administration-rule-core : Impélmentation des APIs

|— **functional-administration-accession-register** [fournir des traitements de base pour la gestion des registres de fonds] |— functional-administration-accession-register-core : Impélmentation des traitements des registres de fonds

|— functional-administration-rest : le serveur REST de functional-administration qui donnes des traitement sur les traitements de format référentiel et gestion de règles administratives. |— functional-administration-client : client functional-administration qui sera utilisé par les autres modules pour les appels de traitement sur le format référentiel & gestion de règles. |— functional-administration-contract : fournis les traitements de base pour les contrat d'accès et les contrat d'entrées |— functional-administration-profile : fournis les traitements de base pour les profile.

|— functional-administration-context : fournis les traitements de base pour les contextes

2. Classes métiers

Dans cette section, nous présentons quelques classes principales dans des modules/packages abordés ci-dessus.

2.1. functional-administration-common :

fr.gouv.vitam.functional.administration.common

- FileFormat.java :

une extension de VitamDocument définissant le référentiel des formats.

- ReferentialFile.java :

interface définissant des opérations liées au référentiel des format : importation du fichier PRONOM, vérificaton du fichier PRONOM soumis, recherche d'un format existant et suppression du référentiel des formats.

- IngestContract.java :

Le modèle de données des contracts d'entrée, ce modèle étend VitamDocument.

- AccessContract.java :

Le modèle de données des contracts d'accès, ce modèle étend VitamDocument.

- Profile.java :

Le modèle de données des profiles, ce modèle étend VitamDocument.

- Context.java :

Le modèle de données des contextes, ce modèle étend VitamDocument.

fr.gouv.vitam.functional.administration.common.embed ProfileFormat.class : Une enum embeded dans le profile qui sert à définir le format du fichier profile (xsd, rng) ProfileStatus.class : Une enum embeded dans le profile qui sert à définir le status (ACTIVE, INACTIVE)

fr.gouv.vitam.functional.administration.common.exception : définir des exceptions concernant de opération sur le référentiel des formats

fr.gouv.vitam.functional.administration.common.server les classe de traitement auprès de la base de données mongodb pour les opérations de référentiel de format.

- FunctionalAdminCollections.java :

définir la collection dans mongodb pour des données de formats référentiels

- MongoDbAccessReferential.java :

interface définissant des opérations sur le format de fichier auprès de la base mongodb : insert d'une base de PRONOM, delete de la collection, recherche d'un format par son Id dans la base,recherche des format par conditions

- MongoDbAccessAdminImpl.java :

une implémentation de l'interface MongoDbAccessReferential en extension le traitementMongoDbAccess commun pour mongodb

2.2. functional-administration-format

- functional-administration-format-api
- functional-administration-format-core
- PronomParser.java : le script de traitement permettant de de récupérer l'ensemble de format en format json depuis d'un fichier PRONOM stantard en format XML contient des différents formats référentiels
- ReferentialFormatFileImpl.java : implémentation de base des opération sur le format référentiel de fichier à partir d'un fichier PRONOM jusqu'à la base MongoDB.
- functional-administration-format-import

2.3. functional-administration-rest

- AdminManagementResource.java : définir des ressources différentes pour le serveur REST functional-administration
- AdminManagementApplication.java : créer & lancer le serveur d'application avec une configuration
- ContractResource.java : Définir l'endpoints de l'api rest des contrats (entrée et accès)
- ProfileResource.java : Définir l'endpoint de l'api rest du profile
- ContextResource.java : Définir l'endpoint de l'api rest du contexte

2.4. functional-administration-client

- AdminManagementClientRest.java : créer le client de et des fonctionnalités en se connectant au serveur REST
- AdminManagementClientMock.java : créer le client et des fonctionnalités en se connectant au mock de serveur

2.5. functional-administration-rules

- functional-administration-rules-api
- functional-administration-rules-core
- RulesManagerParser.java :permett de de parser le fichier de référentiel de règle de gestion d'extension .CSV et récupérer le contenu en ArrayNode
- RulesManagerFileImpl.java : implémentation de base des opération sur les paramètres de référentiel de regle de gestion à partir de l'array Node générer après le parse de CSV File jusqu'à la base MongoDB.

Le contrôle au niveau de RulesManagerFileImpl de fichier CSV a été mis à jour .

Définition d'un référentiel valide en se basant sur les critères ci_dessous :

Chaque RuleId doit être UNIQUE dans le référentiel RuleType doit être dans l'énumération suivante, non sensible à la casse : (AppraisalRule, AccessRule, StorageRule, DisseminationRule, ClassificationRule, ReuseRule) RuleDuration :

- Depuis le fichier CSV, peut être un entier positif ou nul ou "unlimited" (insensible à la casse). La valeur réelle de l'enregistrement dans la collection est laissée à la discrétion des équipes de développements (ex "-1" si on veut garder une valeur numérique)
- Permettre les manipulations sur des nombres (plus grand que.. plus petit que... Et calcul de date). Actuellement le champ est de type string, ce qui semble poser de nombreuses contraintes

RuleMeasurement :

RuleMeasurement doit être dans l'énumération suivante, non sensible à la casse : (year, month, day) RuleMeasurement peut aussi avoir comme valeur, non sensible à la casse "second". Cette demande est dans l'optique de la story #740 et n'a de sens qu'à des fins de tests. L'association de RuleDuration et RuleMeasurement doit être inférieure ou égale à 999 ans. (Mettre "15000 jours est donc autorisé)

L'unité de mesure (RuleMeasurement) doit être écrite en français dans l'interface, comme c'est déjà le cas actuellement : année(s), mois, jour(s), seconde(s)

Dans le cas des règles unlimited

- La valeur que doit renvoyer l'API lorsque la règle a une durée 'unlimited' dépend du choix de design effectué pour l'enregistrement de la valeur 'unlimited'
- Dans l'IHM standard, la date de fin doit être au choix marquée comme :
- "Illimitée (date de début inconnue)" : dans le cas où la date de fin n'est pas connue car la startDate n'est pas connue
- "Illimitée (règle à durée illimitée)" : dans le cas où la date de fin ne peut pas être calculée car la durée de la règle est 'unlimited'

2.6. functional-administration-accession-register

- functional-administration-accession-register-api
- functional-administration-accession-register-core
- ReferentialAccessionRegisterImpl.java :implémentation de base des opération sur la collection registre de fond .

permet de créer une collection registre de fond et de faire la recherche par Service Producteur et l'affichage de détaile.

2.7. functional-administration-contract

fr.gouv.vitam.functional.administration.contract.api

- ContractService.java : Interface définissant les différentes opérations sur les contrats (contrat d'accès et contrat d'entrée)

fr.gouv.vitam.functional.administration.contract.core

- AccessContractImpl.java : Classe d'implémentation pour la gestion des contrats d'accès
- ContractStatus.java : Enum pour les différents status des contrat d'accès et des contrat d'entrées
- ContractValidator.java : Interface fonctionnelle de validations des contrats
- GenericContractValidator.java : Interface fonctionnelle de validations des contrats
- IngestContractImpl.java : Classe d'implémentation pour la gestion des contrats d'entrées

2.8. functional-administration-profile

fr.gouv.vitam.functional.administration.profile.api

- ProfileService.java : Interface définissant les différentes opérations sur les profiles.

fr.gouv.vitam.functional.administration.profile.api.impl

- ProfileServiceImpl.java : Implémentation du service ProfileService.

fr.gouv.vitam.functional.administration.profile.core

- ProfileManager.java : Gère toutes les opérations du logbook et toutes les opérations de validation concernant les profiles. Lors de la validation, il vérifie (si déjà existence dans la base de données, champs obligatoires, fichiers au format xsd ou rng valides, ..).
- ProfileValidator.java : Interface fonctionnelle de validations des contrats

2.8. functional-administration-profile

fr.gouv.vitam.functional.administration.context.api

-ContextService.java : Interface définissant les différentes opérations sur les contextes

fr.gouv.vitam.functional.administration.context.core

-ContextServiceImpl.java : Implémentation du Service ContextService -ContextValidator.java : Interface fonctionnelle de validations des contextes

1.3.3 Administration-Management-Common

Parent package : **fr.gouv.vitam.functional.administration**

Package proposition : **fr.gouv.vitam.functional.administration.common**

Ce package implémente les différentes opérations sur le module functional-administration (insert, delete, select pour les formats, les règles de gestion et les registres de fonds)

1.3.3.1 1. Modules et packages

—fr.gouv.vitam.functional.administration.common : contenant des modèles de document MongoDb — fr.gouv.vitam.functional.administration.common.client.model : contenant les modèles de la réponse du client —fr.gouv.vitam.functional.administration.common.exception : contenant les exceptions du module — fr.gouv.vitam.functional.administration.common.server : contenant les classes pour l'accès aux bases de données

1.3.3.2 2. Classes

Dans cette section, nous présentons quelques classes principales dans les modules/packages abordés ci-dessus.

1.3.3.2.1 2.1 Class ElasticsearchAccessFunctionalAdmin

Class ElasticsearchAccessFunctionalAdmin : il s'agit de la classe qui permet de gérer les requêtes de functional.administration à la base de données ElasticSearch Les différents traitements sont l'ajout, la recherche et la suppression.

Pour la recherche :

- La méthode search(final FunctionalAdminCollections collection, final QueryBuilder query, final QueryBuilder filter) permet de chercher dans l'index Elasticsearch avec le query et le filter.

Pour l'insert :

- La Méthode addIndex(final FunctionalAdminCollections collection) permet d'ajouter un index dans Elasticsearch
- La Méthode addEntryIndexes(final FunctionalAdminCollections collection, final Map<String, String> mapIdJson) permet d'insérer les indexes dans l'index ElasticSearch.

Pour le delete :

- La Méthode deleteIndex(final FunctionalAdminCollections collection) permet de supprimer un index dans Elasticsearch.

1.3.3.2.2 2.2 Class MongoDbAccessAdminImpl

- La Méthode insertDocuments(ArrayNode arrayNode, FunctionalAdminCollections collection)
  permet d'insérer un ensemble d'entrées dans mongodb et les indexe dans ElasticSearch (seulement pour les formats et les règles de gestion) .
- La Méthode MongoCursor< ?> findDocuments(JsonNode select, FunctionalAdminCollections collection)
  permet de chercher les documents dans mongoDb (pour les formats et les règles de gestion. On cherche d'abord dans Elasticsearch pour récupérer identifiant unique puis cherche dans mongoDb).
- La Méthode public void updateDocumentByMap(Map<String, Object> map, JsonNode objNode,FunctionalAdminCollections collection, UPDATEACTION operator)
  permet de mettre à jour un ensemble d'entrées dans les document mongodb et l'index ElasticSearch (seulement pour les formats et les règles de gestion).
- La Méthode public void updateData(JsonNode update, FunctionalAdminCollections collection)
  permet de mettre à jour une entrée dans un document mongodb via une requête au format json
- La Méthode deleteCollection(FunctionalAdminCollections collection)
  permet de supprimer un ensemble d'entrées dans monfoDb et l'index ElasticSearch (seulement pour les formats et les règles de gestion).

3. Mapping elasticsearch des documents (recherche rapprochée)

Cette section concerne le mapping elasticsearch des documents géré au niveau functional administration. Mais c'est la même règle partout ailleur.

Pour qu'un document soit analysé par elasticsearch et que la recherche rapprochée marche il faut ce qui suit :

- Ajouter un paramètre typeunique au document concerné. Ce paramètre est utilisé par elasticsearch.

Exemple : le document profile contient bien un paramètre :

```
public static final String TYPEUNIQUE = "typeunique";
...
```

- Créer dans le dossier resources les fichiers mapping au format json.

profile-es-mapping.json, accesscontract-es-mapping.json, ...

Exemple de fichier json de mapping elasticsearch :

```
{
  "properties": {
    "Name": {
      "type": "string"
    },
    "Status": {
      "type": "string",
```

```
      "index": "not_analyzed"
    },
    "CreationDate": {
      "type": "date",
      "format": "strict_date_optional_time"
    },
    "LastUpdate": {
      "type": "string",
      "index": "not_analyzed"
    }
  }
}
...
```

- Ces fichers sont ensuite chargé au niveau de ElasticsearchAccessFunctionalAdmin.
- Dans la méthode getMapping de ElasticsearchAccessFunctionalAdmin, il faut rajouter le document concerné, ainsi récupérer le mapping correspondant.

```
private String getMapping(FunctionalAdminCollections collection) throws IOException {
    if (collection.equals(FunctionalAdminCollections.PROFILE)) {
        return ElasticsearchUtil.transferJsonToMapping(FileRules.class.
↪getResourceAsStream(MAPPING_PROFILE_FILE_JSON));
    }
    return "";
}
...
```

- Dans la méthode getTypeUnique ajouter TYPEUNIQUE du document concerné.

```
private String getTypeUnique(FunctionalAdminCollections collection) {
    if (collection.equals(FunctionalAdminCollections.PROFILE)) {
            return PROFILE.TYPEUNIQUE;
    }
    return "";
}
...
```

Attention :

- Il faut supprimer l'index s'il existe déjà pour qu'il puisse être crée avec les bon mapping.
- Si on supprime l'index il faut re-indexer les données de la base de données.

1.3.4 Administration-Management-client

1.3.5 Utilisation

1.3.5.1 Paramètres

Administration-Management-client-format Les paramètres sont les InputStreams du fichier Pronom pour l'import ou la validation. Pour la recherche des formats, les paramètres sont les requête DSL construites par les builders de common-database

Administration-Management-client-rules Les paramètres sont les InputStreams du fichier CSV pour l'import ou la validation. Pour la recherche des règles, les paramètres sont les requête DSL construites par les builders de common-database

Administration-Management-client-accession-register Les paramètres sont les InputStreams du fichier pour l'import ou la validation. Pour la recherche des registres, les paramètres sont les requête DSL construites par les builders de common-database

1.3.5.2 La factory

Afin de récupérer le client, une factory a été mise en place.

```
// Récupération du client
AdminManagementClient client = AdminManagementClientFactory.getInstance().
↪getAdminManagementClient();
```

1.3.5.2.1 Le Mock

Par défaut, le client est en mode Mock. Il est possible de récupérer directement le mock :

```
// Changer la configuration du Factory
AdminManagementClientFactory.setConfiguration(AdminManagementClientFactory.
↪AdminManagementClientType.MOCK_CLIENT, null);
// Récupération explicite du client mock
AdminManagementClient client = AdminManagementClientFactory.getInstance().
↪getLogbookClient();
```

1.3.5.3 Le client

Pour instancier son client en mode Production :

```
// Ajouter un fichier functional-administration-client.conf dans /vitam/conf
// Récupération explicite du client
AdminManagementClient client = AdminManagementClientFactory.getInstance().
↪getAdminManagementClient();
```

Le client propose actuellement 18 méthodes :

```
Status status();
void checkFormat(InputStream stream);
void importFormat(InputStream stream);
void deleteFormat();
JsonNode getFormatByID(String id);
JsonNode getFormats(JsonNode query);
checkRulesFile(InputStream stream);
importRulesFile(InputStream stream);
deleteRulesFile();
JsonNode getRuleByID(String id);
JsonNode getRule(JsonNode query);
createorUpdateAccessionRegister(AccessionRegisterDetail register);
JsonNode getAccessionRegister(JsonNode query);
JsonNode getAccessionRegisterDetail(JsonNode query);

Status importContexts(List<ContextModel> ContextModelList)
RequestResponse<ContextModel> updateContext(String id, JsonNode queryDsl)
RequestResponse<ContextModel> findContexts(JsonNode queryDsl)
RequestResponse<ContextModel> findContextById(String id)
```

1.4 IHM demo

1.4.1 Introduction

1.4.1.1 But de cette documentation

L'ensemble de ces documents est le manuel de développement du module IHM-logbook, qui représente le métier fonctionnel de US story #90 de projet VITAM, dont le but et de faire affichers des logs des opérations et effectuer la recherche sur ces logs.

Le manuel se compose de deux parties - DAT présente l'architecture technique du module au niveau des packages, classes - REST-RAML explique comment on utitlise des différents service proprosés par module

1.4.2 IHM Front

1.4.2.1 Cette documentation décrit la partie front/Angular de l'ihm et en particulier sa configuration et ses modules.

1.4.2.1.1 Utils et général / Composition du projet Angular

TODO

1.4.2.1.1.1 Composition du projet

NPM + Bower : npm est utilisé pour gérer les dépendances liées au build de l'application (par exemple la minification), tandis que bower est utilisé pour gérer les dépendances de l'appication (par exemple angular, moment.js, ...) Pour construire le projet, l'outil gulp a été mis en place. Celui permet d'automatiser les tâches permettant d'arriver à la construction d'un livrable contenant les fichiers html, javascript et css minifiés. La commande 'gulp package' permet de construire le projet.

Tests unitaires : Voir ihm-tests.rst

1.4.2.1.1.2 Gulp et déploiement à chaud

Le déploiement à chaud est possible via la commande 'gulp serve'. Si un fichier est modifié pendant que le serve est lancé, les modifications seront automatiquement mises à jour. Le backend ciblé peut être spécifié en ajoutant un fichier local.json (Voir local.json.sample) et en modifiant la propriété target.

1.4.2.1.1.3 Karma et Tests unitaires

Les tests unitaires se lances via les commandes :

- 'gulp tests' : Lance un serveur (basé sur le module karma serveur) + les tests karma (Unitaires) et Protractor (e2e)
- 'gulp testKarma' : Lance les tests unitaires seules (Nécéssite un serveur lancé)
- 'gulp testProtractor' : Lance les tests end to end seuls (Nécéssite un serveur lancé)

1.4.2.1.1.4 Qualité du code Javascript

La qualité du code javascript est validée grâce au module lint. Pour celà, il suffit de lancer la commande 'gulp lint'.

1.4.2.1.1.5 Modèle MVC

Le front va petit à petit être migré vers une architecture reprenant le modèle MVC :

- Une couche Modèle, récupérant les données depuis l'API vitam (Server d'app + dossier resources Angular)
- Une couche Service, traitant les promesses des resources ou proposant des méthodes utilitaires
- Une couche Vues, proposant l'affichage d'une page avec son controller associé

Au final l'arbo type des fichiers devrait être la suivante :

/app => Fichiers de conf globaux du projet (bower/npm/jshint/index.html/...)

/core => Fichiers de configuration globaux (core.module.js, main.controller.js, app.config.js, app.module.js, ...) /static => Fichiers de traductions front (Key=value pour les champs statiques de l'IHM) /services => Services utilitaires partagés de l'application (Faire des modules pour chaque services si externalisables) /directives => Directives utilitaires partagés de l'application (Faire des modules pour chaque directives si externalisables) /filters => Filtres utilitaires partagés de l'application ?

/resources accession-register.resource.js => Une méthode par endpoint du server d'app (Search / GetAll / GetDetails / ...) archive-unit.resource.js => Une méthode par endpoint du server d'app (Search / GetArchive / GetDetails / ...) ...

/services accession-register.service.js => Une ou plusieurs méthodes par méthode de la resource fund-register archive-unit.service.js => Une ou plusieurs méthodes par méthode de la resource archive-unit ...

/pages (Nom à valider)

/accession-register-detail => Controller + Template de la page Détails de Registre de Fonds /archive-unit => Controller + Template de la page archive-unit ...

/styles /css /img /fonts => A migrer dans le /css ?

1.4.2.1.1.6 Internationalisation

Cette partie est gérée par le module angular-translate

Pour ajouter une traduction, ajouter une clé valeur dans le fichier static/languages_<lang>.json L'entrée être formatée de la manière suivante "<pageGroup>.<pageDetail>.<bloc>.<key>"="<value>" où :

- <pageGroup> est le groupe de page ou du module dans l'application (Exemple archiveSearch ou administration)

- <pageDetail> est le nom de page dans le groupe (Exemple managementRules ou archiveUnitDetails)
- <bloc> est le nom du bloc dans la page (Exemple searchForm ou technicalMetadata)
- <key> est le nom de la clé (Exemple 'id' ou 'evDetData')

  Si possible essayez de regrouper les clés définies par groupe/detail/bloc/ordre alphabetique pour s'y retrouver.

Pour utiliser une traduction, utilisez dans une instruction angular de votre HTML le filtre translate :

```
<div>{{'archiveSearch.searchForm.id' | translate}}</div>
```

Si votre key est dynamique et présente dans une variable, il est possible d'inserer du js en plus de la chaine :

```
<div>{{'archive.archiveUnitDetails.technicalMetadata.' + metadata[$index] | translate}
↪}</div>
```

Enfin il est également possible de faire le traitement de traduction en js en appliquant le filtre :

Note : $filter doit avoir été injecté

```
var translatedLabel = $filter('translate')('archiveSearch.searchForm.id');
```

À faire

Rendre dynamique la langue choisi pour les traductions (actuellement static FR)

À faire

Utiliser la langue de fallback fr (ou autre ?)

À faire

Une grosse partie des constantes (js) et des String statiques (html) devraient être mises dans ces fichiers

À faire

Récupérer la liste des valeurs du référentiel VITAM (Build / Appel API)

1.4.3 Modules IHM Front

1.4.3.1 Cette documentation décrit les principaux modules réutilisables de l'IHM front (js)

1.4.3.1.1 Module archive-unit

Ce module ne comprends pas le module 'archive-unit-search' Ce module permet le processing et l'affichage des données liées à une Archive Unit. Les directives utilisées sont :

- display-field qui permet d'afficher un champ en prenant en compte le mode édition

- display-fieldtree qui permet d'afficher un ensemble de champs en utilisant le directive display-field avec des paramètres standards pour chaque champ

1.4.3.1.1.1 Directive display-field

Cette directive permet d'afficher un champ 'simple' en mode visualisation ou edition. Un champ 'simple' est un champ qui à simplement une valeur (Texte/nombre) et pas de sous-élément.

Usages : Pour utiliser cette directive il suffit d'appeler la balise '<display-field' en spécifiant les parametres suivants :

- field-label : Surcharge du nom du label
- field-object : L'ensemble des propriétés de l'objet. Doit contenir au moins :

– isModificationAllowed : vrai si le champ est éditable – isFieldEdit : vrai si le champ est en cours d'édition – fieldValue : La valeur affichée du champ

- edit-mode : Vrai si le formulaire est en mode édition
- field-size : La valeur du XX dans la classe CSS de bootstrap col-md-XX.
- **intercept-user-change : Fonction de callback à appeler lorsque la champ est modifié** Cette fonction doit prendre un fieldSet en paramètres.

Il est également possible de donner une valeur de surcharge pour la valeur du champ grâce à ce dernier paramètre :

- display-value : Affiche une valeur spécifique à la place de fieldValue (Le mode édition reprends la valeur réelle)

Exemple :

```
<div class="col-xs-12">
   <div class="form-group col-md-6">
     <display-field field-label="'Service producteur'" field-size="'11'"
             intercept-user-change="$ctrl.interceptUserChanges(fieldSet)"
         field-object="$ctrl.mainFields['OriginatingAgency'].content[0]" edit-mode="
↪$ctrl.isEditMode">
     </display-field>
   </div
</div>
```

1.4.3.1.1.2 Directive display-fieldtree

Cette directive permet d'afficher un champ et leurs sous élément si nécessaire de manière récursive. - field-object : L'ensemble des propriétés de l'objet. Doit contenir au moins :

– isModificationAllowed : vrai si le champ est éditable – isFieldEdit : vrai si le champ est en cours d'édition – fieldValue : La valeur affichée du champ – typeF : Le type de champ

'P' correspond à un champs 'parent' avec des sous éléments. 'S' correspond à un champ simple.

– content : Tableau de fieldObject contenant les enfants de ce champ.

- edit-mode : Vrai si le formulaire est en mode édition
- intercept-user-change : Fonction de callback à appeler lorsque la champ est modifié.

  Cette fonction doit prendre un fieldSet en paramètres.

Exemple :

```
<div class="row archiveDesc panel-collapse collapse in" id="{{'box' + key}}">
     <div ng-repeat="fieldSet in $ctrl.managmentItems">
         <display-fieldtree intercept-user-change="$ctrl.
↪interceptUserChanges(fieldSet)"
             field-object="fieldSet" edit-mode="$ctrl.isEditMode">
```

```
        </display-fieldtree>
    </div>
</div>
```

1.4.3.1.1.3 Affichage des Libéllés des champs

La fonction self.displayLabel du controller archive-unit permet de récupérer la valeur française des champs à afficher.

- key : nom technique du champ à afficher
- parent : nom technique de son parent direct.

  permet de reconstituer la clé parent.key pour les champs 'parent'
- constantes : Nom du fichier de constantes à utiliser.

  Cela permet d'avoir plusieurs _id (par exemple) en fonction du context. Les fichiers de constantes sont définis dans archive-unit.constant.js. Les clés des constantes équivalent à "key" pour les champs simples et à 'parent.key' pour les champs parent.
- retourne le label si présent dans le fichier de constantes ou la clé (key) sinon.

Exemple :

```
var key = fieldSet.fieldId;
var parent = fieldSet.parent;
var constants = ARCHIVE_UNIT_MODULE_OG_FIELD_LABEL;
fieldSet.fieldName = self.displayLabel(key, parent, constants);
```

1.4.3.1.2 Affichage dynamiqueTable

Cette directive permet de dynamiser les tableaux de données pour sélectionner les colonnes à afficher.

- custom-fields : Ce sont les champs dynamiques pour le tableau.
  Ces objets doivent au moins avoir les champs 'id' (Valeur technique et unique) et 'label' (Valeur affichable à l'utilisateur).

  selected-objects : Ce sont les objets sélectionnés à afficher. L'objet en etrée peut être un tableau vide et sera nourri par la directive

Attention, pour des raisons d'ergonomie, il est demandé d'ajouter la classe CSS 'dynamic-table-box' au div 'panel-default' englobant. Cela permet à ce div de devenir dynamique et de dépasser de la page si plus de colones sont affichés. Ainsi la scrollbar horizontale est accessible directement.

1.4.3.1.3 Service de recherche

Le service ProcessSearchService (process-search.service.js) permet de factoriser les actions de recherche et de globaliser son fonctionnement. Tout écran de recherche doit l'utiliser.

Il met à disposition une fonction d'initialisation (initAndServe) du service de recherche qui renvoie 3 functions possibles :

- processSearch - Lance la requête HTTP et traite le comportement d'erreur si besoin (Affichage du message / vider les résultats / ...)
- reinitForm - Efface tout les champs de recherche pour reprendre les valeurs initiales des champs et relance une recherche (si besoin).

- onInputChange - Fonction qui peut être appelée par le contrôleur lors d'une modification d'un champ pour déclancher une réinitialisation de la recherche si le formulaire est revenu à son état initial.

Aussi, en plus des autres paramètres (voir JS doc de la fonction initAndServe), l'initialisation prends en paramètre un objet 'searchScope' qui doit être lié au scope et doit être de la forme suivante :

```
searchScope = {
   form: {/* Valeurs initiales des champs de recherche (seront donc mises à jour par
↪la vue et par le service) */},
   pagination: { /* Valeurs des variables de pagination */ },
   error: { /* Mise à jour des message d'erreur */ },
   response: { /*  */ }
}
```

Ce service permet d'effectuer les actions suivantes de manière uniforme quelque soit le controller qui l'appelle :

- Obliger d'utiliser la chaîne de fonctions fournies (Evite d'avoir une implem differente sur chaque controller)
- Gérer la réinitialisation des messages d'erreur lors du lancement d'une nouvelle recherche (searchScope.error)
- Gérer la réinitialisation du nombre de résultats lors de chaque recherches (searchscope.response)
- Gestion de la recherche automatique à l'initialisation de la page (Ou à la réinitialisation du formulaire)

Par la suite, ce service pourra être complété par des directives (liste non exaustive) pour automatiser l'affichage des informations similaires :

- Messages d'erreur (On peut imaginer une directive à assossier à un formulaire qui affiche les boutons d'effactement multi-champs, le bouton de résultat et le message d'erreur en se basant sur le searchScope.form et searchScope.error)
- Affichage des résultats (On peut imaginer une directive se basant sur searchScope.response déffinissant un pattern pour le tableau de résultat et le titre + Nb résultats).
- Gestion de la pagination (On peut imaginer une directive se basant sur le searchScope.pagination et searchScope.response pour calculer les éléments de pagination).

1.4.3.1.4 Service d'affichage des mesures d'un objet physique

Le service uneceMappingService à pour but d'aller chercher les unitées de mesures contenu dans le fichiers unece.json pour l'afficher dans une valeur compréhensible pour les utilisateurs

1.4.4 IHM Front - Tests

1.4.4.1 Cette documentation décrit la partie tests (unitaires et end to end) du front/Angular de l'ihm.

Il est possible de lancer tout les tests via la commande *gulp tests* (Protractor nécessite Chrome). Un *npm install* est nécessaire.

1.4.4.1.1 Tests unitaires

1.4.4.1.1.1 Installation / Lancement des tests unitaires

Karma est installé automatiquement avec les autres dépendances via la commande *npm install*. Le lancement des tests s'effectue vià la commande *gulp testKarma*

1.4.4.1.1.2 Informations sur la configuration des tests unitaires

La configuration des tests unitaires se trouve dans webapp/karma.conf.js

En particulier, la partie 'files' définit les fichiers à charger pour les tests unitaires. Il sera nécessaire d'en ajouter lors de l'ajout de prochaines fonctionnalités et tests unitaires.

1.4.4.1.1.3 Exemples de tests unitaires

4 samples de tests ont étés implémentés pour montrer ce qu'il est globalement possible de faire :

Base beforeEach (Charger un service) / Test de retour de valeur en fonction du paramètre
: Exemples : date-validator.service.js / response-validator.service.js

Espion SpyOn permettant de vérifier qu'une fonction est bien appelée comme il faut
: Exemple : load-static-value.service.js (Test nombre appel) / response-validator.service.js (Bon paramètres)

HTTPMock httpBackend permettant de mocker un appel rest / afterEach permettant de vérifier les appels traités
: Exemple : accession-register.service.js

CallMock initialisation d'un controller / mock de l'appel des méthodes d'un service / cohérence des résultats
: accession-register-details.controller.js

1.4.4.1.2 Tests end to end

1.4.4.1.2.1 Initialisation / Lancement des tests e2e

Pour le moment, il est nécessaire d'avoir un environement lancé dans le serveur d'App pour servir les resources. Un gulp serve devrait régler le problème.

[Inutile si lancé via gulp]Installation de protractor

```
npm install -g protractor@2
protractor --version
```

Cette commande devrait avoir installer un 'webdriver-manager' (Sélénium).

[Inutile si lancé via gulp]Il est nécessaire de le mettre à jour et de le lancer pour lancer les tests e2e.

```
node_modules/protractor/bin/webdriver-manager update
node_modules/protractor/bin/webdriver-manager start
```

Si une erreur 'info : driver.version : unknown' est remontée, vérifier la compatibilité entre votre navigateur Chrome et son plugin ChromeDriver. Si besoin, modifiez le fichier webapp/node_modules/protractor/config.json, et mettez à jour la propriété "chromedriver" avec une valeur compatible (2.27 pour les plus récent). Cette modification 'hardcodded' doit être faite après chaque mise à jour de npm (npm install).

[Inutile si lancé via gulp]Le lancement des tests end to end se font grâce à la commande suivante :

```
protractor protractor.conf.js
```

Il est également possible de le lancer via gulp via la commande :

```
gulp testProtractor
```

Il est possible de surcharger divers arguments grâce aux arguments suivants (donnés à titre d'exemple : - –baseUrl='http://localhost:8082/ihm-demo/#!' (Permet de modifier l'URL de base utilisée. Peut par exemple servir a lancer les tests e2e sur le serveur de recette. - –params.<paramName>='<paramValue>' (Permet de modifier un paramètre de la configuration protractor (params) - –suite=<maSuite> (Permet d'utiliser seulement une ou plusieurs suites de tests plutôt que de lancer toute la baterie.

Ces paramètres sont aussi settables dans le json de configuration gulp de la tache testProtractor.

1.4.4.1.2.2 Informations sur la configuration des tests e2e

La configuration définit des batteries de tests (suites). Lors de l'ajout d'un test e2e, il est nécessaire d'ajouter une entrée dans les suites en précisant les fichiers à éxécuter.

La configuration permet aussi de : - Définir un login/password (Via la surcharge des params userName/password) - Utiliser ou non le mode mock http (Via la surcharge du param mock)

1.4.4.1.2.3 Exemple d'utilisation des outils e2e

Création de fonctions réutilisables dans chaque tests :

- Création d'un fichier utils/*.function.js
- Création d'une fonction éxportée via module.exports
- Import des fonctions dans le test via require('./path/to/file') ;

Sélection des éléments

- Sélection d'une balise a laquelle le modèle associé est variable.name (<input ng-model="variable.name" />)

– element(by.model('variable.name'))

- Sélection d'une balise grâce à son identifiant ( )

– element(by.id('navbar')) ;

- Sélection d'une balise contenant un attribut 'type' et une valeur 'submit' (<button type="submit" />)

– element(by.css('[type="submit"]'))

- Sélection d'une balise grâce à son tag (
)

– element(by.css('ul')) ;

- Sélection multiple d'éléments (
)

– element.all(by.css('li')) ;

- Sélection d'un sous élément (xxx yyy <button/>)

– var div = element(by.css('div')) ; – div.element(by.css('button')) ; / div.all(by.css('p')) ;

- Sélection d'une partie d'un ensemble d'éléments (xxx yyy zzz)

– var ps = element.all(by.css('p')) ; – var firstP = ps.first() ; // xxx – var pNumber1 = ps.get(1) ; // yyy – var lastP = ps.last() ; // zzz

Conclusion :

- Selection classique : element(by.xxx()) ;
- Sélection multiple : element.all(by.yyy()) ;
- Sélections Chaînées : element(by.xxx()).all(by.yyy()).get(2).element(by.zzz()) ;

Récupérations des propriétés configurés dans protractor.conf.js :

- browser.baseUrl (L'url configurée)
- browser.params.paramName (Récupère le paramètre paramName)

Actions / promise et Expects :

- Les actions sur un élément (item.click() / item.count() / ...) renvoient une promise qu'il faut traiter dans un then si on veut enchainer une action ou récupérer une valeur.
- Les expects expect(item.count())toBe(2) ; traitent la promise de la bonne manière pour comparer la valeur.

Mock HTTP :

- Exemple simple dans login ou on configure le httpMocker dans beforeEach si le mode mock est activé.
- Exemple plus complexe dans accession-register où on renvoie une réponse en fonction des paramètres.

1.4.5 DAT : module IHM logbook operations

Ce document présente l'ensemble de manuel développement concernant le développment du module ihm-demo qui représente le story #90, qui contient :

- modules & packages
- classes métiers

---

1. Modules et packages

Au présent : nous proposons le schéma ci-dessous représentant le module principal et ses sous modules.

ihm-demo

|—ihm-demo-core : le traitement essentiel pour construire des requêtes DSL depuis des données saisies dans l'interface web |—ihm-demo-web-application : le serveur d'application web qui fournit des services au client pour les traitements sur la recherche de logbook des opérations

Depuis ces deux modules, nous proposons deux packages correspondants :

ihm-demo-core –> fr.gouv.vitam.ihmdemo.core ihm-demo-web-application –> fr.gouv.vitam.ihmdemo.appserver ihm-demo-web-application –> webapp (resources)

2. Classes de métiers

Cette section, nous présentons les classes/fonctions principales dans chaque module/package qui permettent de réaliser l'ensemble de tâches requis par User Story #90.

2.1. Partie Backend

ihm-demo-core : CreateDSLClient.java La classe a pour l'objecttif de création d'une requête DSL à partir de l'ensemble de données de critères saisies depuis l'interface web du client. Les données en paramètres sont représentées dans un Map de type de String.

ihm-demo-web-application

- ServerApplication.java : créer & lancer le serveur d'application avec un configuration en paramètre
- WebApplicationConfig.java :

créer la configuration pour le serveur d'application en utilisant différents paramètres : host, port, context

- WebApplicatationResource.java :

définir des ressources différentes pour être utilisé par les contrôles de la partie Fontend. Le détail sur la resource de l'application serveur sera présenté dans le document RAML associé ihm-logbook-rest.rst.

2.1. Partie Frontend

La partie Fontend web se trouve dans le sous module ihm-demo-web-application. Ce fontend web est développé en AngularJS. Dans cette partie, nous trouvons des composants différents

- views
- modules
- controller js

1.4.6 ihm-demo

1.4.6.1 Présentation

Ce document présente le schéma de rest resources défini qui sera appelé par le backend de l'application web.

---

package :* **fr.gouv.vitam.api** | *Package proposition :* **fr.gouv.vitam.metadata.rest**

Module pour le module opération : api / fr.gouv.vitam.ihmdemo.appserver

1.4.6.2 Services

1.4.6.3 Rest API

URL Path : /

POST /archivesearch/units -> la recherche des métadata

POST /logbook/operations -> Recherche dans logbook par un nom (critère).
Cela retourne l'ensemble de logbook opération (avec id opération)
POST /logbook/operations/{idOperation} –> Recherche de logbook de l'opération
de logbook par idOperation
POST /admin/formats -> Recherche des formats par DSL
POST /admin/formats/{idFormat} -> Recherche d'un formats par son id
POST /format/check -> Validation du fichier PRONOM
POST /format/import -> Import des formats dans le fichier PRONOM
DELETE /format/delete -> Supprimmer tous les format dans la base de données
PUT /archiveupdtae/units/{id} -> update des métadata
POST /admin/accession-register -> Recherche dans AccessionRegisterSummary par un critère potentiel
Cela retourne une liste d'AccessionRegisterSummary (Si recherche par Service producteur, liste de 1 élément)
POST /admin/accession-register/detail -> Recherche dans AccessionRegisterDetail par un id de service producteur (critère)
Cela retourne une liste d'AccessionRegisterDetail correspondant au Service producteur donné

1.4.7 IHM Front - Requêtes HTTP et Tenant ID

1.4.7.1 Cette documentation décrit le process de récupération / sélection et communication du tenant ID depuis IHM-DEMO front vers les API publiques VITAM

1.4.7.1.1 Gestion du tenantId

1.4.7.1.1.1 Coté front

Actuelement, le tenantID est sauvegardé dans le navigateur client sous forme de cookie au moment de la connexion.

1.4.7.1.1.2 Coté serveur d'app

1.4.7.1.2 Création de requêtes HTTP utilisant un tenantID (front)

1.4.7.1.2.1 Utilisation de ihmDemoClient

Le service ihmDemoClient permet d'instancier un client HTTP préconfiguré pour dialoguer avec le serveur d'app IHM-DEMO. Ce dernier contient entre autre : - La racine de l'url à appeler (Exemple : ihm-demo/v1/api) - Un intercepteur permettant d'ajouter le HEADER X-request-id à chaque requêtes.

1.4.7.1.2.2 Requêtes http personnalisées

Si nécessaire il est possible d'utiliser $http ou un autre procédé pour faire votre requête HTTP. Dans ce cas, il est possible de récupérrer la clé et la valeur du header via la ligne de code suivante :

```
var key = loadStaticValues.loadFromFile().then(function(response) {
 return response.data.headers;
});
var value = authVitamService.cookieValue(authVitamService.COOKIE_TENANT_ID);
```

Note : Les services authVitamService et loadStaticValues doivent avoir été injectés.

1.4.8 Gestion des droits sur IHM demo

1.4.8.1 Cette documentation décrit la gestion des droits sur IHM-demo.

La gestion des droits (authorisations et habilitations) sur IHM demo (et VITAM en génénral) se fait grâce à shiro.

1.4.8.1.1 Gestion des autorisations

Les utilisateurs sont définis dans le fichier *shiro.ini* sous la forme d'un login suivi du mot de passé encodé avec l'algorithme md5.

1.4.8.1.2 Gestion des permissions

Sur chaque endpoint (couple URI / verbe HTTP), qui correpond à une méthode Java, on définit une permission grâce à l'annotation *RequiresPermissions*.

Par convention, la permission est nommée en fonction de l'URI est du verbe HTTP correspondant. Par exemple, la permission définissant la lecture sur l'URL */logbook* est : *logbook :read*. Si une l'URL possède une sous collection, par exemple */logbook/operations*, alors le nom de la permission pour lire les informations est : *logbook :operation :read*.

La correspondance entre les verbes HTTP et les permissions est la suivante : - GET : read - POST : update - PUT : create - DELETE : delete

Par contre, dans le cas ou on utilise un POST pour de la lecture (cas typique du DSL), on nommera quand même la permission avec *read*.

Au niveau du fichier shiro.ini, dans la section *roles*, on définit trois rôles (admin, user et guest), auxquels on associe les différentes permissions définies précédemment.

Enfin, dans la section *users*, on associe le rôle à un utilisateur.

1.4.9 IHM Filter for X-Request-ID

1.4.9.1 Description

En cas d'erreur technique, depuis le IHM demo, nous pouvons trouver le X-Request-ID affiché dans un popup. Le code d'erreur a une valeur 500 renvoyé par les APIs externes.

1.4.9.2 Côté serveur

Le filtre RequestIdContainerFilter (package fr.gouv.vitam.common.server.*) est ajouté dans l'application serveur IHM demo pour envoyer X-Request-ID dans le VitamSession en cas d'erreur. (fr.gouv.vitam.common.server.RequestIdHeaderHelper est mis à jour pour traiter des X-Request-ID en cas d'erreur)

1.4.9.3 Côté IHM Front

On ajoute aussi un intercepteur filter pour récupérer le X-Request-ID dans le cas d'erreur dans la session. On utilise d'un intercepteur angular sur $httpProvider.

1.5 Ingest

1.5.1 Introduction

L'ensemble de ces documents est le manuel de développement du module ingest, qui représente le métier fonctionnel de l'user story #84 de projet VITAM, dont le but et de réaliser des opérations sur le dépôt de document SIP vers la base de données MongoDb (upload SIP).

Le module est divisé en deux sous modules : ingest-internal et ingest-external. Le module ingest-internal fournnit les fonctionalités pour des traitements internes de la plate-forme Vitam, autrement dit il n'est visible que pour les appels internes de Vitam. Le module ingest-external fournit des services pour les appels extérieur de la plate-forme cela veux dire qu'il est visible pour les appels de l'extérieur de Vitam.

Le manuel se compose de deux parties - DAT présente l'architecture technique du module au niveau des packages, classes - REST-RAML explique comment on utitlise des différents service proprosés par module - détail d'utilisation du client

1.5.2 DAT : module ingest-internal

Ce document présente l'ensemble du manuel développement concernant le développment du module ingest-internal qui est identifié par le user story #84, qui contient :

- modules & packages
- classes métiers

---

1. Modules et packages

ingest-internal

|— ingest-internal-common : contenant des classes pour les traitements commons de modules ingest-internal |— ingest-internal-model : définir les modèles de données utilisé dans le module |— ingest-internal-api : définir des APIs de traitement dépôt des SIP vers le base MongoDb |— ingest-internal-core : implémentation des APIs |— ingest-internal-rest : le serveur REST de ingest-internal qui donnes des traitement sur dépôt de document SIP. |— ingest-internal-client : client ingest-internal qui sera utilisé par les autres modules interne de VITAM pour le service de dépôt des SIPs

2. Classes métiers

Dans cette section, nous présentons quelques classes principales dans des modules/packages qu'on a abordé ci-dessus.

ingest-internal-model :

-UploadResponseDTO.java : définir le modèle de réponse sur l'opération de dépôt SIP (upload). Il contient l'information sur le nom de fichier SIP, le code de retour VITAM, le code de retour HTTP, le message et le status.

ingest-internal-api :

-UploadService.java : interface pour le service de dépôt interne.

ingest-internal-core :

-MetaDataImpl.java : implémenter des fonctionnalités de traitement sur le métadata, pré-défini dans -MetaData.java

ingest-internal-rest :

- IngestInternalRessource.java : définir des ressources différentes pour le serveur REST ingest-internal
- IngestInternalApplication.java : créer & lancer le serveur d'application avec une configuration

ingest-internal-client

- IngestInternalClient.java : interface client IngestInternal
- IngestInternalInternalClientMock.java : mock client ingest-internal
- IngestInternalClientRest.java : le client ingest-internal et des fonctionnalités en se connectant au serveur REST

1.5.3 DAT : module ingest-external

Ce document présente l'ensemble du manuel développement concernant le développment du module ingest-external qui identifié par la user story #777 (refacto ingest), qui contient :

- modules & packages
- classes métiers

---

1. Modules et packages

ingest-external

|— ingest-external-common : contenant des classes pour les traitements commons de modules ingest-external : | code d'erreur, configuration et le script de scan antivirus |— ingest-external-api : définir des APIs de traitement dépôt des SIP vers le base | MongoDb |— ingest-external-core : implémentation des APIs |— ingest-external-rest : le serveur REST de ingest-external qui donnes des traitement | sur dépôt de document SIP. |— ingest-external-client : client ingest-external qui sera utilisé par les autres application externe de VITAM

2. Classes métiers

Dans cette section, nous présentons quelques classes principales dans des modules/packages qu'on a abordé ci-dessus.

2.1 ingest-external-common :

fr.gouv.vitam.ingest.external.common.util

-JavaExecuteScript.java : classe java exécute l'anti-virus pour détecter des virus de fichiers.

fr.gouv.vitam.ingest.external.common.model.response

- IngestExternalError.java : modèle de réponse d'erreur sur la request de dépôt ingest

ingest-external-api :

-IngestExternal.java : interface pour le service de dépôt externe. - IngestExternalOutcomeMessage.java : définir message de réponse du résultat de scan virus

ingest-external-core :

-IngestExternalImpl.java : implémenter des fonctionnalités de traitement sur le dépôt SIP , pré-défini dans - IngestExternal.java

ingest-external-rest :

- IngestExternalRessource.java : définir des ressources différentes pour le serveur REST ingest-external
- IngesteEternalApplication.java : créer & lancer le serveur d'application avec une configuration

ingest-external-client

- IngestExternalClient.java : interface client Ingestexternal
- IngestExternalexternalClientMock.java : mock client ingest-external
- IngestExternalClientRest.java : le client ingest-external et des fonctionnalités en se connectant au serveur REST ingest-external

1.5.4 ingest-internal-client

1.5.5 Utilisation

1.5.5.1 Paramètres

Les paramètres sont les InputStreams du fichier SIP pour le dépôt dans la base VITAM

1.5.5.2 La factory

Afin de récupérer le client, une factory a été mise en place.

```
// Récupération du client ingest-internal
IngestInternalClient client = IngestInternalClientFactory.getInstance().
↪getIngestInternalClient();
```

1.5.5.2.1 Le Mock

Par défaut, le client est en mode Mock. Il est possible de récupérer directement le mock :

```
// Changer la configuration du Factory
IngestInternalClientFactory.setConfiguration(IngestInternalClientFactory.
↪IngestInternalClientType.MOCK_CLIENT, null);
// Récupération explicite du client mock
IngestInternalClient client = IngestInternalClientFactory.getInstance().
↪getIngestInternalClient();
```

1.5.5.3 Le client

Pour instancier son client en mode Production :

```
// Ajouter un fichier functional-administration-client.conf dans /vitam/conf
    // Récupération explicite du client
IngestInternalClient client = IngestInternalClientFactory.getInstance().
↪getIngestInternalClient();
```

Le client propose trois méthodes :

```
         Status status();
         UploadResponseDTO upload(String archiveMimeType,List<LogbookParameters>␣
↪logbookParametersList, InputStream inputStream);
         // Télécharger un object du serveur sauvegardé de l'operation upload ci-
↪dessus avec son ID et type
         Response downloadObjectAsync(String objectId, IngestCollection type)



Cette méthde ( à la version 0.9.0) capable de télécharger un sip compressé en 3␣
↪formats (zip, tar, tar.gz)
```

- **Paramètres :**
  - archiveMimeType : : String (mimetype de l'archive ;par exemple application/x-tar)
  - logbookParametersList : : List<LogbookParameters>
  - inputStream : InputStream (stream de sip compressé dont le format doit être zip, tar ou tar.gz)
- Retourne : ATR en format xml
- Exceptions :

1.5.6 ingest-external-client

1.5.7 Utilisation

1.5.7.1 Paramètres

Les paramètres sont les InputStreams du fichier SIP pour le dépôt dans la base VITAM

1.5.7.2 La factory

Afin de récupérer le client, une factory a été mise en place.

```
// Récupération du client ingest-external
IngestExternalClient client = IngestExternalClientFactory.getInstance().
↪getIngestExternalClient();
```

1.5.7.2.1 Le Mock

Par défaut, le client est en mode Mock. Il est possible de récupérer directement le mock :

```
// Changer la configuration du Factory
IngestExternalClientFactory.setConfiguration(IngestExternalClientFactory.
↪IngestExternalClientType.MOCK_CLIENT, null);
// Récupération explicite du client mock
IngestExternalClient client = IngestExternalClientFactory.getInstance().
↪getIngestExternalClient();
```

1.5.7.3 Le client

Pour instancier son client en mode Production :

```
 // Ajouter un fichier functional-administration-client.conf dans /vitam/conf
     // Récupération explicite du client
IngestExternalClient client = IngestExternalClientFactory.getInstance().
↪getIngestExternalClient();
```

Le client les méthodes suivantes :

```
// Upload un SIP
RequestResponse<JsonNode> upload(InputStream stream, Integer tenantId, String␣
↪contextId, String action)
    throws IngestExternalException;

// Télécharger un object du serveur sauvegardé de l'operation upload ci-dessus avec␣
↪son ID et type
Response downloadObjectAsync(String objectId, IngestCollection type, Integer tenantId)
    throws IngestExternalException;

// Exécuter une action sur un Process Workflow
RequestResponse<JsonNode> executeOperationProcess(String operationId, String workflow,
↪ String contextId,
    String actionId, Integer tenantId)
    throws VitamClientException;

// Exécuter une action sur un Process Workflow
Response updateOperationActionProcess(String actionId, String operationId, Integer␣
↪tenantId) throws VitamClientException;

// Retourne le statut d'un process workflow et son état
ItemStatus getOperationProcessStatus(String id, Integer tenantId) throws␣
↪VitamClientException;

// Retourne le détail d'un process workflow
ItemStatus getOperationProcessExecutionDetails(String id, JsonNode query, Integer␣
↪tenantId) throws VitamClientException;

RequestResponse<JsonNode> cancelOperationProcessExecution(String id, Integer␣
↪tenantId) throws VitamClientException, BadRequestException;

@Deprecated //Not used
ItemStatus updateVitamProcess(String contextId, String actionId, String container,␣
↪String workflow,
    Integer tenantId)
```

```
    throws InternalServerException, BadRequestException, VitamClientException;

void initVitamProcess(String contextId, String container, String workFlow, Integer␣
↪tenantId)
    throws InternalServerException, VitamClientException;

@Deprecated
void initWorkFlow(String contextId, Integer tenantId) throws VitamException;

RequestResponse<JsonNode> listOperationsDetails(Integer tenantId) throws␣
↪VitamClientException;
```

Le client implémente aussi l'interface PoolingStatusClient : Avec cette interface, on peut utiliser les méthodes wait pour mieux gérer le pooling côté serveur et remédier à l'asynchrone des certains opérations.

```
// Possibilité de faire plusieurs (nbTry) appel espacé d'un temps (timeWait) avant de␣
↪répondre au client final
public boolean wait(int tenantId, String processId, ProcessState state, int nbTry,␣
↪long timeWait, TimeUnit timeUnit) throws VitamException;

public boolean wait(int tenantId, String processId, int nbTry, long timeWait,␣
↪TimeUnit timeUnit) throws VitamException;

public boolean wait(int tenantId, String processId, ProcessState state) throws␣
↪VitamException;

public boolean wait(int tenantId, String processId) throws VitamException;
```

1.5.8 ingest-external-antivirus

Dans cette section, nous expliquons comment utiliser et configurer le script d'antivirus pour le service ingest-external.

1. Configuration pour ingest-external : ingest-external.conf

   Dans ce fichier de configuration, nous précisons le nom de la script antivirus utilisé et le temps limité pour le scan. pour le moment, nous utilisons le script de scan scan-clamav.sh utilisant l'antivirus ClamAV.

```
antiVirusScriptName : scan-clamav.sh
timeoutScanDelay : 60000
```

2. Script d'antivirus scan-clamav.sh

   Le script permettant de lancer d'un scan d'un fichier envoyé avec l'antivirus ClamAV et retourner le résulat :

   0 : Analyse OK - no virus 1 : Virus trouvé et corrigé 2 : Virus trouvé mais non corrigé 3 : Analyse NOK

   Ce fichier est mis dans le répertoire vitam/conf avec le droit d'exécution.

3. Lancer le script en Java et intégration

JavaExecuteScript.java (se trouve dans ingest-external-common) permettant de lancer le script de clamav en Java en prenant des paramètres d'entrées : le script utilisé, le chemin du fichier à scanner et le temps limité d'un scan Pour l'intégration dans ingest-external, ce script est appelé dans l'implémentation des APIs de ingest-externe. la section suivant montre comment on appelle le script depuis ingest-external en Code.

```
antiVirusResult = JavaExecuteScript.executeCommand(antiVirusScriptName, filePath,␣
↪timeoutScanDelay);
.......
switch (antiVirusResult) {
    case 0:
        LOGGER.info(IngestExternalOutcomeMessage.OK_VIRUS.toString());
        endParameters.setStatus(LogbookOutcome.OK);
        endParameters.putParameterValue(LogbookParameterName.outcomeDetailMessage,
            IngestExternalOutcomeMessage.OK_VIRUS.value());
        break;
    case 1:
        LOGGER.debug(IngestExternalOutcomeMessage.OK_VIRUS.toString());
        endParameters.setStatus(LogbookOutcome.OK);
        endParameters.putParameterValue(LogbookParameterName.outcomeDetailMessage,
            IngestExternalOutcomeMessage.KO_VIRUS.value());
        break;
    case 2:
        LOGGER.error(IngestExternalOutcomeMessage.KO_VIRUS.toString());
        endParameters.setStatus(LogbookOutcome.KO);
        endParameters.putParameterValue(LogbookParameterName.outcomeDetailMessage,
            IngestExternalOutcomeMessage.KO_VIRUS.value());
        isFileInfected = true;
        break;
    default:
        LOGGER.error(IngestExternalOutcomeMessage.KO_VIRUS.toString());
        endParameters.setStatus(LogbookOutcome.FATAL);
        endParameters.putParameterValue(LogbookParameterName.outcomeDetailMessage,
            IngestExternalOutcomeMessage.KO_VIRUS.value());
        isFileInfected = true;
}
    ..................................................................................
↪...................
```

1.5.9 INGEST

1.5.9.1 L'application rest

1.5.9.1.1 ingest-internal : IngestInternalApplication

La méthode startApplication avec l'argument String[] permet aux tests unitaires de démarrer sur un port spécifique, le deuxième argument. Le premier argument contient le nom du fichier de configuration ingest-internal.conf (il est templetiser avec ansible).

1.5.9.1.2 ingest-external : IngestExternalApplication

même chose que pour IngestInternalApplication et avec ingest-external.conf à la place de ingest-internal.conf.

1.6 Logbook

1.6.1 Introduction

1.6.1.1 But de cette documentation

L'objectif de cette documentation est de compléter la Javadoc pour ce module.

1.6.2 Logbook

1.6.3 Utilisation

1.6.3.1 Paramètres

Les paramètres sont représentés via une interface **LogbookParameters** sous le package fr.gouv.vitam.logbook.common.parameters.

L'idée est de représenter les paramètres sous forme de Map<LogbookParameterName, String>.

Une methode getMapParameters() permet de récuperer l'ensemble de ces paramètres. Une methode getMandatoriesParameters() permet de récuperer un set de paramètre qui ne doivent pas être null ni vide.

On retrouve une implémentation dans la classe **LogbookOperationParameters** qui représente les paramètres pour journaliser une **opération**.

Il existe egalement une Enum **LogbookParameterName** qui permet de définir tous les noms de paramètres possible. Elle permet de remplir la map de paramètres ainsi que le set permettant de tester les paramètres requis.

1.6.3.2 La factory

Afin de récupérer le client ainsi que la bonne classe de paramètre, une factory a été mise en place. Actuellement, elle ne fonctionne que pour le journal des opérations.

```
// Récupération du client
LogbookOperationsClientFactory.changeMode(ClientConfiguration configuration)
LogbookOperationsClient client = LogbookOperationsClientFactory.getInstance().
↪getClient();
// Récupération de la classe paramètre
LogbookParameters parameters = LogbookParametersFactory.
↪newLogbookOperationParameters();
// Utilisation des setter : parameters.putParameterValue(parameterName,␣
↪parameterValue);
parameters.putParameterValue(LogbookParameterName.eventTypeProcess,
LogbookParameterName.eventTypeProcess.name())
.putParameterValue(LogbookParameterName.outcome,
LogbookParameterName.outcome.name());
// Usage recommandé : utiliser le factory avec les arguments obligatoires à remplir
LogbookParameters parameters = LogbookParametersFactory.
↪newLogbookOperationParameters(args);
// Des helpers pour aider
parameters.setStatus(LogbookOutcome).getStatus();
parameters.setTypeProcess(LogbookTypeProcess).getTypeProcess();
parameters.getEventDateTime();
parameters.setFromParameters(LogbookParameters).
↪getParameterValue(LogbookParameterName);
```

1.6.3.2.1 Le Mock

Par défaut, le client est en mode Mock. Il est possible de récupérer directement le mock :

```
// Changer la configuration du Factory
LogbookOperationsClientFactory.changeMode(null)
// Récupération explicite du client mock
LogbookOperationsClient client = LogbookOperationsClientFactory.getInstance().
↪getClient();
```

Pour instancier son client en mode Production :

```
// Changer la configuration du Factory
LogbookOperationsClientFactory.changeMode(ClientConfiguration configuration);
// Récupération explicite du client
LogbookOperationsClient client = LogbookOperationsClientFactory.getInstance().
↪getClient();
```

1.6.3.3 Le client

Le client propose actuellement quatre méthodes : create, update, selectOperation et selectOperationbyId

Le mock de create et upadate ne vérifie pas l'identifiant (eventIdentifier) ni la date (evendDateTime). En effet, il ne doit pas exister pour le create et inversement pour l'update.

Chacune de ces méthodes prend en arguement la classe paramètre instanciée via la factory et peuplée au besoin.

Le mock de selectOperation retourne un JsonNode qui contient MOCK_SELECT_RESULT_1 et MOCK_SELECT_RESULT_2

Le mock de selectOperationbyId retourne un JsonNode qui contient seulement MOCK_SELECT_RESULT_1. En effet, chaque opération a un identifiant unique.

Chacune de ces méthodes prend en arguement une requête select en String

```
// Récupération du client
LogbookOperationsClient client = LogbookOperationsClientFactory.getInstance().
↪getClient();

// Récupération de la classe paramètre
LogbookParameters parameters = LogbookParametersFactory.
↪newLogbookOperationParameters();
// Utilisation des setter : parameters.putParameterValue(parameterName,␣
↪parameterValue);

// create
client.create(parameters);
// possibilité de réutiliser le même parameters
// Utilisation des setter : parameters.putParameterValue(parameterName,␣
↪parameterValue);
// update
client.update(parameters);

// select opération
client.selectOperation(String select);
// select opération par id
client.selectOperationbyId(String select);
```

1.6.3.3.1 Exemple d'usage générique

```
// Récupération du client
LogbookOperationsClient client = LogbookOperationsClientFactory.getInstance().
↪getClient();

// Récupération de la classe paramètre
LogbookParameters parameters = LogbookParametersFactory.
↪newLogbookOperationParameters();
// Utilisation des setter : parameters.putParameterValue(parameterName,␣
↪parameterValue);
parameters.putParameterValue(LogbookParameterName.eventIdentifierProcess,
          GUIDFactory.newOperationId(tenant).getId())
        .setStatus(outcome).setTypeProcess(type);

// create global du processus AVANT toute opération sur ce processus
parameters.setStatus(LogbookOutcome.STARTED);
client.create(parameters);

// et maintenant append jusqu'à la fin du processus global
LogbookParameters subParameters = LogbookParametersFactory.
↪newLogbookOperationParameters();
// Récupère les valeurs du parent: attention à resetter les valeurs propres !
subParameters.setFromParameters(parameters);
// Event GUID
subParameters.putParameterValue(LogbookParameterName.eventIdentifier,
    GUIDFactory.newOperationIdGUID(tenantId).getId());
// Event Type
subParameters.putParameterValue(LogbookParameterName.eventType,
    "UNZIP");
subParameters.setStatus(LogbookOutcome.STARTED);
// Et autres paramètres
...
// Start sous opération
client.update(subParameters);
// Unsip
subParameters.setStatus(LogbookOutcome.OK);
// Sous opération OK
client.update(subParameters);

// Autres Opérations

// Fin Opération Globale
// create global du processus AVANT toute opération sur ce processus
parameters.setStatus(LogbookOutcome.OK);
client.update(parameters);

// Quand toutes les opérations sont terminées
client.close();
```

1.6.3.3.2 Exemple Ingest

```
// Available informations
// TenantId
int tenantId = 0;
// Process Id (SIP GUID)
```

```
String guidSip = "xxx";
// X-Request-Id
String xRequestId = "yyy";
// Global Object Id: in ingest = SIP GUID



// Récupération du client
LogbookOperationsClient client =
LogbookOperationsClientFactory.getInstance().getClient();



// Récupération de la classe paramètre avec ou sans argument
LogbookParameters parameters =
LogbookParametersFactory.newLogbookOperationParameters();
LogbookParameters parameters =
LogbookParametersFactory.newLogbookOperationParameters(eventIdentifier,
eventType, eventIdentifierProcess, eventTypeProcess,
outcome, outcomeDetailMessage, eventIdentifierRequest);


// Utilisation du setter
// Event GUID
parameters.putParameterValue(LogbookParameterName.eventIdentifier,
GUIDFactory.newOperationIdGUID(tenantId).getId());
// Event Type
parameters.putParameterValue(LogbookParameterName.eventType,
"UNZIP");
// Event Identifier Process
parameters.putParameterValue(LogbookParameterName.eventIdentifierProcess,
guidSip);
// Event Type Process
parameters.setTypeProcess(LogbookTypeProcess.INGEST);
// X-Request-Id
parameters.putParameterValue(LogbookParameterName.eventIdentifierRequest,
xRequestId);
// Global Object Id = SIP GUID for Ingest
parameters.putParameterValue(LogbookParameterName.objectIdentifier,
guidSip);



// Lancement de l'opération
// Outcome: status
parameters.setStatus(LogbookOutcome.STARTED);
// Outcome detail message
parameters.putParameterValue(LogbookParameterName.outcomeDetailMessage,
"One infotmation to set before starting the operation");



// 2 possibilities
// 1) Démarrage de l'Opération globale (eventIdentifierProcess) dans INGEST première␣
↪fois
client.create(parameters);
// 2) update global process Operation (same eventIdentifierProcess) partout ailleurs
client.update(parameters);
```

```
// Run Operation
runOperation();



// Finalisation de l'opération, selon le statut
// 1) Si OK
parameters.setStatus(LogbookOutcome.OK);
// 2) Si non OK
parameters.setStatus(LogbookOutcome.ERROR);
parameters.putParameterValue(LogbookParameterName.outcomeDetail,
"404_123456"); // 404 = code http, 123456 = code erreur Vitam



// Outcome detail message
parameters.putParameterValue(LogbookParameterName.outcomeDetailMessage,
"One infotmation to set after the operation");
// update global process operation
client.update(parameters);



// When all client opération is done
client.close();
```

1.6.3.3.3 Exemple ihm-demo-web-application

```
@POST
@Path("/logbook/operations")
@Produces(MediaType.APPLICATION_JSON)
public Response getLogbookResult(String options)

// Traduction de Mappeur à la requête DSL
Map<String, String> optionsMap = JsonHandler.getMapStringFromString(options);
query = CreateDSLClient.createSelectDSLQuery(optionsMap);

// Récupération du client
LogbookOperationsClient logbookClient = LogbookOperationsClientFactory.getInstance().
↪getLogbookOperationClient();

// Sélection des opérations par la requête DSL
result = logbookClient.selectOperation(query);

@POST
@Path("/logbook/operations/{idOperation}")
@Produces(MediaType.APPLICATION_JSON)
public Response getLogbookResultById(@PathParam("idOperation") String operationId,␣
↪String options)

// Récupération du client
LogbookClient logbookClient = LogbookClientFactory.getInstance().
↪getLogbookOperationClient();
```

```
// Sélection des opérations par ID
result = logbookClient.selectOperationbyId(operationId);
```

1.6.3.4 Données

La recherche des logbook de type TRACEABILITY passent par ElasticSearch, il faut faire attention à ce que le requête contenant des filtres de type "OrderBy" correspondent bien a des champs définit dans le mapping de l'index Elastic : *LogbookOperation.MAPPING*.

1.6.4 Logbook-lifecycle

1.6.5 Utilisation

1.6.5.1 Paramètres

Les paramètres sont représentés via une interface **LogbookParameters** sous le package fr.gouv.vitam.logbook.common.parameters.

L'idée est de représenter les paramètres sous forme de Map<LogbookParameterName, String>.

Une methode getMapParameters() permet de récuperer l'ensemble de ces paramètres. Une methode getMandatoriesParameters() permet de récuperer un set de paramètre qui ne doivent pas être null ni vide.

On retrouve une implémentation dans la classe **LogbookLifeCycleObjectGroupParameters** qui représente les paramètres pour journaliser un cycle de vie d'object group. **LogbookLifeCycleUnitParameters** qui représente les paramètres pour journaliser un cycle de vie d'archive unit.

Il existe egalement une Enum **LogbookParameterName** qui permet de définir tous les noms de paramètres possible. Elle permet de remplir la map de paramètres ainsi que le set permettant de tester les paramètres requis.

1.6.5.2 La factory

Afin de récupérer le client ainsi que la bonne classe de paramètre, une factory a été mise en place.

```
// Récupération du client
LogbookLifeCyclesClientFactory.changeMode(ClientConfiguration configuration)
LogbookLifeCycleClient client = LogbookLifeCyclesClientFactory.getInstance().
↪getClient();
// Récupération de la classe paramètre pour Object Group
LogbookParameters parameters = LogbookParametersFactory.
↪newLogbookLifeCycleObjectGroupParameters();
// Récupération de la classe paramètre pour Archive Unit
LogbookParameters parameters = LogbookParametersFactory.
↪newLogbookLifeCycleUnitParameters();
// Utilisation des setters pour Object Group et Archive Unit : parameters.
↪putParameterValue(parameterName, parameterValue);
parameters.putParameterValue(LogbookParameterName.agentIdentifier,
    SERVER_IDENTITY.getJsonIdentity());
parameters.putParameterValue(LogbookParameterName.eventDateTime,
    LocalDateUtil.now().toString());
// Usage recommandé : utiliser le factory avec les arguments obligatoires à remplir
      // Object Group
LogbookParameters parameters = LogbookParametersFactory.
↪newLogbookLifeCycleObjectGroupParameters(args);
```

```
// Archive Unit
LogbookParameters parameters = LogbookParametersFactory.
↪newLogbookLifeCycleUnitParameters(args);
// Des helpers pour aider
parameters.setStatus(LogbookOutcome).getStatus();
parameters.setTypeProcess(LogbookTypeProcess).getTypeProcess();
parameters.getEventDateTime();
parameters.setFromParameters(LogbookParameters).
↪getParameterValue(LogbookParameterName);
```

1.6.5.2.1 Le Mock

Par défaut, le client est en mode Mock. Il est possible de récupérer directement le mock :

```
// Changer la configuration du Factory
      LogbookLifeCyclesClientFactory.changeMode(null)
// Récupération explicite du client mock
LogbookClient client = LogbookLifeCyclesClientFactory.getInstance().getClient();
```

Pour instancier son client en mode Production :

```
// Changer la configuration du Factory
LogbookLifeCyclesClientFactory.changeMode(ClientConfiguration configuration)
// Récupération explicite du client
LogbookLifeCyclesClient client = LogbookLifeCyclesClientFactory.getInstance().
↪getClient();
```

1.6.5.3 Le client

Le client propose actuellement six méthodes : create, update, commit, rollback, selectOperation et selectLifeCycles et selectLifeCyclesById

// TODO

Cas d'usage provenant de processing.

1.7 Metadata

1.7.1 Métadata - Introduction

L'ensemble de ces documents est le manuel de développement du module Metadata, qui représente le métier fonctionnel de l'user story #70 de projet VITAM, dont le but et de réaliser des opérations sur la métatadata auprès de la base de données (insert/update/select/delete).

Le manuel se compose de deux parties - DAT présente l'architecture technique du module au niveau des packages, classes - REST-RAML explique comment on utitlise des différents service proprosés par module

1.7.2 DAT : module metadata

Ce document présente l'ensemble du manuel développement concernant le développment du module metadata qui identifié par la user story #70, qui contient :

- modules & packages
- classes métiers

---

1. Modules et packages

metadata

|— metadata-api : définir des APIs de traitement des requêtes un utilisant la base de données choisie |— metadata-core : implémentation des APIs |— metadata-rest : le serveur REST de métadata qui donnes des traitement sur les requêtes DSL |— metadata-client : client métadata qui sera utilisé par les autres modules pour faire des requête DSL sur le métadata

2. Classes métiers

Dans cette section, nous présentons quelques classes principales dans des modules/packages qu'on a abordé ci-dessus.

metadata-api :

-MetaData.java : définir des interface métiers pour le métadata

metadata-core :

-MetaDataImpl.java : implémenter des fonctionnalités de traitement sur le métadata, pré-défini dans -MetaData.java

metadata-rest

- MetaDataRessource.java : définir des ressources différentes pour le serveur REST métadata
- MetaDataApplication.java : créer & lancer le serveur d'application avec une configuration

metadata-client

- MetaDataClient.java : créer le client et des fonctionnalités en se connectant au serveur REST

1.7.3 Métadata

1.7.4 Utilisation

1.7.4.1 Paramètres

1.7.4.2 Le client

Le client propose actuellement différentes méthodes : insert et selection des units, select des objectGroups.

Il faut ajouter la dependance au niveau de pom.xml

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



    1. Créer le client métadata
    En deux étapes
    - chargement de la configuration en utilisant
            MetaDataClientFactory.changeMode(new ClientConfigurationImpl(server,␣
↪port));
```

```
            ou
            MetaDataClientFactory.changeMode(ConfigurationFilePath);)
    - création du client
    final MetaDataClient metadataClient = MetaDataClientFactory.getInstance().
↪getClient();

    2. Accéder aux fonctionnalités différents
    le client métadata fournit les foncitonnalités suivantes : insérer un ArchiveUnit,
    insérer un ObjectGroup et sélectionner un métadata (archiveUnit). Le détail de l
↪'utilisation
    de chaque fonctionnalité est ci-dessous.

    2.1 Insérer des ArchiveUnits
            try {
                    JsonNode result= metadataClient.insertUnit(JsonNode insertQuery)
            } catch (InvalidParseOperationException e) {
                LOG.error("parsing error", e);
                throw e;
            } catch (MetaDataExecutionException e) {
                LOG.error("execution error", e);
                throw e;
            } catch (MetaDataDocumentSizeException e) {
                LOG.error("document size input error", e);
                throw e;
            } catch (MetaDataAlreadyExistException e) {
                LOG.error("data already exists error", e);
                throw e;
            } catch (MetaDataNotFoundException e) {
                LOG.error("not found parent/path error", e);
                throw e;
            }

    Paramètre d'entrée est une requête DSL de type JsonNode, indiquant la requête sur␣
↪la collection Unit.
    Un exemple de la requête paramètrée est le suivant :

            {
              "$root" : [],
              "$queries": [{ "$path": "aaaaa" }],
              "$filter": { },
              "$data": { "_id": "value" }
            }

    Cette fonction retourne une réponse de type JsonNode contenant les informations :␣
↪code de retour en cas d'erreur,
    la requête effectuée sur la collection ...

    2.1 Insérer des ObjectGroups
    try {
                    JsonNode result= metadataClient.insertObjectGroup(JsonNode␣
↪insertQuery)
            } catch (InvalidParseOperationException e) {
                LOG.error("parsing error", e);
                throw e;
            } catch (MetaDataExecutionException e) {
                LOG.error("execution error", e);
                throw e;
            } catch (MetaDataDocumentSizeException e) {
```

```
                LOG.error("document size input error", e);
                throw e;
            } catch (MetaDataAlreadyExistException e) {
                LOG.error("data already exists error", e);
                throw e;
            } catch (MetaDataNotFoundException e) {
                LOG.error("not found parent/path error", e);
                throw e;
            }

    Paramètre d'entrée est une requête DSL de type JsonNode, indiquant la requête sur␣
↪la collection ObjectGroup.
    Un exemple de la requête paramètrée est le suivant :
```

```
{
    "$root" : [],
    "$queries": [{ "$exists": "value" }],
    "$filter": { },
    "$data": { "_id": "objectgroupValue" }
}

    Cette fonction retourne une réponse de type JsonNode contenant les informations :␣
↪code de retour en cas d'erreur,
    la requête effectuée sur la collection ...

    2.3 Sélection des ArchiveUnits
```

```
 try {
     // return JsonNode
     jsonNode = metaDataClient.selectUnits(
         accessModuleBean != null ? accessModuleBean.getRequestDsl() : "");
     } catch (InvalidParseOperationException e) {
     LOG.error("parsing error", e);
     throw e;
     } catch (MetadataInvalidSelectException e) {
     LOG.error("invalid select", e);
     throw e;
     } catch (MetaDataDocumentSizeException e) {
     LOG.error("document size problem", e);
     throw e;
     } catch (MetaDataExecutionException e) {
     LOG.error("metadata execution problem", e);
     throw e;
     } catch (IllegalArgumentException e) {
     LOG.error("illegal argument", e);
     throw new AccessExecutionException();
     } catch (Exception e) {
     LOG.error("exeption thrown", e);
     throw e;
 }

2.4 Sélection d'un ObjectGroup
```

```
try {
JsonNode selectQuery;
String objectGroupId;
// return JsonNode
```

```
jsonNode = metaDataClient.selectObjectGrouptbyId(selectQuery, objectGroupId);

} catch (InvalidParseOperationException e) {
LOG.error("parsing error", e);
throw e;
} catch (MetadataInvalidSelectException e) {
LOG.error("invalid select", e);
throw e;
} catch (MetaDataDocumentSizeException e) {
LOG.error("document size problem", e);
throw e;
} catch (MetaDataExecutionException e) {
LOG.error("metadata execution problem", e);
throw e;
} catch (IllegalArgumentException e) {
LOG.error("illegal argument", e);
throw new AccessExecutionException();
} catch (MetadataInvalidSelectException e) {
LOG.error("invalid selection", e);
throw new AccessExecutionException();
} catch (Exception e) {
LOG.error("exeption thrown", e);
throw e;
}
```

1.7.5 Métadata : API REST Raml

1.7.5.1 Présentation

Parent package : **fr.gouv.vitam.api**
Package proposition : **fr.gouv.vitam.metadata.rest**

Module pour le module opération : api / rest.

1.7.5.2 Rest API

URL Path : http://server/metadata/v1
POST /units -> **POST nouvelle unité d'archive récupération d'une liste des units avec une réquête **
GET /status -> **statut du server rest metadata (available/unavailable)**
POST /objectgroups -> **POST : insérer une nouveau groupe d'objects via une requête DSL**

1.7.6 Métadata-tenant

Les indices elasticsearch des unités archives et les groupes d'objets technique doivent être séparées par les tenants. Ces indices doivent être créées lors de démarrage du serveur grâce au fichier de configuration. Par exemple, pour metadata.conf on ajourte d'une ligne suivante :

- tenants : [0, 1, 2]

  indiqué que le serveur va travailler sur différents tenants 0, 1 et 2.

1. Valeur du tenant

La valeur de tenant est sauvegardé dans VitamSession et cette valeur sera récupérée par la fonction suivante.

```
VitamThreadUtils.getVitamSession().getTenantId()
```

Les indices sont créées basé sur les tenant pour chaque collection correspondante.

- Pour collection des unités archives, les indices sont : unit_0, unit_1, ... pour la liste de tenant 0, 1 ...
- Pour la collection des groupes d'objets technique, les indices sont objectgroups_0, objectgroups_1...

2. Refactor

Pour permettre de réaliser les opérations sur les collections de métadata via l'elastichsearch par le tenant, nous faisons un refactor sur les classe DbRequest et ElasticsearchAccessMetadata.

2.1. ElasticsearchAccessMetadata

Les fonctions d'ajout des indices pour la collection, mise à jour des indices ou delete des indices sont fait par le paramètre tenantId.

```
deleteIndex(final MetadataCollections collection, Integer tenantId)
addIndex(final MetadataCollections collection, Integer tenantId)
refreshIndex(final MetadataCollections collection, Integer tenantId)
addEntryIndexesBlocking(final MetadataCollections collection, final Integer tenantId,␣
↪final Map<String, String> mapIdJson)
addEntryIndex(final MetadataDocument<?> document, Integer tenantId)
...
```

2.2. DbRequest

- Le tenantId est récupéré dans la session par VitamThreadUtils.getVitamSession().getTenantId() pour appliquer au executeQuery() pour exécuter une requête.

```
    Result executeQuery(final RequestToAbstract requestToMongodb, final int rank,␣
↪final Result previous) {
    Integer tenantId = ParameterHelper.getTenantParameter();

    ...
}
```

1.7.7 Métadata

1.7.7.1 Utilisation

1.7.7.1.1 Paramètres

1.7.7.1.2 Calcul des règles de gestion pour une unité archivistique

1. Requête DSL

Pour calculer les règles héritées de l'archive Unit. Il faut ajouter "$rules : 1" dans le filtre de la requête DSL.

2. Calculer des règles de gestion pour une unité archivistique

Le serveur vérifie la requête, si son filtre contient "$rules : 1". On démarre la procédure de calcul des règles héritées

2.1 Rechercher les règles de gestion des parents et lui même

createSearchParentSelect(List<String> unitList)

2.1 Construire le graphe DAG avec tous les unité archivistique
ArrayNode unitParents = selectMetadataObject(newSelectQuery.getFinalSelect(), null, null) ;
Map<String, UnitSimplified> unitMap = UnitSimplified.getUnitIdMap(unitParents) ; UnitRuleCompute unitNode = new UnitRuleCompute(unitMap.get(unitId)) ; unitNode.buildAncestors(unitMap, allUnitNode, rootList) ;
2.3 Calculer des règles de gestion et mettre dans le résultat final
unitNode.computeRule() ; JsonNode rule = JsonHandler.toJsonNode(unitNode.getHeritedRules().getInheritedRule()) ;
((ObjectNode)arrayNodeResponse.get(0)).set(UnitInheritedRule.INHERITED_RULE, rule) ;

1.7.7.1.3 L'algorithme pour Calculer les règles de gestion

1. Initialiser les règles de gestion pour les units racines

```
Si (unit n'a pas le parent direct, s'est-à-dire, il est racine)
    Initialiser un objet UnitInheritedRule avec management et Id du unit
Autrement
    Créer un objet UnitInheritedRule vide
Fin Si
```

2. Ajouter les règles de gestion qui est hérité par les units parents.
   Cependant, il y a deux cas particuliers – la prévention d'héritage et l'exclusion d'héritage.

```
Pour (chaque parentId dans la liste de parent direct)
    Créer un objet UnitRuleCompute avec UnitSimplified qui contient management et Id␣
↪du unit et la liste de son parent
    Calculer le règle (Cette étape est récursive. Il va calculer les règles jusqu'à␣
↪la racine)
    Créer un objet UnitInheritedRule qui contient les règles hérités
    Concaténer les règles par défaut et les règles hérités
Fin Pour
```

2.1 La prévention d'héritage

L'intégration d'une balise <PreventInheritance> dans le SEDA Si le champ est « true », toutes les règles héritées des parents sont ignorées sur le nœud courant

2.2 L'exclusion d'héritage

L'intégration d'une balise <RefNonRuleId> dans le SEDA indiquant la règle à désactiver à partir de ce niveau.

1.8 Processing

1.8.1 Introduction

1.8.1.1 But de cette documentation

L'objectif de cette documentation est de compléter la Javadoc pour ce module.

1.8.2 Paramètres

Mise en place d'une classe de paramètres s'appuyant sur une map.

1.8.2.1 WorkerParamerterName, les noms de paramètre

Les noms de paramètres se trouvent dans l'énum WorkerParameterName. Pour ajouter un nouveau paramètre, ajouter son nom dans l'énum.

1.8.2.2 ParameterHelper, le helper

Utiliser le ParameterHelper afin de valider les éléments requis.

1.8.2.3 WorkerParametersFactory, la factory

Utiliser WorkerParametersFactory afin d'instancier une nouvelle classe de worker. Actuellement 5 paramètres sont obligatoires pour tous les workers : * urlMetadata afin d'intialiser le client metadata * urlWorkspace afin d'initialiser le client workspace * objectName le fichier json de l'object lorsque l'on boucle sur liste * currentStep le nom de l'étape * containerName l'identifiant du container

1.8.2.4 AbstractWorkerParameters, les implémentations par défaut

La classe abstraite AbstractWorkerParameters est l'implémentation par défaut de l'interface WorkerParameters. Si un paramètre est ajouté, il est possible de vouloir un getter et un setter particulier (aussi bien dans l'interface que dans l'implémentation abstraite).

1.8.2.5 DefaultWorkerParameters, l'implémentation actuelle

C'est l'implémentation actuelle des paramètres de worker.

1.8.3 Processing Management

Version 27/02/2017

1.8.3.1 Présentation

Parent package : **fr.gouv.vitam.processing**
Package proposition : **fr.gouv.vitam.processing.management**

4 modules composent la partie processing-management :

- processing-management : incluant la partie core + la partie REST.
- processing-management-client : incluant le client permettant d'appeler le REST.
- processing-engine : le moteur workflow.
- processing-data : le module de persistance et d'accès aux processus lancés (en éxécution, en pause, annulés, terminés).

1.8.3.1.1 Processing-management

1.8.3.1.1.1 Rest API

Dans le module Processing-management (package rest) : | http://server/processing/v1

GET /status -> **statut du logbook**

POST /operations/{id} -> **initialiser et/ou exécuter une action sur un processus workflow existant**
PUT /operations/{id} -> **exécuter une action sur processus existant**
- Relancer un processus en mode continu avec header X-ACTION==> resume
- Exécuter l'étape suivante avec header X-ACTION==> next
- Mettre en pause un processus avec header X-ACTION==> pause
GET /operations/{id} -> **récupérer les details d'un processus workflow par id et tenant**
HEAD /operations/{id} -> **récupérer l'état d'éxécution d'un processus workflow par id et tenant**
DELETE /operations/{id} -> **Annuler un processus**

De plus est ajoutée à la resource existante une resource déclarée dans le module processing-distributor (package rest).

http://server/processing/v1/worker_family
POST /{id_family}/workers/{id_worker} -> **POST Permet d'ajouter un worker à la liste des workers**
DELETE /{id_family}/workers/{id_worker} -> **DELETE Permet de supprimer un worker**

1.8.3.1.1.2 Core

Dans la partie Core, la classe ProcessManagementImpl propose les méthodes suivantes :

- **init** : Initialiser un processus avec un workflow donné. Dans cette étape on attach avec un cardinalité un-à-un un ProcessEngine et une machine à état à ce processus.
- **next** : Exécute l'action next (exécuter l'étape suivante mode step by step) sur un processus existant.
- **resume** : Exécute l'action resume (exécuter toutes les étapes mode continu) sur un processus existant.
- **pause** : Exécute l'action pause (mettre le processus en état pause dès que possible) sur un processus existant.
- **cancel** : Exécute l'action cancel (annuler un processus dès que possible) sur un processus existant.
- **findAllProcessWorkflow** : Lister tous les processus d'un tenant donné.
- **findOneProcessWorkflow** : Trouver un processus avec son id et son tenant.

1.8.3.1.1.3 La machine à état :

Dans la partie core on trouve aussi la classe StateMachine. Elle gère toutes les actions sur un processus donné.

- **next** :

Evaluer le passage de l'état actuel du processus vers l'état RUNNING en mode step by step. On ne peut passer à l'état RUNNING que depuis un état en cours PAUSE. Si cette évaluation ne lance pas d'exception alors lancer l'exéction d'un processus appel de ma la méthode **doRunning**.

- **resume** :

Evaluer le passage de l'état actuel du processus vers l'état RUNNING en mode continu. On ne peut passer à l'état RUNNING que depuis un état en cours PAUSE. Si cette évaluation ne lance pas d'exception alors lancer l'exéction d'un processus appel de ma la méthode **doRunning**.

- **pause** :

Evaluer le passage de l'état actuel du processus vers l'état PAUSE. On ne peut passer à l'état PAUSE que depuis un état en cours (PAUSE, RUNNING). Si cette évaluation ne lance pas d'exception alors , dans le cas d'un état en cours RUNNING, finir l'exécution de l'étape en cours et passer à l'état PAUSE, et si c'est la dernière étape, alors passer à l'état COMPLETED. Appel de ma la méthode **doPause**

- **cancel** :

Evaluer le passage de l'état actuel du processus vers l'état COMPLETED. On ne peut passer à l'état COMPLETED que depuis un état en cours (PAUSE, RUNNING). Si cette évaluation ne lance pas d'exception alors , dans le cas d'un état en cours RUNNING, finir l'exécution de l'étape en cours et passer à l'état COMPLETED. Appel de ma la méthode **doCompleted**

-**doRunning** : Appelée depuis **next** ou **resume**. -**doPause** : Appelée depusi **pause**. -**doCompleted** : Appelée depuis **cancel**.

-**onComplete** :

Appelée depuis le ProcessEngine quand une étape a été exécuté. Evaluation sur l'exécution de l'étape suivante selon les informations suivantes :

- Si la dernière étape alors exécuter finaliser le logbook et persister le processus.
- Sinon :
  - Vérifier si le status de l'étape est KO bloquant ou FATAL alors exécuter la dernière étape.
  - Sinon vérifier si une demande d'action est présente (évaluer la targetState) :
    - targetState = COMPLETED : Exécuter la dernière étape.
    - targetState = PAUSE : Alors pause
    - Sinon exécuter l'étape suivante.

-**onError** :

Appelée depuis le ProcessEngine quand une exception est levée lors de l'exécution d'un étape. Si c'est pas la dernière étape alors essayer d'exécuter la dernière étape. Dans tous les cas, finaliser le logbook et persister le processus.

-**onUpdate** :

Appelée depuis le ProcessEngine pour metter à jour les informations du processus à la volé.

Lors de la finalisation du logbook, la mise à jours des informations sur l'état et le status son effectué au niveau du processus. Une suppression de l'opération depuis le workspace.

1.8.3.1.1.4 Processing-management-client

1.8.3.1.1.5 Utilisation

Le client propose les méthode suivantes :

- **initVitamProcess** : initialiser le contexte d'un processus.
- **executeVitamProcess** : ! absolète !.
- **executeOperationProcess** :lancer un processus workflow avec un mode d'éxécution (resume/step by step).
- **updateOperationActionProcess** :relancer un processus workflow pour éxécuter une etape (mode : "next") ou toutes les étapes ("resume").
- **getOperationProcessStatus** :récupérer l'état d'éxécution d'un processus workflow par id et tenant.

- **cancelOperationProcessExecution** :annuler un processus workflow par id et tenant.
- **listOperationsDetails** :récupérer la liste des processus.
- **registerWorker** : permet d'ajouter un nouveau worker à la liste des workers.
- **unregisterWorker** : permet de supprimer un worker à la liste des workers.

1.8.3.1.1.6 Exemple :

```
processingClient = ProcessingManagementClientFactory.getInstance().getClient();
Response response = processingClient.executeOperationProcess("containerName",
↪"workflowId",
        logbookTypeProcess.toString(), ProcessAction.RESUME.getValue());
```

1.8.3.1.2 Processing-data

Le module Processing data est responsable de la partie persistance ,accès aux données des processus avoir l'état d'exécution et l'oordonnancement des étapes.

Le module processing data propose plusieurs méthodes :

- **initProcessWorkflow** : initialiser le contexte d'un processus.
- **updateStep** : mettre à jour une étape (les elements éxécutés/restés).
- **findOneProcessWorkflow** : Trouver depuis la map un processus par son id et son tenant.
- **findAllProcessWorkflow** : Trouver depuis la map tous les processus d'un tenant.
- **addToWorkflowList** : Ajouter un processus à la map (sauvegrade mémoire)

1.8.3.2 Configuration

1. Configuration du pom

Configuration du pom avec maven-surefire-plugin permet le build sous jenkins. Il permet de configurer le chemin des resources de esapi dans le common private.

1.8.4 Processing Distributor

1.8.4.1 Présentation

Parent package : **fr.gouv.vitam.processing**
Package proposition : **fr.gouv.vitam.processing.distributor**

2 modules composent la partie processing-distributor : - processing-distributor : incluant la partie core + la partie REST. - processing-distributor-client : incluant le client permettant d'appeler le REST.

1.8.4.1.1 Processing-distributor

1.8.4.2 Rest API

Pour l'instant les uri suivantes sont déclarées :

http://server/processing/v1/worker_family
POST /{id_family}/workers/{id_worker} -> **POST Permet d'ajouter un worker à la liste des workers**
DELETE /{id_family}/workers/{id_worker} -> **DELETE Permet de supprimer un worker**

A noter, que la resource ProcessDistributorResource est utilisée dans la partie Processing-Management.

1.8.4.3 Core

Dans la partie core la classe ProcessDistributorImpl propose une méthode principale : distribute. Cette méthode permet de lancer des étapes et de les diriger vers différents Workers (pour l'instant un seul worker existe). De plus, un système de monitoring permet d'enregistrer le statut des étapes lancées par la méthode distribute (cf module ProcessMonitoring). En attributs de l'implémentation du ProcessDistributor, sont présents 1 map de Workers ainsi qu'une liste de Workers disponibles. Ces 2 objets permettent (et permettront plus finement dans le futur) de gérer la liste des workers disponibles. Deux méthodes : registerWorker et unregisterWorker permettent d'ajouter ou de supprimer les workers à la liste des workers disponibles.

1.8.4.3.1 Processing-distributor-client

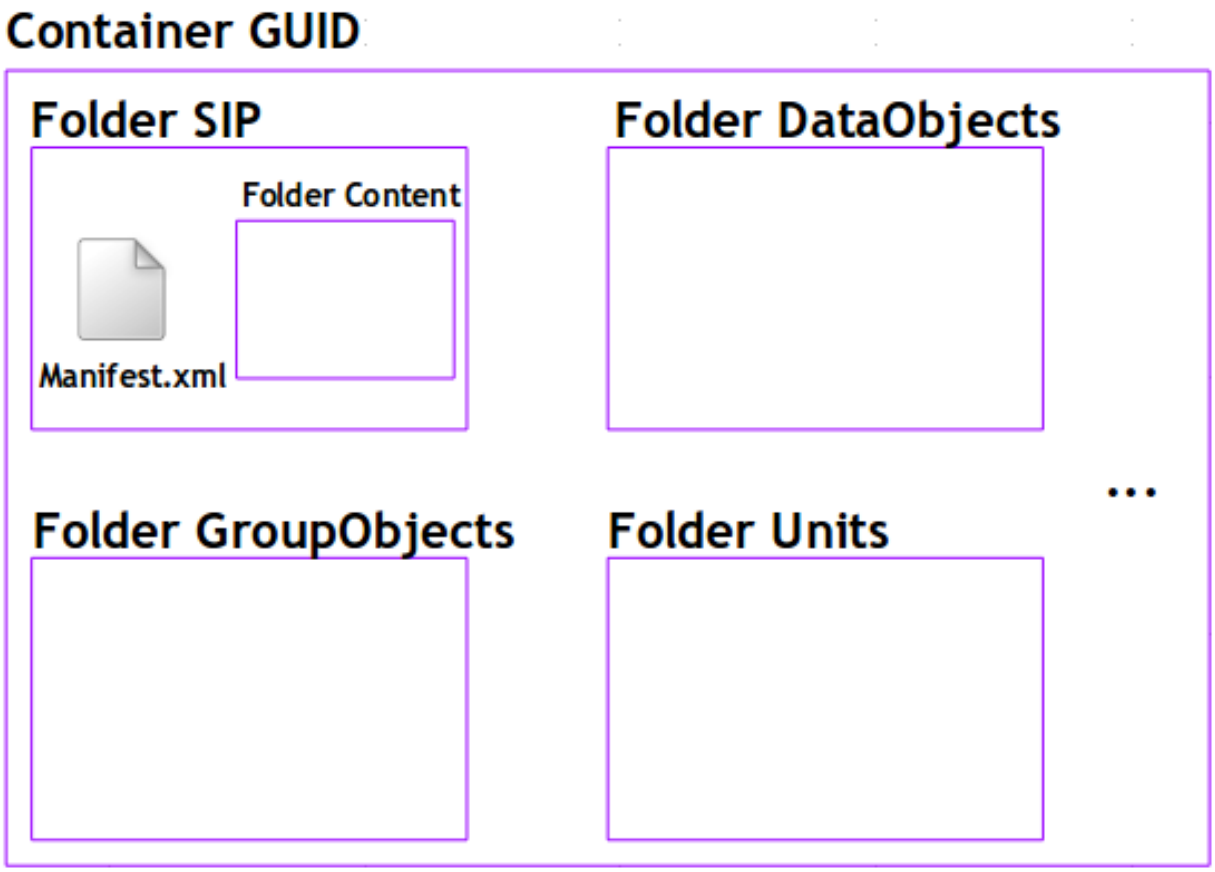
Pour le momentn le module est vide, car la partie client permettant d'appeler les méthodes register / unregister est portée par le module processing-management-client. A terme, il sera souhaité d'avoir 2 clients séparés.

1.8.5 Etudes en cours

1.8.5.1 Workspace

1.8.5.1.1 Arborescence

- ** example d'arborescence d'un container dans le workspace ** :



- ** détails ** : TODO

Pour chaque stream SIP

Container GUID

Folder GUID/SIP : stream SIP dézipé (manifest.xml et content)

Folder GUID/DataObjects : Physical/Binary DataObject

Folder GUID/ObjectGroups : hypothèse à ce stade un BinaryDataObject = un ObjectGroup

Folder GUID/Units : ArchiveUnit

1.8.5.2 Workflow

1.8.5.2.1 DefaultIngestWorkflow

Un Workflow est défini en JSON avec la structure suivante :

- un identifiant (id)
- une liste de Steps :

- un identifiant de famille de Workers (workerGroupId)
- un identifiant de Step (stepName)
- un modèle d'exécution (behavior) pouvant être : BLOCKING : le traitement est bloqué en cas d'erreur, il est nécessaire de recommencer le workflow NOBLOCKING : le traitement peut continuer malgrée les erreurs
- un modèle de distribution :

        - un type (kind) pouvant être REF ou LIST
        - l'élément de distribution (element) indiquant l'élément unique (REF) ou le chemin sur le Workspace (LIST)
    - une liste d'Actions :
        - un nom d'action (actionKey)
        - un modèle d'exécution (behavior) pouvant être BLOCKING ou NOBLOCKING
        - des paramètres d'entrées (in) :
            - un nom (name) utilisé pour référencer cet élément entre différents handlers d'une même étape
            - une cible (uri) comportant un schema (WORKSPACE, MEMORY, VALUE) et un path :
                - WORKSPACE :path indique le chemin relatif sur le workspace
                - MEMORY :path indique le nom de la clef de valeur
                - VALUE :path indique la valeur statique en entrée
            - chaque handler peut accéder à ces valeurs, définies dans l'ordre stricte, via le handlerIO
                - WORKSPACE : implicitement un File
                - MEMORY : implicitement un objet mémoire déjà alloué par un Handler précédent
                - VALUE : implicitement une valeur String
        - des paramètres de sortie (out) :
            - un nom (name) utilisé pour référencer cet élément entre différents handlers d'une même étape
            - une cible (uri) comportant un schema (WORKSPACE, MEMORY) et un path :
                - WORKSPACE :path indique le chemin relatif sur le workspace
                - MEMORY :path indique le nom de la clef de valeur
            - chaque handler peut stocker les valeurs finales, définies dans l'ordre stricte, via le handlerIO
                - WORKSPACE : implicitement un File local
                - MEMORY : implicitement un objet mémoire

Exemple :

```
1
2  ": "DefaultIngestWorkflow",
3  mment": "Default Ingest Workflow V6",
4  eps": [
5
6   "workerGroupId": "DefaultWorker",
7   "stepName": "STP_INGEST_CONTROL_SIP",
8   "behavior": "BLOCKING",
9   "distribution": {
10    "kind": "REF",
11    "element": "SIP/manifest.xml"
12  },
13  "actions": [
14    {
15      "action": {
16        "actionKey": "CHECK_SEDA",
17        "behavior": "BLOCKING"
18      }
19    },
20    {
21      "action": {
22        "actionKey": "CHECK_MANIFEST_DATAOBJECT_VERSION",
23        "behavior": "BLOCKING"
```

```
      }
    },
    {
      "action": {
        "actionKey": "CHECK_MANIFEST_OBJECTNUMBER",
        "behavior": "NOBLOCKING"
      }
    },
    {
      "action": {
        "actionKey": "CHECK_MANIFEST",
        "behavior": "BLOCKING",
        "out": [
          {
            "name": "unitsLevel.file",
            "uri": "WORKSPACE:UnitsLevel/ingestLevelStack.json"
          },
          {
            "name": "mapsDOtoOG.file",
            "uri": "WORKSPACE:Maps/DATA_OBJECT_TO_OBJECT_GROUP_ID_MAP.json"
          },
          {
            "name": "mapsDO.file",
            "uri": "WORKSPACE:Maps/DATA_OBJECT_ID_TO_GUID_MAP.json"
          },
          {
            "name": "mapsObjectGroup.file",
            "uri": "WORKSPACE:Maps/OBJECT_GROUP_ID_TO_GUID_MAP.json"
          },
          {
            "name": "mapsObjectGroup.file",
            "uri": "WORKSPACE:Maps/OG_TO_ARCHIVE_ID_MAP.json"
          },
          {
            "name": "mapsDOIdtoDODetail.file",
            "uri": "WORKSPACE:Maps/DATA_OBJECT_ID_TO_DATA_OBJECT_DETAIL_MAP.json"
          },
          {
            "name": "mapsUnits.file",
            "uri": "WORKSPACE:Maps/ARCHIVE_ID_TO_GUID_MAP.json"
          },
          {
            "name": "globalSEDAParameters.file",
            "uri": "WORKSPACE:ATR/globalSEDAParameters.json"
          }
        ]
      }
    },

      "action": {
        "actionKey": "CHECK_CONTRACT_INGEST",
        "behavior": "BLOCKING",
        "in": [
          {
            "name": "globalSEDAParameters.file",
            "uri": "WORKSPACE:ATR/globalSEDAParameters.json"
          }
        ]
```

```
      }
    }
    {
      "action": {
        "actionKey": "CHECK_CONSISTENCY",
        "behavior": "NOBLOCKING",
        "in": [
          {
            "name": "mapsDOtoOG.file",
            "uri": "WORKSPACE:Maps/OG_TO_ARCHIVE_ID_MAP.json"
          },
          {
            "name": "mapsDOtoOG.file",
            "uri": "WORKSPACE:Maps/OBJECT_GROUP_ID_TO_GUID_MAP.json"
          }
        ]
      }
    }
  ]
,

  "workerGroupId": "DefaultWorker",
  "stepName": "STP_OG_CHECK_AND_TRANSFORME",
  "behavior": "BLOCKING",
  "distribution": {
    "kind": "LIST",
    "element": "ObjectGroup"
  },
  "actions": [
     {
      "action": {
        "actionKey": "CHECK_DIGEST",
        "behavior": "BLOCKING",
        "in": [
          {
            "name": "algo",
            "uri": "VALUE:SHA-512"
          }
        ]
      }
    },
    {
      "action": {
        "actionKey": "OG_OBJECTS_FORMAT_CHECK",
        "behavior": "BLOCKING"
      }
    }
  ]
,

  "workerGroupId": "DefaultWorker",
  "stepName": "STP_UNIT_CHECK_AND_PROCESS",
  "behavior": "BLOCKING",
  "distribution": {
    "kind": "LIST",
    "element": "Units"
  },
  "actions": [
```

```
    {
       "action": {
          "actionKey": "CHECK_UNIT_SCHEMA",
          "behavior": "BLOCKING"
       }
    },
   {
     "action": {
       "actionKey": "UNITS_RULES_COMPUTE",
       "behavior": "BLOCKING"
     }
   }
 ]
,

 "workerGroupId": "DefaultWorker",
 "stepName": "STP_STORAGE_AVAILABILITY_CHECK",
 "behavior": "BLOCKING",
 "distribution": {
   "kind": "REF",
   "element": "SIP/manifest.xml"
 },
 "actions": [
   {
     "action": {
       "actionKey": "STORAGE_AVAILABILITY_CHECK",
       "behavior": "BLOCKING"
     }
   }
 ]
,

 "workerGroupId": "DefaultWorker",
 "stepName": "STP_OG_STORING",
 "behavior": "BLOCKING",
 "distribution": {
   "kind": "LIST",
   "element": "ObjectGroup"
 },
 "actions": [
   {
     "action": {
       "actionKey": "OG_STORAGE",
       "behavior": "BLOCKING"
     }
   },
   {
     "action": {
       "actionKey": "OG_METADATA_INDEXATION",
       "behavior": "BLOCKING"
     }
   }
 ]
,

 "workerGroupId": "DefaultWorker",
 "stepName": "STP_UNIT_STORING",
 "behavior": "BLOCKING",
```

```
 "distribution": {
   "kind": "LIST",
   "element": "Units"
 },
 "actions": [
   {
     "action": {
       "actionKey": "UNIT_METADATA_INDEXATION",
       "behavior": "BLOCKING"
     }
   }
 ]
,

 "workerGroupId": "DefaultWorker",
 "stepName": "STP_ACCESSION_REGISTRATION",
 "behavior": "BLOCKING",
 "distribution": {
   "kind": "REF",
   "element": "SIP/manifest.xml"
 },
 "actions": [
   {
     "action": {
       "actionKey": "ACCESSION_REGISTRATION",
       "behavior": "BLOCKING",
       "in": [
         {
           "name": "mapsUnits.file",
           "uri": "WORKSPACE:Maps/ARCHIVE_ID_TO_GUID_MAP.json"
         },
         {
           "name": "mapsDO.file",
           "uri": "WORKSPACE:Maps/OBJECT_GROUP_ID_TO_GUID_MAP.json"
         },
         {
           "name": "mapsDO.file",
           "uri": "WORKSPACE:Maps/BDO_TO_BDO_INFO_MAP.json"
         },
         {
           "name": "globalSEDAParameters.file",
           "uri": "WORKSPACE:ATR/globalSEDAParameters.json"
         }
       ]
     }
   }
 ]
,

 "workerGroupId": "DefaultWorker",
 "stepName": "STP_INGEST_FINALISATION",
 "behavior": "FINALLY",
 "distribution": {
   "kind": "REF",
   "element": "SIP/manifest.xml"
 },
 "actions": [
   {
```

```
    "action": {
      "actionKey": "ATR_NOTIFICATION",
      "behavior": "BLOCKING",
      "in": [
        {
          "name": "mapsUnits.file",
          "uri": "WORKSPACE:Maps/ARCHIVE_ID_TO_GUID_MAP.json",
          "optional": "true"
        },
        {
          "name": "mapsDO.file",
          "uri": "WORKSPACE:Maps/DATA_OBJECT_ID_TO_GUID_MAP.json",
          "optional": "true"
        },
        {
          "name": "mapsDOtoOG.file",
          "uri": "WORKSPACE:Maps/DATA_OBJECT_TO_OBJECT_GROUP_ID_MAP.json",
          "optional": "true"
        },
        {
          "name": "mapsDOIdtoDODetail.file",
          "uri": "WORKSPACE:Maps/DATA_OBJECT_ID_TO_DATA_OBJECT_DETAIL_MAP.json",
          "optional": "true"
        },
        {
          "name": "globalSEDAParameters.file",
          "uri": "WORKSPACE:ATR/globalSEDAParameters.json",
          "optional": "true"
        }
      ],
      "out": [
        {
          "name": "atr.file",
          "uri": "WORKSPACE:ATR/responseReply.xml"
        }
      ]
    }
  }
]
```

1.8.5.2.1.1 Etapes

- **Step 1** - STP_INGEST_CONTROL_SIP : Check SIP / distribution sur REF GUID/SIP/manifest.xml
  - CHECK_SEDA : - Test existence manifest.xml - Validation XSD SEDA manifest.xml
  - CHECK_MANIFEST_DATAOBJECT_VERSION :
  - CHECK_MANIFEST_OBJECTNUMBER : - Comptage BinaryDataObject dans manifest.xml en s'assurant d'aucun doublon : - List Workspace GUID/SIP/content/ - CheckObjectsNumber Comparaison des 2 nombres et des URI
  - CHECK_MANIFEST : - Extraction BinaryDataObject de manifest.xml / MAP des Id BDO / Génération GUID - Extraction ArchiveUnit de manifest.xml / MAP des id AU / Génération GUID - Contrôle des références dans les AU des Id BDO - Stockage dans Workspace des BDO et AU
  - CHECK_CONTRACT_INGEST : Vérification de la présence et contrôle du contrat d'entrée

    - CHECK_CONSISTENCY : vérification de la cohérence objet/unit
- **Step 2** - STP_OG_CHECK_AND_TRANSFORME : Check Objects Compliance du SIP / distribution sur LIST GUID/BinaryDataObject
    - CHECK_DIGEST : Contrôle de l'objet binaire correspondant du BDO taille et empreinte via Workspace
    - OG_OBJECTS_FORMAT_CHECK : - Contrôle du format des objets binaires - Consolidation de l'information du format dans l'object groupe correspondant si nécessaire
- **Step 3** - STP_UNIT_CHECK_AND_PROCESS : Check des archive unit et de leurs règles associées
    - CHECK_UNIT_SCHEMA : Contrôles intelligents du Json représentant l'Archive Unit par rapport à un schéma Json
    - UNITS_RULES_COMPUTE : Calcul des règles de gestion
- **Step 4** - STP_STORAGE_AVAILABILITY_CHECK : Check Storage Availability / distribution REF GUID/SIP/manifest.xml
    - STORAGE_AVAILABILITY_CHECK : Contrôle de la taille totale à stocker par rapport à la capacité des offres de stockage pour une stratégie et un tenant donnés
- **Step 5** - STP_OG_STORING : Rangement des objets
    - OG_STORAGE : Écriture des objets sur l'offre de stockage des BDO des GO
    - OG_METADATA_INDEXATION : Enregistrement en base des ObjectGroup
- **Step 6** - STP_UNIT_STORING : Index Units / distribution sur LIST GUID/Units
    - UNIT_METADATA_INDEXATION : - Transformation Json Unit et intégration GUID Unit + GUID GO - Enregistrement en base Units
- **Step 7** - STP_ACCESSION_REGISTRATION : Alimentation du registre de fond
    - ACCESSION_REGISTRATION : enregistrement des archives prises en charge dans le Registre des Fonds
- **Step 8 et finale** - STP_INGEST_FINALISATION : Notification de la fin de l'opération d'entrée. Cette étape est obligatoire et sera toujours exécutée, en dernière position.
    - ATR_NOTIFICATION : - génération de l'ArchiveTransferReply xml (OK ou KO) - enregistrement de l'ArchiveTransferReply xml dans les offres de stockage

1.8.5.2.1.2 Création d'un nouveau step

Un step est une étape de workflow. Il regroupe un ensemble d'actions (handler). Ces steps sont définis dans le workflowJSONvX.json (X=1,2).

1.9 Storage

1.9.1 Présentation

1.9.2 Storage Driver

Note : la récupération du bon driver associée à l'offre qui doit être utilisée est la responsabilité du DriverManager et ne sera pas décrit ici.

1.9.2.1 Utilisation d'un Driver

Comme expliqué dans la section architecture technique, le driver est responsable de l'établissement d'une connexion avec une ou plusieurs offres de stockage distantes. Le choix du driver à utiliser est la responsabilité du DriverManager qui fournit l'implémentation (si elle existe) du bon **Driver** en fonction de l'identifiant de l'offre de stockage.

1.9.2.1.1 Vérifier la disponibilité de l'offre

```
// Définition des paramètres nécessaires à l'établissement d'une connexion avec l
↪'offre de stockage
// Note: dans un vrai cas d'utilisation, ces paramètres doivent être récupérés de la␣
↪configuration de
// l'offre et ne pourrons pas être défini en dur de cette manière car l'utilisation␣
↪des drivers est un traitement
// générique à la fois vis à vis de l'offre et vis à vis du driver.
Properties parameters = new Properties();
parameters.put(StorageDriverParameterNames.USER.name(), "bob");
parameters.put(StorageDriverParameterNames.PASSWORD.name(), "p4ssword");

// 1Vérification de la disponibilité de l'offre
if (myDriver.isStorageOfferAvailable("http://my.storage.offer.com", parameters)) {
    // L'offre est disponible est accessible
} else {
    // L'offre est indisponible
}
```

1.9.2.1.2 Vérification de la capacité de l'offre

```
// Définition des paramètres nécessaires à l'établissement d'une connexion avec l
↪'offre de stockage
// Note: dans un vrai cas d'utilisation, ces paramètres doivent être récupérés de la␣
↪configuration de
// l'offre et ne pourrons pas être défini en dur de cette manière car l'utilisation␣
↪des drivers est un traitement
// générique à la fois vis à vis de l'offre et vis à vis du driver.
Properties parameters = new Properties();
parameters.put(StorageDriverParameterNames.USER.name(), "bob");
parameters.put(StorageDriverParameterNames.PASSWORD.name(), "p4ssword");

// Etablissement d'une connexion avec l'offre de stockage et réalisation d'une␣
↪opération
try (Connection myConnection = myDriver.connect("http://my.storage.offer.com",␣
↪parameters)) {
    // Le tenantId afin de récupérer la capacité
    Integer tenantId = 0;
    // Récupération de la capacité
    StorageCapacityResult capacity = myConnection.getStorageCapacity(tenantId);
    // On peut ici verifier que l'espace disponible est suffisant par exemple
} catch (StorageDriverException exc) {
    // Un problème est survenu lors de la communication avec le service distant
}
```

1.9.2.1.3 Compter les objets d'un conteneur de l'offre

```
// Définition des paramètres nécessaires à l'établissement d'une connexion avec l
↪'offre de stockage
// Note: dans un vrai cas d'utilisation, ces paramètres doivent être récupérés de la␣
↪configuration de
// l'offre et ne pourrons pas être défini en dur de cette manière car l'utilisation␣
↪des drivers est un traitement
```

```
// générique à la fois vis à vis de l'offre et vis à vis du driver.
Properties parameters = new Properties();
parameters.put(StorageDriverParameterNames.USER.name(), "bob");
parameters.put(StorageDriverParameterNames.PASSWORD.name(), "p4ssword");

Integer tenantId = 0;
String type = DataCategory.OBJECT.getFolder();
InputStream dataStream = new FileInputStream(PropertiesUtils.findFile("digitalObject.
↪pdf"));
// Etablissement d'une connexion avec l'offre de stockage et réalisation d'une␣
↪opération
try (Connection myConnection = myDriver.connect("http://my.storage.offer.com",␣
↪parameters)) {
    StorageRequest request = new StorageRequest(tenantId, type);
    StorageCountResult result = myConnection.countObjects(request);
    // On peut vérifier ici le résultat du count
} catch (StorageDriverException exc) {
    // Un problème est survenu lors de la communication avec le service distant
}
```

1.9.2.1.4 Put d'un objet dans l'offre de stockage

```
// Définition des paramètres nécessaires à l'établissement d'une connexion avec l
↪'offre de stockage
// Note: dans un vrai cas d'utilisation, ces paramètres doivent être récupérés de la␣
↪configuration de
// l'offre et ne pourrons pas être défini en dur de cette manière car l'utilisation␣
↪des drivers est un traitement
// générique à la fois vis à vis de l'offre et vis à vis du driver.
Properties parameters = new Properties();
parameters.put(StorageDriverParameterNames.USER.name(), "bob");
parameters.put(StorageDriverParameterNames.PASSWORD.name(), "p4ssword");

Integer tenantId = 0;
String type = DataCategory.OBJECT.getFolder();
String guid = "GUID";
String digestAlgorithm = DigestType.MD5.getName();
InputStream dataStream = new FileInputStream(PropertiesUtils.findFile("digitalObject.
↪pdf"));
// Etablissement d'une connexion avec l'offre de stockage et réalisation d'une␣
↪opération
try (Connection myConnection = myDriver.connect("http://my.storage.offer.com",␣
↪parameters)) {
    StoragePutRequest request = new StoragePutRequest(tenantId, type, guid,␣
↪digestAlgorithm, dataStream);
    StoragePutResult result = myConnection.putObject(request);
    // On peut vérifier ici le résultat du put
} catch (StorageDriverException exc) {
    // Un problème est survenu lors de la communication avec le service distant
}
```

1.9.2.1.5 Get d'un objet dans l'offre de stockage

```
// Définition des paramètres nécessaires à l'établissement d'une connexion avec l
↪'offre de stockage
// Note: dans un vrai cas d'utilisation, ces paramètres doivent être récupérés de la␣
↪configuration de
// l'offre et ne pourrons pas être défini en dur de cette manière car l'utilisation␣
↪des drivers est un traitement
// générique à la fois vis à vis de l'offre et vis à vis du driver.
Properties parameters = new Properties();
parameters.put(StorageDriverParameterNames.USER.name(), "bob");
parameters.put(StorageDriverParameterNames.PASSWORD.name(), "p4ssword");

Integer tenantId = 0;
String type = DataCategory.OBJECT.getFolder();
String guid = "GUID";
// Etablissement d'une connexion avec l'offre de stockage et réalisation d'une␣
↪opération
try (Connection myConnection = myDriver.connect("http://my.storage.offer.com",␣
↪parameters)) {
    StorageObjectRequest request = new StorageObjectRequest(tenantId, type, guid);
    StorageGetResult result = myConnection.getObject(request);
    // On peut vérifier ici le résultat du get
} catch (StorageDriverException exc) {
    // Un problème est survenu lors de la communication avec le service distant
}
```

1.9.2.1.6 Head d'un objet dans l'offre de stockage

```
// Définition des paramètres nécessaires à l'établissement d'une connexion avec l
↪'offre de stockage
// Note: dans un vrai cas d'utilisation, ces paramètres doivent être récupérés de la␣
↪configuration de
// l'offre et ne pourrons pas être défini en dur de cette manière car l'utilisation␣
↪des drivers est un traitement
// générique à la fois vis à vis de l'offre et vis à vis du driver.
Properties parameters = new Properties();
parameters.put(StorageDriverParameterNames.USER.name(), "bob");
parameters.put(StorageDriverParameterNames.PASSWORD.name(), "p4ssword");

Integer tenantId = 0;
String type = DataCategory.OBJECT.getFolder();
String guid = "GUID";
// Etablissement d'une connexion avec l'offre de stockage et réalisation d'une␣
↪opération
try (Connection myConnection = myDriver.connect("http://my.storage.offer.com",␣
↪parameters)) {
    StorageObjectRequest request = new StorageObjectRequest(tenantId, type, guid);
    Boolean result = myConnection.objectExistsInOffer(request);
    // On peut vérifier ici le résultat du head
} catch (StorageDriverException exc) {
    // Un problème est survenu lors de la communication avec le service distant
}
```

1.9.2.1.7 Delete d'un objet dans l'offre de stockage

```
// Définition des paramètres nécessaires à l'établissement d'une connexion avec l
↪'offre de stockage
// Note: dans un vrai cas d'utilisation, ces paramètres doivent être récupérés de la␣
↪configuration de
// l'offre et ne pourrons pas être défini en dur de cette manière car l'utilisation␣
↪des drivers est un traitement
// générique à la fois vis à vis de l'offre et vis à vis du driver.
Properties parameters = new Properties();
parameters.put(StorageDriverParameterNames.USER.name(), "bob");
parameters.put(StorageDriverParameterNames.PASSWORD.name(), "p4ssword");

Integer tenantId = 0;
String type = DataCategory.OBJECT.getFolder();
String guid = "GUID";
String digestAlgorithm = DigestType.MD5.getName();
final Digest digest = new Digest(algo);
InputStream dataStream = new FileInputStream(PropertiesUtils.findFile("digitalObject.
↪pdf"));
digest.update(dataStream);
// Etablissement d'une connexion avec l'offre de stockage et réalisation d'une␣
↪opération
try (Connection myConnection = myDriver.connect("http://my.storage.offer.com",␣
↪parameters)) {
    StorageRemoveRequest request = new StorageRemoveRequest(tenantId, type, guid,␣
↪digestType, digest.toString());
    StorageRemoveResult result = myConnection.removeObject(request);
    // On peut vérifier ici le résultat du delete
} catch (StorageDriverException exc) {
    // Un problème est survenu lors de la communication avec le service distant
}
```

1.9.2.1.8 Check d'un objet dans l'offre de stockage

```
// Définition des paramètres nécessaires à l'établissement d'une connexion avec l
↪'offre de stockage
// Note: dans un vrai cas d'utilisation, ces paramètres doivent être récupérés de la␣
↪configuration de
// l'offre et ne pourrons pas être défini en dur de cette manière car l'utilisation␣
↪des drivers est un traitement
// générique à la fois vis à vis de l'offre et vis à vis du driver.
Properties parameters = new Properties();
parameters.put(StorageDriverParameterNames.USER.name(), "bob");
parameters.put(StorageDriverParameterNames.PASSWORD.name(), "p4ssword");

Integer tenantId = 0;
String type = DataCategory.OBJECT.getFolder();
String guid = "GUID";
String digestAlgorithm = DigestType.MD5.getName();
final Digest digest = new Digest(algo);
InputStream dataStream = new FileInputStream(PropertiesUtils.findFile("digitalObject.
↪pdf"));
digest.update(dataStream);
// Etablissement d'une connexion avec l'offre de stockage et réalisation d'une␣
↪opération
```

```
try (Connection myConnection = myDriver.connect("http://my.storage.offer.com",␣
↪parameters)) {
    StorageCheckRequest request = new StorageCheckRequest(tenantId, type, guid,␣
↪digestType, digest.toString());
    StorageCheckResult result = myConnection.checkObject(request);
    // On peut vérifier ici le résultat du check
} catch (StorageDriverException exc) {
    // Un problème est survenu lors de la communication avec le service distant
}
```

1.9.2.1.9 Lister des types d'objets dans l'offre de stockage

```
// Définition des paramètres nécessaires à l'établissement d'une connexion avec l
↪'offre de stockage
// Note: dans un vrai cas d'utilisation, ces paramètres doivent être récupérés de la␣
↪configuration de
// l'offre et ne pourrons pas être défini en dur de cette manière car l'utilisation␣
↪des drivers est un traitement
// générique à la fois vis à vis de l'offre et vis à vis du driver.
Properties parameters = new Properties();
parameters.put(StorageDriverParameterNames.USER.name(), "bob");
parameters.put(StorageDriverParameterNames.PASSWORD.name(), "p4ssword");

Integer tenantId = 0;
String type = DataCategory.OBJECT.getFolder();
String guid = "GUID";
String digestAlgorithm = DigestType.MD5.getName();
final Digest digest = new Digest(algo);
InputStream dataStream = new FileInputStream(PropertiesUtils.findFile("digitalObject.
↪pdf"));
digest.update(dataStream);
// Etablissement d'une connexion avec l'offre de stockage et réalisation d'une␣
↪opération
try (Connection myConnection = myDriver.connect("http://my.storage.offer.com",␣
↪parameters)) {
    // Construction de l'objet permettant d'effectuer la requete. L'identifiant du␣
↪curseur n'existe pas et est à
    // null, c'est une demande de nouveau cusreur, x-cursor à vrai.
    StorageListRequest request = new StorageListRequest(tenantId, type, null, true);
    VitamRequestIterator<JsonNode> result = myConnection.listObjects(request);

    // On peut alors itérer sur le résultat
    while(result.hasNext()) {
        JsonNode json = result.next();
        // Traitement....
    }
} catch (StorageDriverException exc) {
    // Un problème est survenu lors de la communication avec le service distant
}
```

1.9.2.1.10 Récupérer les metadatas d'un objet

```
// Définition des paramètres nécessaires à l'établissement d'une connexion avec l
↪'offre de stockage
// Note: dans un vrai cas d'utilisation, ces paramètres doivent être récupérés de la␣
↪configuration de
// l'offre et ne pourrons pas être défini en dur de cette manière car l'utilisation␣
↪des drivers est un traitement
// générique à la fois vis à vis de l'offre et vis à vis du driver.
Properties parameters = new Properties();
parameters.put(StorageDriverParameterNames.USER.name(), "bob");
parameters.put(StorageDriverParameterNames.PASSWORD.name(), "p4ssword");

Integer tenantId = 0;
String type = DataCategory.OBJECT.getFolder();
String guid = "GUID";
String digestAlgorithm = DigestType.MD5.getName();
final Digest digest = new Digest(algo);
InputStream dataStream = new FileInputStream(PropertiesUtils.findFile("digitalObject.
↪pdf"));
digest.update(dataStream);
// Etablissement d'une connexion avec l'offre de stockage et réalisation d'une␣
↪opération
try (Connection myConnection = myDriver.connect("http://my.storage.offer.com",␣
↪parameters)) {
    // Construction de l'objet permettant d'effectuer la requete. L'identifiant du␣
↪curseur n'existe pas et est à
    // null, c'est une demande de nouveau cusreur, x-cursor à vrai.
    StorageListRequest request = new StorageListRequest(tenantId, type, null, true);
    VitamRequestIterator<JsonNode> result = myConnection.getMetadatas(request);

    // On peut alors itérer sur le résultat
    while(result.hasNext()) {
        JsonNode json = result.next();
        // Traitement....
    }
} catch (StorageDriverException exc) {
    // Un problème est survenu lors de la communication avec le service distant
}
```

1.9.3 Storage Engine

1.9.4 Storage Engine Client

1.9.4.1 La factory

Afin de récupérer le client une factory a été mise en place.

```
// Récupération du client
StorageClientFactory.changeMode(ClientConfiguration configuration)
StorageClient client = StorageClientFactory.getInstance().getClient();
```

A la demande l'instance courante du client, si un fichier de configuration storage-client.conf est présent dans le classpath le client en mode de production est envoyé, sinon il s'agit du mock.

1.9.4.1.1 Le Mock

En l'absence d'une configuration, le client est en mode Mock. Il est possible de récupérer directement le mock :

```
// Changer la configuration du Factory
StorageClientFactory.changeMode(null)
// Récupération explicite du client mock
StorageClient client = StorageClientFactory.getInstance().getClient();
```

1.9.4.1.2 Le mode de production

Pour instancier son client en mode Production :

```
// Changer la configuration du Factory
StorageClientFactory.setConfiguration(StorageConfiguration configuration);
// Récupération explicite du client
StorageClient client = StorageClientFactory.getInstance().getClient();
```

1.9.4.2 Les services

Le client propose actuellement des fonctionnalités nécéssitant toutes deux paramètres obligatoires :

- l'identifiant du tenant (valeur de test "0")
- l'identifiant de la stratégie de stockage (valeur de test "default")

Ces fonctionnalités sont :

- la récupération des informations sur une offre de stockage pour une stratégie (disponibilité et capacité) :

```
JsonNode result = client.getStorageInformation("0", "default");
```

- **l'envoi d'un objet sur une offre de stockage selon une stratégie donnée :**
  - pour les objets contenus dans le workspace (objets binaires) :

```
     StoredInfoResult result = storeFileFromWorkspace("0", "default",␣
↪StorageCollectionType.OBJECTS, "aeaaaaaaaaaam7mxaaaamakv3x3yehaaaaaq");


- pour les metadatas Json (objectGroup, unit, logbook -- pas encore implémenté côté␣
↪serveur) :
```

- **la vérification de l'existance d'un objet dans l'offre de stockage selon une stratégie donnée :**
  - pour les conteneurs (pas encore implémenté côté serveur) :

```
    boolean exist = existsContainer("0", "default");

- pour les autres objets (object, objectGroup, unit, logbook -- implémenté côté␣
↪serveur uniquement pour object) :
```

```
boolean exist = exists("0", "default", StorageCollectionType.OBJECTS,
↪"aeaaaaaaaaaam7mxaaaamakv3x3yehaaaaaq");
```

- la suppression d'un objet dans l'offre de stockage selon une stratégie donnée : - pour les conteneurs (pas encore implémenté côté serveur) :

```
boolean deleted = deleteContainer("0", "default");

 - pour les autres objets (object, objectGroup, unit, logbook -- implémenté côté␣
↪serveur uniquement pour object) :
```

```
boolean deleted = delete("0", "default", StorageCollectionType.OBJECTS,
↪"aeaaaaaaaaaam7mxaaaamakv3x3yehaaaaaq");
```

- la récupération d'un objet (InputStream) contenu dans un container :

```
Response response = client.getContainerAsync("0", "default",
↪"aeaaaaaaaaaam7mxaaaamakv3x3yehaaaaaq");
```

- La récupération de la liste d'objets d'un certain type :

```
// Si cursorId non connu
Response response = listContainerObjects("default", DataCategory.OBJECT, null)
// Si cursorId connu
Response response = listContainerObjects("default", DataCategory.OBJECT, "idcursor")
```

- La récupération du status est également disponible :

```
StatusMessage status = client.getStatus();
```

1.10 Technical administration

1.10.1 Introduction

1.11 Worker

1.11.1 Introduction

1.11.1.1 But de cette documentation

L'objectif de cette documentation est de compléter la Javadoc pour ce module.

1.11.2 Worker

1.11.2.1 1. Présentation

Parent package : **fr.gouv.vitam**
Package proposition : **fr.gouv.vitam.worker**

4 modules composent la partie worker : - worker-common : incluant la partie common (Utilitaires...), notamment le SedaUtils. - worker-core : contenant les différents handlers. - worker-client : incluant le client permettant d'appeler le REST. - worker-server : incluant la partie REST.

1.11.2.2 2. Worker-server

1.11.2.2.1 2.1 Rest API

Pour l'instant les uri suivantes sont déclarées :

http://server/worker/v1
POST /tasks -> **POST Permet de lancer une étape à exécuter**

1.11.2.2.2 2.2 Registration

Une partie registration permet de gérer la registration du Worker.

La gestion de l'abonnement du *worker* auprès du serveur *processing* se fait à l'aide d'un ServletContextListener : *fr.gouv.vitam.worker.server.registration.WorkerRegistrationListener*.

Le WorkerRegistrationListener va lancer l'enregistrement du *worker* au démarrage du serveur worker, dans un autre Thread utilisant l'instance *Runnable* : *fr.gouv.vitam.worker.server.registration.WorkerRegister*.

L'execution du *WorkerRegister* essaie d'enregistrer le *worker* suivant un retry paramétrable dans la configuration du serveur avec :

- un délai (registerDelay en secondes)
- un nombre d'essai (registerTry)

Le lancement du serveur est indépendant de l'enregistrement du *worker* auprès du *processing* : le serveur *worker* ne s'arrêtera pas si l'enregistrement n'a pas réussi.

2.3. Configuration de worker

Cela présente la configuration pour un worker quand il est déployé. Deux paramètres importants quand le worker fonctionne en mode parallèle.

- WorkerCapacity :

  Cela présente la capacité d'un worker qui réponds au demande de parallélisation de la distribution de tâches du workflow. Il est précisé par le paramètre capacity dans le WorkerConfiguration.

- WorkerFamily :

Chaque worker est configuré pour traiter groupe de tâches corresponsant à ses fonctions et on cela permetre de définir les familles de worker. Il est précisé par workerFamily dans le WorkerConfigration.

2.4. WorkerBean

présente l'information complète sur un worker pour la procédure d'enregistrement d'un worker. Il contient les information sur le nom, la famille et la capacité ... d'un worker et présente en mode json. Voici un example :

```
{ "name" : "workername", "family" : "DefaultWorker", "capacity" : 10, "storage" : 100,
"status" : "Active", "configuration" : {"serverHost" : "localhost", "serverPort" :␣
↪12345 } }
```

1.11.2.2.3 2.5. Persistence des workers

La lise de workers est persistée dans une base de données. Pour le moment, la base est un fichier de données qui contient une tableau de workers en format ArrayNode et chaque worker est une élément JsonNode. Exemple ci-dessous est des données d'une liste de workers

```
[
  {"workerId": "workerId1", "workerinfo": { "name" : "workername", "family" :
↪"DefaultWorker", "capacity" : 10, "storage" : 100,
  "status" : "Active", "configuration" : {"serverHost" : "localhost", "serverPort" :␣
↪12345 }}},
  {"workerId": "workerId2", "workerinfo": { "name" : "workername2", "family" :
↪"BigWorker", "capacity" : 10, "storage" : 100,
  "status" : "Active", "configuration" : {"serverHost" : "localhost", "serverPort" :␣
↪54321 } }}
]
```

Le fichier nommé "worker.db" qui sera créé dans le répertoire `/vitam/data/processing`.

Chaque worker est identifié par workerId et l'information générale du champs workerInfo. L'ensemble des actions suivantes sont traitées :

- Lors du redémarrage du distributor, il recharge la liste des workers enregistrés. Ensuite, il vérifie le status de chaque worker de la liste,

(serverPort :serverHost) en utilisant le WorkerClient. Si le worker qui n'est pas disponible, il sera supprimé de la liste des workers enregistrés et la base sera mise à jour.

- Lors de l'enregistrement/désenregistrement, la liste des workers enregistrés sera mis à jour (ajout/supression d'un worker).

```
checkStatusWorker(String serverHost, int serverPort) // vérifier le statut d'un worker
marshallToDB()   // mise à jour la base de la liste des workers enregistrés
```

1.11.2.2.4 2.6. Désenregistrement d'un worker

Lorsque le worker s'arrête ou se plante, ce worker doit être désenregistré.

- Si le worker s'arrête, la demande de désenregistrement sera lancé pour le contexte "contextDestroyed" de la WorkerRegistrationListener (implémenté de ServletContextListener) en utilisant le ProcessingManagementClient pour appeler le service de desenregistrement de distributeur.
- Si le worker se plante, il ne réponse plus aux requêtes de WorkerClient dans la "run()" WorkerThread et dans le catch() des exceptions de de traitement,

une demande de désenregistrement doit être appelé dans cette boucle.

- le distributeur essaie de faire une vérification de status de workers en appelant checkStatusWorker() en plusieurs fois définit dans GlobalDataRest.STATUS_CHECK_RETRY).
- si après l'étape 1 le statut de worker est toujours indisponible, le distributeur va appeler la procédure de désenregistrement de ce worker de la liste de worker enregistrés.

1.11.2.3 3. Worker-core

Dans la partie Core, sont présents les différents Handlers nécessaires pour exécuter les différentes actions.

- CheckConformityActionHandler
- CheckObjectsNumberActionHandler

- CheckObjectUnitConsistencyActionHandler
- CheckSedaActionHandler
- CheckStorageAvailabilityActionHandler
- CheckVersionActionHandler
- ExtractSedaActionHandler
- CheckIngestContractActionHandler
- IndexObjectGroupActionHandler
- IndexUnitActionHandler
- StoreObjectGroupActionHandler
- FormatIdentificationActionHandler
- AccessionRegisterActionHandler
- TransferNotificationActionHandler
- UnitsRulesCompteHandler
- DummyHandler

Plugins Worker : les plugins proposent des actions comme les Handler. Quand le service worker démarré, les plugins et leur fichier properties sont chargés. Les actions sont cherché d'abord dans le plugin pour le traitement, si l'action ne trouve pas dans plugin, il sera appelé dans le Handler correspondant.

- CheckConfirmityActionPlugin : pour la vérification de la conformité de document
- FormatIdentificationActionPlugin : pour le vérification de formats de fichiers
- StoreObjectGroupActionPlugin : pour le storage des groupes d'objets
- UnitsRulesComputeActionPlugin : pour la gestion de règles de gestion
- IndexUnitActionPlugin : pour indexer des unités archivistes
- IndexObjectGroupActionPlugin : pour indexer des groupes d'objets

La classe WorkerImpl permet de lancer ces différents handlers.

1.11.2.3.1 3.1 Focus sur la gestion des entrées / sorties des Handlers

Chaque Handler a un constructeur sans argument et est lancé avec la commande :

```
CompositeItemStatus execute(WorkerParameters params, HandlerIO ioParam).
..
```

Le HandlerIO a pour charge d'assurer la liaison avec le Workspace et la mémoire entre tous les handlers d'un step.

La structuration du HandlerIO est la suivante :

- des paramètres d'entrées (in) :
  - un nom (name) utilisé pour référencer cet élément entre différents handlers d'une même étape
  - une cible (uri) comportant un schema (WORKSPACE, MEMORY, VALUE) et un path :
    - WORKSPACE :path indique le chemin relatif sur le workspace
    - MEMORY :path indique le nom de la clef de valeur
    - VALUE :path indique la valeur statique en entrée
  - chaque handler peut accéder à ces valeurs, définies dans l'ordre stricte, via le handlerIO
    - WORKSPACE : implicitement un File

```
File file = handlerIO.getInput(rank);
..


    - MEMORY : implicitement un objet mémoire déjà alloué par un Handler précédent
```

```
// Object could be whatever, Map, List, JsonNode or even File
Object object = handlerIO.getInput(rank);
..

    - VALUE : implicitement une valeur String
```

```
String string = handlerIO.getInput(rank);
..
```

- des paramètres d'entrées (out) :
  - un nom (name) utilisé pour référencer cet élément entre différents handlers d'une même étape
  - une cible (uri) comportant un schema (WORKSPACE, MEMORY) et un path :
    - WORKSPACE :path indique le chemin relatif sur le workspace
    - MEMORY :path indique le nom de la clef de valeur
  - chaque handler peut stocker les valeurs finales, définies dans l'ordre stricte, via le handlerIO
    - WORKSPACE : implicitement un File local

```
// To get the filename as specified by the workflow
ProcessingUri uri = handlerIO.getOutput(rank);
String filename = uri.getPath();
// Write your own file
File newFile = handlerIO.getNewLocalFile(filename);
// write it
...
// Now give it back to handlerIO as ouput result,
// specifying if you want to delete it right after or not
handlerIO.addOuputResult(rank, newFile, true);
// or let the handlerIO delete it later on
handlerIO.addOuputResult(rank, newFile);
..

    - MEMORY : implicitement un objet mémoire
```

```
// Create your own Object
MyClass object = ...
// Now give it back to handlerIO as ouput result
handlerIO.addOuputResult(rank, object);
..
```

Afin de vérifier la cohérence entre ce qu'attend le Handler et ce que contient le HandlerIO, la méthode suivante est à réaliser :

```
List<Class<?>> clasz = new ArrayList<>();
// add in order the Class type of each Input argument
clasz.add(File.class);
clasz.add(String.class);
// Then check the conformity passing the number of output parameters too
```

```
boolean check = handlerIO.checkHandlerIO(outputNumber, clasz);
// According to the check boolean, continue or raise an error
..
```

1.11.2.3.2 3.2 Cas particulier des Tests unitaires

Afin d'avoir un handlerIO correctement initialisé, il faut redéfinir le handlerIO manuellement comme l'attend le handler :

```
// In a common part (@Before for instance)
HandlerIO handlerIO = new HandlerIO("containerName", "workerid");
List<IOParameter> out = new ArrayList<>();
out.add(new IOParameter().setUri(new ProcessingUri(UriPrefix.WORKSPACE, "UnitsLevel/
↪ingestLevelStack.json")));
out.add(new IOParameter().setUri(new ProcessingUri(UriPrefix.WORKSPACE, "Maps/DATA_
↪OBJECT_TO_OBJECT_GROUP_ID_MAP.json")));
out.add(new IOParameter().setUri(new ProcessingUri(UriPrefix.WORKSPACE, "Maps/DATA_
↪OBJECT_ID_TO_GUID_MAP.json")));
out.add(new IOParameter().setUri(new ProcessingUri(UriPrefix.WORKSPACE, "Maps/OBJECT_
↪GROUP_ID_TO_GUID_MAP.json")));
out.add(new IOParameter().setUri(new ProcessingUri(UriPrefix.WORKSPACE, "Maps/OG_TO_
↪ARCHIVE_ID_MAP.json")));
out.add(new IOParameter().setUri(new ProcessingUri(UriPrefix.WORKSPACE, "Maps/DATA_
↪OBJECT_ID_TO_DATA_OBJECT_DETAIL_MAP.json")));
out.add(new IOParameter().setUri(new ProcessingUri(UriPrefix.WORKSPACE, "Maps/ARCHIVE_
↪ID_TO_GUID_MAP.json")));
out.add(new IOParameter().setUri(new ProcessingUri(UriPrefix.WORKSPACE, "ATR/
↪globalSEDAParameters.json")));
// Dans un bloc @After, afin de nettoyer les dossiers
@After
public void aftertest() {
  handlerIO.close();
}
// Pour chaque test
@Test
public void test() {
  handlerIO.addOutIOParameters(out);
  ...
}
```

Si nécessaire et si compatible, il est possible de passer par un mode MEMORY pour les paramètres "in" :

```
// In a common part (@Before for instance)
HandlerIO handlerIO = new HandlerIO("containerName", "workerid");
// Declare the signature in but instead of using WORKSPACE, use MEMORY
List<IOParameter> in = new ArrayList<>();
in.add(new IOParameter().setUri(new ProcessingUri(UriPrefix.MEMORY, "file1")));
in.add(new IOParameter().setUri(new ProcessingUri(UriPrefix.MEMORY, "file2")));
in.add(new IOParameter().setUri(new ProcessingUri(UriPrefix.MEMORY, "file3")));
in.add(new IOParameter().setUri(new ProcessingUri(UriPrefix.MEMORY, "file4")));
// Dans un bloc @After, afin de nettoyer les dossiers
@After
public void aftertest() {
handlerIO.close();
}
// Pour chaque test
```

```
@Test
public void test() {
// Use it first as Out parameters
handlerIO.addOutIOParameters(in);
// Initialize the real value in MEMORY using those out parameters from Resource Files
handlerIO.addOuputResult(0, PropertiesUtils.getResourceFile(ARCHIVE_ID_TO_GUID_MAP));
handlerIO.addOuputResult(1, PropertiesUtils.getResourceFile(OBJECT_GROUP_ID_TO_GUID_
↪MAP));
handlerIO.addOuputResult(2, PropertiesUtils.getResourceFile(DO_TO_DO_INFO_MAP));
handlerIO.addOuputResult(3, PropertiesUtils.getResourceFile(ATR_GLOBAL_SEDA_
↪PARAMETERS));
// Reset the handlerIo in order to remove all In and Out parameters
handlerIO.reset();
// And now declares the In parameter list, that will use the MEMORY default values
handlerIO.addInIOParameters(in);
...
}
// If necessary, delcares real OUT parameters too there
List<IOParameter> out = new ArrayList<>();
out.add(new IOParameter().setUri(new ProcessingUri(UriPrefix.WORKSPACE, "file5")));
handlerIO.addOutIOParameters(out);
// Now handler will have access to in parameter as File as if they were coming from␣
↪Workspace
```

1.11.2.3.3 3.3 Création d'un nouveau handler

La création d'un nouveaux handler doit être motivée par certaines conditions nécessaires :

- lorsque qu'il n'y a pas de handler qui répond au besoin
- lorsque rajouter la fonctionnalité dans un handler existant, le surcharge et le détourne de sa fonctionalité première
- lorsque l'on veut refactorer un handler existant pour donner des fonctionalités 'un peu' plus 'élémentaires'

Les handlers doivent étendrent la classe ActionHandler et implémenter la méthode execute. Lors de la création d'un nouveau handler, il faut ajouter une nouvelle instance, dans WorkerImpl.init pour enregistrer le handler dans le worker et définir le handler id. Celui-ci sert de clé pour :

- les messages dans logbook (vitam-logbook-messages_fr.properties) en fonction de la criticité
- les fichiers json de définition des workflows json (exemple : DefaultIngestWorkflow.json)

cf. workflow

1.11.2.4 4. Details des Handlers

1.11.2.4.1 4.1 Détail du handler : CheckConformityActionHandler

1.11.2.4.1.1 4.1.1 description

Ce handler permet de contrôle de l'empreinte. Il comprend désormais 2 tâches :

– Vérification de l'empreinte par rapport à l'empreinte indiquée dans le manifeste (en utilisant algorithme déclaré dans manifeste) – Calcul d'une empreinte en SHA-512 si l'empreinte du manifeste est calculée avec un algorithme différent

1.11.2.4.1.2 4.1.2 exécution

CheckConformityActionHandler recupère l'algorithme de Vitam (SHA-512) par l'input dans workflow et le fichier en InputStream par le workspace.

Si l'algorithme est différent que celui dans le manifest, il calcul l'empreinte de fichier en SHA-512

```
      DigestType digestTypeInput = DigestType.fromValue((String) handlerIO.getInput().
↪get(ALGO_RANK));
response = handlerIO.getInputStreamNoCachedFromWorkspace(
IngestWorkflowConstants.SEDA_FOLDER + "/" + binaryObject.getUri());
InputStream inputStream = (InputStream) response.getEntity();
final Digest vitamDigest = new Digest(digestTypeInput);
Digest manifestDigest;
boolean isVitamDigest = false;
if (!binaryObject.getAlgo().equals(digestTypeInput)) {
    manifestDigest = new Digest(binaryObject.getAlgo());
    inputStream = manifestDigest.getDigestInputStream(inputStream);
} else {
    manifestDigest = vitamDigest;
    isVitamDigest = true;
}
.....................
```

Si les empreintes sont différents, c'est le cas KO. Le message { "MessageDigest" : "value", "Algorithm" : "algo", "ComputedMessageDigest" : "value"} va être stocké dans le journal Sinon le message { "MessageDigest" : "value", "Algorithm" : "algo", "SystemMessageDigest" : "value", "SystemAlgorithm" : "algo"} va être stocké dans le journal Mais il y a encore deux cas à ce moment :

> si l'empreinte est avec l'algorithme SHA-512, c'est le cas OK. sinon, c'est le cas WARNING. le nouveau empreint et son algorithme seront mis à jour dans la collection ObjectGroup.

CheckConformityActionHandler compte aussi le nombre de OK, KO et WARNING. Si nombre de KO est plus de 0, l'action est KO.

1.11.2.4.1.3 4.1.3 journalisation :

1.11.2.4.1.4 logbook lifecycle

CA 1 : Vérification de la conformité de l'empreinte. (empreinte en SHA-512 dans le manifeste)

Dans le processus d'entrée, l'étape de vérification de la conformité de l'empreinte doit être appelée en position 450. Lorsque l'étape débute, pour chaque objet du groupe d'objet technique, une vérification d'empreinte doit être effectuée (celle de l'objet avec celle inscrite dans le manifeste SEDA). Cette étape est déjà existante actuellement. Le calcul d'empreinte en SHA-512 (CA 2) ne doit pas s'effectuer si l'empreinte renseigné dans le manifeste a été calculé en SHA-512. C'est cette empreinte qui sera indexée dans les bases Vitam.

CA 1.1 : Vérification de la conformité de l'empreinte. (empreinte en SHA-512 dans le manifeste) - Started

- Lorsque l'action débute, elle inscrit une ligne dans les journaux du cycle de vie des GOT :

- eventType EN – FR : « Digest Check», « Vérification de l'empreinte des objets»
- outcome : "Started"
- outcomeDetailMessage FR : « Début de la vérification de l'empreinte »
- eventDetailData FR : "Empreinte manifeste : <MessageDigest>, algorithme : <MessageDigest attribut algorithm>"
- objectIdentifierIncome : MessageIdentifier du manifest

CA 1.2 : Vérification de la conformité de l'empreinte. (empreinte en SHA-512 dans le manifeste) - OK

- Lorsque l'action est OK, elle inscrit une ligne dans les journaux du cycle de vie des GOT :

- eventType EN – FR : « Digest Check», « Vérification de l'empreinte des objets»
- outcome : "OK"
- outcomeDetailMessage FR : « Succès de la vérification de l'empreinte »
- eventDetailData FR : "Empreinte : <MessageDigest>, algorithme : <MessageDigest attribut algorithm>"
- objectIdentifierIncome : MessageIdentifier du manifest

Comportement du workflow décrit dans l'US #680

- La collection ObjectGroup est aussi mis à jour, en particulier le champs : Message Digest : { empreinte, algorithme utlisé }

CA 1.3 : Vérification de la conformité de l'empreinte. (empreinte en SHA-512 dans le manifeste) - KO

- Lorsque l'action est KO, elle inscrit une ligne dans les journaux du cycle de vie des GOT :

- eventType EN – FR : « Digest Check», « Vérification de l'empreinte des objets»
- outcome : "KO"
- outcomeDetailMessage FR : « Échec de la vérification de l'empreinte »
- eventDetailData FR : "Empreinte manifeste : <MessageDigest>, algorithme : <MessageDigest attribut algorithm> Empreinte calculée : <Empreinte calculée par Vitam>"
- objectIdentifierIncome : MessageIdentifier du manifest

Comportement du workflow décrit dans l'US #680

CA 2 : Vérification de la conformité de l'empreinte. (empreinte différent de SHA-512 dans le manifeste)

Si l'empreinte proposé dans le manifeste SEDA n'est pas en SHA-512, alors le système doit calculer l'empreinte en SHA-512. C'est cette empreinte qui sera indexée dans les bases Vitam. Lorsque l'action débute, pour chaque objet du groupe d'objet technique, un calcul d'empreinte au format SHA-512 doit être effectué. Cette action intervient juste apres le check de l'empreinte dans le manifeste (mais on est toujours dans l'étape du check conformité de l'empreinte).

CA 2.1 : Vérification de la conformité de l'empreinte. (empreinte différent de SHA-512 dans le manifeste) - Started

- Lorsque l'action débute, elle inscrit une ligne dans les journaux du cycle de vie des GOT :

- eventType EN – FR : « Digest Check», « Vérification de l'empreinte des objets»
- outcome : "Started"
- outcomeDetailMessage FR : « Début de la vérification de l'empreinte »
- eventDetailData FR : "Empreinte manifeste : <MessageDigest>, algorithme : <MessageDigest attribut algorithm>"
- objectIdentifierIncome : MessageIdentifier du manifest

CA 2.2 : Vérification de la conformité de l'empreinte. (empreinte différent de SHA-512 dans le manifeste) - OK

- Lorsque l'action est OK, elle inscrit une ligne dans les journaux du cycle de vie des GOT :

- eventType EN – FR : « Digest Check», « Vérification de l'empreinte des objets»
- outcome : "OK"
- outcomeDetailMessage FR : « Succès de la vérification de l'empreinte »
- eventDetailData FR : "Empreinte Manifeste : <MessageDigest>, algorithme : <MessageDigest attribut algorithm>" "Empreinte calculée (<algorithme utilisé "XXX">) : <Empreinte calculée par Vitam>"
- objectIdentifierIncome : MessageIdentifier du manifest

1.11.2.4.1.5 4.1.5 modules utilisés

processing, worker, workspace et logbook

1.11.2.4.1.6 4.1.4 cas d'erreur

XMLStreamException : problème de lecture SEDA InvalidParseOperationException : problème de parsing du SEDA LogbookClientAlreadyExistsException : un logbook client existe dans ce workflow LogbookClientBadRequestException : LogbookLifeCycleObjectGroupParameters est mal paramétré et le logbook client génère une mauvaise requete LogbookClientException : Erreur générique de logbook. LogbookException classe mère des autres exceptions LogbookClient LogbookClientNotFoundException : un logbook client n'existe pas pour ce workflow LogbookClientServerException : logbook server a un internal error ProcessingException : erreur générique du processing ContentAddressableStorageException : erreur de stockage

1.11.2.4.2 4.2 Détail du handler : CheckObjectsNumberActionHandler

1.11.2.4.2.1 4.2.1 description

Ce handler permet de comparer le nombre d'objet stocké sur le workspace et le nombre d'objets déclaré dans le manifest.

1.11.2.4.3 4.3 Détail du handler : CheckObjectUnitConsistencyActionHandler

Ce handler permet de contrôler la cohérence entre l'object/object group et l'ArchiveUnit.

Pour ce but, on détecte les groupes d'object qui ne sont pas référé par au moins d'un ArchiveUnit. Ce tache prend deux maps de données qui ont été crée dans l'étape précédente de workflow comme input : objectGroupIdToUnitId objectGroupIdToGuid Le ouput de cette contrôle est une liste de groupe d'objects invalide. Si on trouve les groupe d'objects invalide, le logbook lifecycles de group d'object sera mis à jour.

L'exécution de l'algorithme est présenté dans le code suivant :*

```
while (it.hasNext()) {
  final Map.Entry<String, Object> objectGroup = it.next();
  if (!objectGroupToUnitStoredMap.containsKey(objectGroup.getKey())) {
    itemStatus.increment(StatusCode.KO);
    try {
      // Update logbook OG lifecycle
      final LogbookLifeCycleObjectGroupParameters␣
↪logbookLifecycleObjectGroupParameters =
          LogbookParametersFactory.newLogbookLifeCycleObjectGroupParameters();
      LogbookLifecycleWorkerHelper.updateLifeCycleStartStep(handlerIO.getHelper(),
          logbookLifecycleObjectGroupParameters,
          params, HANDLER_ID, LogbookTypeProcess.INGEST,
          objectGroupToGuidStoredMap.get(objectGroup.getKey()).toString());
      logbookLifecycleObjectGroupParameters.setFinalStatus(HANDLER_ID, null,␣
↪StatusCode.KO,
          null);
      handlerIO.getHelper().updateDelegate(logbookLifecycleObjectGroupParameters);
      final String objectID =
          logbookLifecycleObjectGroupParameters.
↪getParameterValue(LogbookParameterName.objectIdentifier);
      handlerIO.getLifecyclesClient().bulkUpdateObjectGroup(params.getContainerName(),
```

```
            handlerIO.getHelper().removeUpdateDelegate(objectID));
        } catch (LogbookClientBadRequestException | LogbookClientNotFoundException |
          LogbookClientServerException | ProcessingException e) {
          LOGGER.error("Can not update logbook lifcycle", e);
        }
        ogList.add(objectGroup.getKey());
      } else {
        itemStatus.increment(StatusCode.OK);
        // Update logbook OG lifecycle
        ....
      }
    }
```

1.11.2.4.4 4.4 Détail du handler : CheckSedaActionHandler

Ce handler permet de valider la validité du manifest par rapport à un schéma XSD. Il permet aussi de vérifier que les informations remplies dans ce manifest sont correctes.

- Le schéma de validation du manifest : src/main/resources/seda-vitam-2.0-main.xsd.

1.11.2.4.5 4.4 Détail du handler : CheckStorageAvailabilityActionHandler

TODO

1.11.2.4.6 4.5 Détail du handler : CheckVersionActionHandler

TODO

1.11.2.4.7 4.6 Détail du handler : ExtractSedaActionHandler

1.11.2.4.7.1 4.6.1 description

Ce handler permet d'extraire le contenu du SEDA. Il y a :

- extraction des BinaryDataObject et PhysicalDataObject
- extraction des ArchiveUnit
- création des lifes cycles des units
- construction de l'arbre des units et sauvegarde sur le workspace
- sauvegarde de la map des units sur le workspace
- sauvegarde de la map des objets sur le workspace
- sauvegarde de la map des objets groupes sur le workspace

1.11.2.4.7.2 4.6.2 Détail des différentes maps utilisées :

Map<String, String> dataObjectIdToGuid

> contenu : cette map contient l'id du DO relié à son guid création : elle est créé lors de la création du handler MAJ, put : elle est populée lors de la lecture des BinaryDataObject et PhysicalDataObject lecture, get : saveObjectGroupsToWorkspace, getObjectGroupQualifiers, suppression : c'est un clean en fin d'execution du handler

Map<String, String> dataObjectIdToObjectGroupId :

contenu : cette map contient l'id du DO relié au groupe d'objet de la balise DataObjectGroupId ou DataObjectGroupReferenceId création : elle est créé lors de la création du handler MAJ, put : elle est populée lors de la lecture des BinaryDataObject et PhysicalDataObject lecture, get : lecture de la map dans mapNewTechnicalDataObjectGroupToDO, getNewGdoIdFromGdoByUnit, completeDataObjectToObjectGroupMap, checkArchiveUnitIdReference et writeDataObjectInLocal suppression : c'est un clean en fin d'execution du handler

Map<String, GotObj> dataObjectIdWithoutObjectGroupId :

contenu : cette map contient l'id du DO relié à un groupe d'objet technique instanciés lors du parcours des objets. création : elle est créé lors de la création du handler MAJ, put : elle est populée lors du parcours des DO dans mapNewTechnicalDataObjectGroupToDO et extractArchiveUnitToLocalFile. Dans extractArchiveUnitToLocalFile, quand on découvre un DataObjectReferenceId et que cet Id se trouve dans dataObjectIdWithoutObjectGroupId alors on récupère l'objet et on change le statut isVisited à true. lecture, get : lecture de la map dans mapNewTechnicalDataObjectGroupToDO, extractArchiveUnitToLocalFile, getNewGdoIdFromGdoByUnit, suppression : c'est un clean en fin d'execution du handler

Le groupe d'objet technique GotObj contient un guid et un boolean isVisited, initialisé à false lors de la création. Le set à true est fait lors du parcours des units.

Map<String, String> objectGroupIdToGuid

contenu : cette map contient l'id du groupe d'objet relié à son guid création : elle est créé lors de la création du handler MAJ, put : elle est populée lors du parcours des DO dans writeDataObjectInLocal et mapNewTechnicalDataObjectGroupToDO lors de la création du groupe d'objet technique lecture, get : lecture de la map dans checkArchiveUnitIdReference, writeDataObjectInLocal, extractArchiveUnitToLocalFile, saveObjectGroupsToWorkspace suppression : c'est un clean en fin d'execution du handler

Map<String, String> objectGroupIdToGuidTmp

contenu : c'est la même map que objectGroupIdToGuid création : elle est créé lors de la création du handler MAJ, put : elle est populée dans writeDataObjectInLocal lecture, get : lecture de la map dans writeDataObjectInLocal suppression : c'est un clean en fin d'execution du handler

Map<String, List<String>> objectGroupIdToDataObjectId

contenu : cette map contient l'id du groupe d'objet relié à son ou ses DO création : elle est créé lors de la création du handler MAJ, put : elle est populée lors du parcours des DO dans writeDataObjectInLocal quand il y a une balise DataObjectGroupId ou DataObjectGroupReferenceId et qu'il n'existe pas dans objectGroupIdToDataObjectId. lecture, get : lecture de la map dans le parcours des DO dans writeDataObjectInLocal. La lecture est faite pour ajouter des DO dans la liste. suppression : c'est un clean en fin d'execution du handler

Map<String, List<String>> objectGroupIdToUnitId

contenu : cette map contient l'id du groupe d'objet relié à ses AU création : elle est créé lors de la création du handler MAJ, put : elle est populée lors du parcours des units dans extractArchiveUnitToLocalFile quand il y a une balise DataObjectGroupId ou DataObjectGroupReferenceId et qu'il nexiste pas dans objectGroupIdToUnitId sinon on ajoute dans la liste des units de la liste lecture, get : lecture de la map dans le parcours des units. La lecture est faite pour ajouter des units dans la liste. suppression : c'est un clean en fin d'execution du handler

Map<String, DataObjectInfo> objectGuidToDataObject

contenu : cette map contient le guid du data object et DataObjectInfo création : elle est créé lors de la création du handler MAJ, put : elle est populer lors de l'extraction des infos du data object vers le workspace lecture, get : elle permet de récupérer les infos binary data object pour sauver l'object group sur le worskapce supression : c'est un clean en fin d'execution du handler

Map<String, String> unitIdToGuid

contenu : cette map contient l'id de l'unit relié à son guid création : elle est créé lors de la création du handler MAJ, put : elle est populée lors du parcours des units dans extractArchiveUnitToLocalFile lecture, get : lecture de la map se fait lors de la création du graph/level des unit dans createIngestLevelStackFile et dans la sauvegarde des object groups vers le workspace suppression : c'est un clean en fin d'execution du handler

Map<String, String> unitIdToGroupId

contenu : cette map contient l'id de l'unit relié à son group id création : elle est créé lors de la création du handler MAJ, put : elle est populée lors du parcours des DO dans writeDataObjectInLocal quand il y a une balise DataObjectGroupId ou DataObjectGroupReferenceId lecture, get : lecture de la map se fait lors de l'extraction des unit dans extractArchiveUnitToLocalFile et permettant de lire dans objectGroupIdToGuid. suppression : c'est un clean en fin d'execution du handler

Map<String, String> objectGuidToUri

contenu : cette map contient le guid du BDO relié à son uri définis dans le manifest création : elle est créé lors de la création du handler MAJ, put : elle est poppulée lors du parcours des DO dans writeDataObjectInLocal quand il rencontre la balise uri lecture, get : lecture de la map se fait lors du save des objects groups dans le workspace suppression : c'est un clean en fin d'execution du handler

sauvegarde des maps (dataObjectIdToObjectGroupId, objectGroupIdToGuid) dans le workspace

1.11.2.4.7.3 4.6.3 Vérifier les ArchiveUnit du SIP

Dans les cas où le SIP contient un objet numérique référencé par un groupe d'objet et qu'une unité archiviste référence cet objet directement (au lieu de déclarer le GOT), le résultat attendu est un statut KO au niveau de l'étape STP_INGEST_CONTROL_SIP dans l'action CHECK_MANIFEST. Ce contrôle est effectué dans la fonction checkArchiveUnitIdReference de ExtractSedaHandler.

Pour ce cas, le map unitIdToGroupId contient une référence entre un unitId et groupId et ce groupId est l'id de l'objet numérique. Dans le objectGroupIdToGuid, il n'existe pas de lien entre id de groupe d'objet et son guid (parce que c'est un id d'object numérique).

On vérifie la valeur des groupIds récupérés dans dataObjectIdToObjectGroupId et unitIdToGroupId. Si ils sont différents, il s'agit du cas abordé ci-dessus, sinon c'est celui des objects numériques sans groupe d'objet technique. Enfin, l'exception ArchiveUnitContainDataObjectException est déclenchée pour ExtractSeda et dans cette étape, le status KO est mise à jour pour l'exécution de l'étape.

L'exécution de l'algorithme est présenté dans le preudo-code ci-dessous :

```
Si (map unitIdToGroupId contient des valeurs)
  Pour (chaque élement ELEM du map unitIdToGroupId)
    Si (la valeur guid de groupe d'object dans objectGroupIdToGuid associé à ELEM) //␣
↪archiveUnit reference par DO
      Prendre (la valeur groupId dans le maps dataObjectIdToObjectGroupId associé à␣
↪groupId d'ELEM)
      Si (cette groupId est NULLE) // ArchiveUnit référencé DO mais il n'existe pas␣
↪un lien DO à groupe d'objet
        Délencher (exception ProcessingException)
      Autrement
        Si (cette groupId est différente grouId associé à ELEM)
          Délencher (exception ArchiveUnitContainDataObjectException)
        Fin Si
      Fin Si
    Fin Si
  Fin Pour
Fin Si
```

1.11.2.4.7.4 4.6.4 Détails du data dans l'itemStatus retourné

Le itemStatus est mis à jour avec les objets du manifest.xml remontées pour mettre à jour evDetData. Il contient dans data le json de evDetData en tant que String. Entre autre, le evDetData contient la valeur evDetDataType à "MASTER" qui définit une action de copie de ce evDetData dans le evDetData master de l'operation. Les champs récupérés (s'ils existent dans le manifest) sont "evDetailReq", "evDateTimeReq", "ArchivalAgreement", "agIfTrans", "ServiceLevel".

1.11.2.4.8 4.7 Détail du handler : IndexObjectGroupActionHandler

1.11.2.4.8.1 4.7.1 description

Indexation des objets groupes en récupérant les objets groupes du workspace. Il y a utilisation d'un client metadata.

1.11.2.4.9 4.8 Détail du handler : IndexUnitActionHandler

1.11.2.4.9.1 4.8.1 description

Indexation des units en récupérant les units du workspace. Il y a utilisation d'un client metadata.

1.11.2.4.10 4.9 Détail du handler : StoreObjectGroupActionHandler

1.11.2.4.10.1 4.9.1 description

Persistence des objets dans l'offre de stockage depuis le workspace.

1.11.2.4.11 4.10 Détail du handler : FormatIdentificationActionHandler

1.11.2.4.11.1 4.10.1 Description

Ce handler permet d'identifier et contrôler automatiquement le format des objets versés. Il s'exécute sur les différents ObjectGroups déclarés dans le manifest. Pour chaque objectGroup, voici ce qui est effectué :

- récupération du JSON de l'objectGroup présent sur le Workspace
- transformation de ce Json en une map d'id d'objets / uri de l'objet associée
- boucle sur les objets :

- téléchargement de l'objet (File) depuis le Workspace
- appel l'outil de vérification de format (actuellement Siegfried) en lui passant le path vers l'objet à identifier + récupération de la réponse.
- appel de l'AdminManagement pour faire une recherche getFormats par rapport au PUID récupéré.
- mise à jour du Json : le format récupéré par Siegfried est mis à jour dans le Json (pour indexation future).
- construction d'une réponse.

- sauvegarde du JSON de l'objectGroup dans le Workspace.
- aggrégation des retours pour générer un message + mise à jour du logbook.

1.11.2.4.11.2 4.10.2 Détail des différentes maps utilisées :

Map<String, String> objectIdToUri

contenu : cette map contient l'id du BDO associé à son uri. création : elle est créée dans le Handler après récupération du json listant les ObjectGroups MAJ, put : elle est populée lors de la lecture du json listant les ObjectGroups. lecture, get : lecture au fur et à mesure du traitement des BDO. suppression : elle n'est pas enregistrée sur le workspace et est présente en mémoire uniquement.

1.11.2.4.11.3 4.10.3 exécution

Ce Handler est exécuté dans l'étape "Contrôle et traitements des objets", juste après le Handler de vérification des empreintes.

1.11.2.4.11.4 4.10.4 journalisation : logbook operation ? logbook life cycle ?

Dans le traitement du Handler, sont mis à jour uniquement les journaux de cycle de vie des ObjectGroups. Les Outcome pour les journaux de cycle de vie peuvent être les suivants :

- Le format PUID n'a pas été trouvé / ne correspond pas avec le référentiel des formats.
- Le format du fichier n'a pas pu être trouvé.
- Le format du fichier a été complété dans les métadonnées (un "diff" est généré et ajouté).
- Le format est correct et correspond au référentiel des formats.

(Note : les messages sont informatifs et ne correspondent aucunement à ce qui sera vraiment inséré en base)

1.11.2.4.11.5 4.10.5 modules utilisés

Le Handler utilise les modules suivants :

- Workspace (récupération / copie de fichiers)
- Logbook (mise à jour des journaux de cycle de vie des ObjectGroups)
- Common-format-identification (appel pour analyse des objets)
- AdminManagement (comparaison format retourné par l'outil d'analyse par rapport au référentiel des formats de Vitam).

1.11.2.4.11.6 4.10.6 cas d'erreur

Les différentes exceptions pouvant être rencontrées :

- ReferentialException : si un problème est rencontré lors de l'interrogation du référentiel des formats de Vitam
- InvalidParseOperationException/InvalidCreateOperationException : si un problème est rencontré lors de la génération de la requête d'interrogation du référentiel des formats de Vitam
- FormatIdentifier*Exception : si un problème est rencontré avec l'outil d'analyse des formats (Siegfried)
- Logbook*Exception : si un problème est rencontré lors de l'interrogation du logbook
- Logbook*Exception : si un problème est rencontré lors de l'interrogation du logbook
- Content*Exception : si un problème est rencontré lors de l'interrogation du workspace
- ProcessingException : si un problème plus général est rencontré dans le Handler

1.11.2.4.12 4.11 Détail du handler : TransferNotificationActionHandler

1.11.2.4.12.1 4.11.1 Description

Ce handler permet de finaliser le processus d'entrée d'un SIP. Cet Handler est un peu spécifique car il sera lancé même si une étape précédente tombe en erreur.

Il permet de générer un xml de notification qui sera :

- une notification KO si une étape du workflow est tombée en erreur.
- une notification OK si le process est OK, et que le SIP a bien été intégré sans erreur.

La première étape dans ce handler est de déterminer l'état du Workflow : OK ou KO.

1.11.2.4.12.2 4.11.2 Détail des différentes maps utilisées :

Map<String, Object> archiveUnitSystemGuid

contenu : cette map contient la liste des archives units avec son identifiant tel que déclaré dans le manifest, associé à son GUID.

Map<String, Object> dataObjectSystemGuid

contenu : cette map contient la liste Data Objects avec leur GUID généré associé à l'identifiant déclaré dans le manifest.

Map<String, Object> bdoObjectGroupSystemGuid

contenu : cette map contient la liste groupes d'objets avec leur GUID généré associé à l'identifiant déclaré dans le manifest.

1.11.2.4.12.3 4.11.3 exécution

Ce Handler est exécuté en dernière position. Il sera exécuté quoi qu'il se passe avant. Même si le processus est KO avant, le Handler sera exécuté.

Cas OK : @TODO@

Cas KO : Pour l'opération d'ingest en cours, on va récupérer dans les logbooks plusieurs informations :

- récupération des logbooks operations générés par l'opération d'ingest.
- récupération des logbooks lifecycles pour les archive units présentes dans le SIP.
- récupération des logbooks lifecycles pour les groupes d'objets présents dans le SIP.

Le Handler s'appuie sur des fichiers qui lui sont transmis. Ces fichiers peuvent ne pas être présents si jamais le process est en erreur avec la génération de ces derniers.

- un fichier globalSedaParameters.file contenant des informations sur le manifest (messageIdentifier).
- un fichier mapsUnits.file : présentant une map d'archive unit
- un fichier mapsDO.file : présentant la liste des data objects
- un fichier mapsDOtoOG.file : mappant le data object à son object group

A noter que ces fichiers ne sont pas obligatoires pour le bon déroulement du handler.

Le handler va alors procéder à la génération d'un XML à partir des informationss aggrégées. Voici sa structure générale :

- MessageIdentifier est rempli avec le MessageIdentifier présent dans le fichier globalSedaParameters. Il est vide si le fichier n'existe pas.
- dans la balise ReplyOutcome :

- dans Operation, on aura une liste d'events remplis par les différentes opérations KO et ou FATAL. La liste sera forcément remplie avec au moins un event. Cette liste est obtenue par l'interrogation de la collection LogbookOperations.
- dans ArchiveUnitList, on aura une liste d'events en erreur. Cette liste est obtenue par l'interrogation de la collection LogbookLifecycleUnits.
- dans DataObjectList, on aura une liste d'events en erreur. Cette liste est obtenue par l'interrogation de la collection LogbookLifecycleObjectGroups.

Le XML est alors enregistré sur le Workspace.

1.11.2.4.12.4 4.11.4 journalisation : logbook operation ? logbook life cycle ?

Dans le traitement du Handler, le logbook est interrogé : opérations et cycles de vie. Cependant aucune mise à jour est effectuée lors de l'exécution de ce handler.

1.11.2.4.12.5 4.11.5 modules utilisés

Le Handler utilise les modules suivants :

- Workspace (récupération / copie de fichiers)
- Logbook (partie server) : pour le moment la partie server du logbook est utilisée pour récupérer les différents journaux (opérations et cycles de vie).
- Storage : permettant de stocker l'ATR.

1.11.2.4.12.6 4.11.6 cas d'erreur

Les différentes exceptions pouvant être rencontrées :

- Logbook*Exception : si un problème est rencontré lors de l'interrogation du logbook
- Content*Exception : si un problème est rencontré lors de l'interrogation du workspace
- XML*Exception : si un souci est rencontré sur la génération du XML
- ProcessingException : si un problème plus général est rencontré dans le Handler

1.11.2.4.13 4.12 Détail du handler : AccessionRegisterActionHandler

1.11.2.4.13.1 4.12.1 Description

AccessionRegisterActionHandler permet de fournir une vue globale et dynamique des archives

sous la responsabilité du service d'archives, pour chaque tenant.

1.11.2.4.13.2 4.12.2 Détail des maps utilisées

Map<String, String> objectGroupIdToGuid
: contenu : cette map contient l'id du groupe d'objet relié à son guid

Map<String, String> archiveUnitIdToGuid
: contenu : cette map contient l'id du groupe d'objet relié à son guid

Map<String, Object> dataObjectIdToDetailDataObject
: contenu : cette map contient l'id du data object relié à ses informations

1.11.2.4.13.3 4.12.3 exécution

L'alimentation du registre des fonds a lieu pendant la phase de finalisation de l'entrée,

une fois que les objets et les units sont rangés. ("stepName" : "STP_INGEST_FINALISATION")

Le Registre des Fonds est alimenté de la manière suivante :

– un identifiant unique – des informations sur le service producteur (OriginatingAgency) – des informations sur le service versant (SubmissionAgency), si différent du service producteur

—des informations sur le contrat (ArchivalAgreement)

– date de début de l'enregistrement (Start Date) – date de fin de l'enregistrement (End Date) – date de dernière mise à jour de l'enregistrement (Last update) – nombre d'units (Total Units) – nombre de GOT (Total ObjectGroups) – nombre d'Objets (Total Objects) – volumétrie des objets (Object Size) – id opération d'entrée associée [pour l'instant, ne comprend que l'evIdProc de l'opération d'entrée concerné] – status (ItemStatus)

1.11.2.4.14 4.13 Détail du handler : CheckIngestContractActionHandler

1.11.2.4.14.1 4.13.1 Description

CheckIngestContractHandler permet de vérifier la présence et contrôler le contrat d'entrée du SIP à télécharger.

1.11.2.4.14.2 4.13.2 Détail des données utilisées

globalSEDAParameters.json Ce handler prend ce fichier comme le parametre d'entrée. Le fichier contient des données gobales sur l'ensemble des parametrès du bordereau et il a été généré à l'étape de l'ExtractSedeActionHandler (CHECK_MANIFEST).

1.11.2.4.14.3 4.13.3 exécution

Le handler cherche d'abord dans globalSEDAParameters.json le nom du contrat déclaré dans le SIP associé au balise <ArchivalAgreement>. Si il n'y as pas de déclaration de contrat d'entrée, le handler retourne le status OK. Si il y a un déclaration de contrat, une liste des opérations suivantes sera effectué :

- recherche du contrat d'entrée déclaré dans la référentiel de contrat
- vérification de contrat :

  si le contrat non trouvé ou contrat trouvé mais en status INACTIVE, le handler retourne le status KO
  si le contrat trouvé et en status ACTIVE, le handler retourne le status OK

L'exécution de l'algorithme est présenté dans le preudo-code ci-dessous :

```
Si (il y as pas de déclaration de contrat)
        handler retourne OK
Autrement
        recherche du contrat dans la base via le client AdminManagementClient
        Si (contrat nou trouvé OU contrat trouvé mais INACTIVE)
                handler retourne KO
        Autrement
            handler retourne OK
        Fin Si
Fin Si
```

1.11.2.4.15 4.14 Détail du handler : CheckNoObjectsActionHandler

1.11.2.4.15.1 4.14.1 Description

CheckNoObjectsActionHandler permet de vérifier s'il y a des objects numériques dans le SIP à verser dans le système.

1.11.2.4.15.2 4.14.2 Détail des données utilisées

Le handler prend ce fichier manifest extrait du WORKSPACE comme le parametre d'entrée.

1.11.2.4.15.3 4.14.3 exécution

Le fichier manifest sera lu pour vérifier s'il y a des TAG "BinaryDataObject" ou "PhysicalDataObject". S'il en y a, le handler retourne KO, sinon OK.

1.11.2.4.16 4.15 Détail du plugin : CheckArchiveUnitSchema

1.11.2.4.16.1 4.15.1 Description

CheckArchiveUnitSchema permet d'exécuter un contrôle intelligent des archive unit en vérifiant la conformité du JSON généré dans le process pour chaque archive unit, par rapport à un schéma défini.

À faire

ne semble pas marcher.

1.11.2.4.16.2 4.15.2 Détail des données utilisées

Le plugin récupère l'id de l'Archive Unit à vérifier.

1.11.2.4.16.3 4.15.3 exécution

A partir de l'Id de l'id de l'Archive Unit à vérifier, le plugin va télécharger le fichier json associé dans le Workspace. Par la suite, il va vérifier la validation de ce Json par rapport au schéma json de Vitam.

1.11.2.4.16.4 4.15.4 détail des vérifications

Dans le schéma Json Vitam défini, voici les spécificités qui ont été ajoutées pour différents champs :

- StartDate pour les Rules : une date contenant une année égale à ou au dessus de l'année 9000 sera refusée.
- Content / Title : peut être de type String, Array ou number (on pourra avoir des titres traduits ainsi que des nombres si besoin)

1.11.2.4.17 4.16 Détail du handler : CheckArchiveProfileActionHandler

1.11.2.4.17.1 4.16.1 Description

Ce handler permet de vérifier le profil dans manifeste

1.11.2.4.17.2 4.16.2 exécution

Le format du profil est XSD ou RNG. L'exécution de l'algorithme est présenté dans le preudo-code ci-dessous :

```
Si le format du profil est équal à XSD
        retourne true si XSD valide le fichier manifest.xml
Fin Si
Si le format du profil est équal à RNG
        retourne true si RNG valide le fichier manifest.xml
Fin Si
```

1.11.2.4.18 4.17 Détail du handler : CheckArchiveProfileRelationActionHandler

1.11.2.4.18.1 4.16.1 Description

Ce handler permet de vérifier la relation entre le contrat d'entrée et le profil dans manifeste

1.11.2.4.18.2 4.16.2 exécution

Si le champ "ArchiveProfiles" dans le contrat d'entrée contient l'identifiant du profil, retourne true

```
    Select select = new Select();
select.setQuery(QueryHelper.eq(IngestContract.NAME, contractName));
JsonNode queryDsl = select.getFinalSelect();
RequestResponse<IngestContractModel> referenceContracts = adminClient.
↪findIngestContracts(queryDsl);
if (referenceContracts.isOk()) {
    IngestContractModel contract = ((RequestResponseOK<IngestContractModel> )␣
↪referenceContracts).getResults().get(0);
    isValid = contract.getArchiveProfiles().contains(profileIdentifier);
}
```

1.11.2.5 5. Worker-common

Le worker-common contient majoritairement des classes utilitaires. A terme, il faudra que SedaUtils notamment soit "retravaillé" pour que les différentes méthodes soit déplacées dans les bons Handlers.

1.11.2.6 6. Worker-client

Le worker client contient le code permettant l'appel vers les API Rest offert par le worker. Pour le moment une seule méthode est offerte : submitStep. Pour plus de détail, voir la partie worker-client.

1.11.3 Worker Client

1.11.3.1 La factory

Afin de récupérer le client une factory a été mise en place. On peut dorenavant lancer plusieurs Client Worker en parallele avec des configurations differentes.

```
WorkerClientFactory.changeMode(WorkerClientConfiguration configuration)
// Récupération du client
WorkerClient client = WorkerClientFactory.getInstance().getClient(configuration);
```

A la demande l'instance courante du client, si un fichier de configuration worker-client.conf est présent dans le classpath le client en mode de production est envoyé, sinon il s'agit du mock.

1.11.3.1.1 Le Mock

En l'absence d'une configuration, le client est en mode Mock. Il est possible de récupérer directement le mock :

```
// Changer la configuration du Factory
WorkerClientFactory.changeMode(null);
// Récupération explicite du client mock
WorkerClient client = WorkerClientFactory.getInstance(null).geClient();
```

1.11.3.1.2 Le mode de production

Pour instancier son client en mode Production :

```
// Changer la configuration du Factory
WorkerClientFactory.changeMode(WorkerClientConfiguration configuration);
//creation du de la configuration
WorkerClientConfiguration workerClientConfiguration = new WorkerClientConfiguration(
                "localhost",
                 8067
);
// Récupération explicite du client
WorkerClient client = WorkerClientFactory.getInstance(workerClientConfiguration).
↪getClient();
```

1.11.3.2 Les services

Le client propose pour le moment une fonctionnalité : - Permet de soumettre le lancement d'une étape. Deux paramètres sont nécessaires : un string requestId + un objet DescriptionStep. Voici un exemple d'utilisation :

```
DescriptionStep ds = new DescriptionStep(new Step(), new WorkParams());
List<EngineResponse> responses =
         client.submitStep("requestId", ds);
// Now we can check the list of response
```

1.12 Workspace

1.12.1 Introduction

1.12.1.1 But de cette documentation

L'objectif de cette documentation est de compléter la Javadoc pour ce module.

1.12.2 workspace

le workspace est un module qui consiste à stocker le sip dans un container lors de traitement. Il y a un controle des paramètres (SanityChecker.checkJsonAll) transmis avec ESAPI.

1.12.2.1 1- Consommer les services exposés par le module :

1.1 - Introduction :

on peut consommer les services via le sous module workspaceClient notament via la classe WorkspaceClient :
Cette classe contient la liste des methodes suivantes :

- CreateContainer :
  - Paramètres :
  - containerName : :String
  - Retourner :
- getUriListDigitalObjectFromFolder :
  - Paramètres :
    - containerName : :String
    - folderName : :String
  - Retourner :
    - List<URI>

Dans le cas echéant la method return une immuatable empty list.

- uncompressObject : cette méthode capable d'extracter des fichiers compressés toute en indiquant le type de l'archive, pour cette version (v0.9.0) supporte 3 types : zip, tar, tar.gz. Elle sauvgarde directement les fichiers extractés dans le workspace, notamment dans le container précisé lors de l'appel (containerName).

  -Paramètres :

  - containerName : :String : c'est le nom de container dans lequel on stocke les objets
  - folderName : :String : c'est le repertoire centrale (pour cette methode, cest le sip).
  - **archiveType : : String** [c'est le nom ou le type de l'archive (exemple : application/zip , application/x-tar)]
    - compressedInputStream : :InputStream : c'est le stream des objets compressés
- retourner :

Dans le cas echéant (uncompress KO) la methode génère une exception avec un message internal server.

- getObjectInformation :

- Paramètres :
- containerName : :String
- objectName : :String
- Retourner :
- JsonNode

  La méthode retourne un Json contenant des informations sur un objet présent sur le workspace (et des exceptions en cas d'erreur : objet non existant, erreur server).

1.12.2.2 2.2 - Exemple d'utilisation

D'abord il faut ajouter la dependence sur la pom.xml du projet.

```
<dependencies>
    <groupId>fr.gouv.vitam</groupId>
    <artifactId>workspace-client<artifactId>
    <version>x.x.x</version>
</dependencies>
```

Supposons que nous avons besoins d'extraire un SIP de format zip dans le workspace.

```
InputStream inputStream=new InputStream(zippedFile);
WorkspaceClientFactory.changeMode(WORKSPACE_URL);
WorkspaceClientFactory.changeMode(FileConfiguration);
WorkspaceClient workspaceClient = WorkspaceClientFactory().getInstance().getClient();
workspaceClient.createContainer(containerName);
workspaceClient.uncompressObject(containerName,"SIP","application/zip" inputStream);
```

1.12.2.3 2- Configuration du pom

Configuration du pom avec maven-surefire-plugin permet le build sous jenkins. Il permet de configurer le chemin des resources de esapi dans le common private.

CHAPITRE 2

Parallélisation des tests

Ce document présente la procédure pour réduire le temps de traitement des tests en les parallélisant. Ce travail réfère au US#714 et au techDesign IT01.

Il y a des tests TDD et des tests d'intégration dans les modules existants de la plate-forme, nous voulons faire paralléliser des classes de tests utilisant JUnit pour avoir la performance. Pour ce but, nous effectuons les étapes suivantes :

- Séparation des tests : tests unitaires et test d'intégration
- Parallélisation des tests unitaires
- Configuration de build avec les options de tests

Séparation des tests TDD et tests d'intégration

- **Il y a trois tests d'intégration et nous les remettons dans le module test-intégration parent** ProcessingIT : test d'intégration pour différents services : workspace, functional-administration, worker, metadata, logbook, processing

  StorageClientIT : test d'intégration pour le client du service de storage. Cela concerne deux modules : storage (client & rest) et le client de workspace

  WorkerIT : test d'intégration pour les services : workspace, worker, metadata, logbook, processing

  Ces tests d'intégration sont en mode séquentiel. Pour cela, nous indiquons dans le pom.xml de ce module de test-integration

```
<build>
        <pluginManagement>
                <plugins>
                        <plugin>
                                <!-- Run the Junit unit tests in an isolated␣
↪classloader and not Parallel. -->
                                <artifactId>maven-surefire-plugin</artifactId>
                                <configuration>
                                        <parallel>classes</parallel>
                                        <threadCount>1</threadCount>
                                        <perCoreThreadCount>false</perCoreThreadCount>
                                        <forkCount>1</forkCount>
                                        <reuseForks>false</reuseForks>
                                        <systemPropertyVariables>
                                                <org.owasp.esapi.opsteam>AC001</org.
↪owasp.esapi.opsteam>
                                                <org.owasp.esapi.devteam>AC001</org.
↪owasp.esapi.devteam>
                                                <org.owasp.esapi.resources>../common/
↪common-private/src/main/resources/esapi</org.owasp.esapi.resources>
                                        </systemPropertyVariables>
                                </configuration>
                        </plugin>
                </plugins>
        </pluginManagement>
</build>
```

Parallélisation de tests unitaires

Les tests unitaires de chaque module sont configurés pour être lancé en mode parallèle. Pour cela, nous indiquons dans le pom.xml parent pour la phrase de build

```
<build>
        <plugins>
                <plugin>
                        <!-- Run the Junit unit tests in an isolated classloader. -->
                        <artifactId>maven-surefire-plugin</artifactId>
                        <version>2.19.1</version>
                        <configuration>
                                <argLine>-Xmx2048m -Dvitam.tmp.folder=/tmp $
↪{coverageAgent}</argLine>
                                <parallel>classes</parallel>
                                <threadCount>3</threadCount>
                                <perCoreThreadCount>true</perCoreThreadCount>
                                <forkCount>3C</forkCount>
                                <reuseForks>false</reuseForks>
                                <trimStackTrace>false</trimStackTrace>
                        </configuration>
                </plugin>
        </plugins>
        </build>
```

Configuration de build avec les options de tests

- mvn install : lancer le build normal avec tous les tests
- mvn clean install -DskipTests : pour ignorer tous les tests :
- mvn clean test ou mvn clean install -DskipITs : pour ignorer les tests d'intégration

Pour cela, nous ajoutons le code suivant dans le pom parent.

```
<plugin>
        <executions>
                <execution>
                        <id>integration-test</id>
                        <goals>
                                <goal>test</goal>
                        </goals>
                        <phase>integration-test</phase>
                        <configuration>
                                <skip>${skipITs}</skip>
                                <excludes>
                                        <exclude>none</exclude>
                                </excludes>
                                <includes>
                                        <include>**/*IT.java</include>
                                </includes>
                        </configuration>
                </execution>
        </executions>
</plugin>
```

- mvn clean test-compile failsafe :integration-test : pour exécuter uniquement les tests d'intégration.

Pour cela, nous ajoutons le code suivant dans le pom parent.

```
<build>
        <plugin>
                <!-- Run the Junit integration tests in an isolated classloader. -->
                <artifactId>maven-failsafe-plugin</artifactId>
                <version>2.19.1</version>
                <executions>
                        <execution>
                                <id>integration-test</id>
                                <goals>
                                        <goal>integration-test</goal>
```

```
                                        <goal>verify</goal>
                                    </goals>
                                </execution>
                            </executions>
                        </plugin>
</build>
```

CHAPITRE 6

Plugin ICU Elasticsearch

Le letter tokenizer elasticsearch qu'on utilise aujourd'jui n'indexe pas les chiffres. Pour pouvoir les indexer les chiffres, nous avons besoin d'un plugin qui hérite ce letter tokenizer. Nous avons choisi le plugin ICU analysis pour Elasticsearch, https://github.com/elasticsearch/elasticsearch-analysis-icu cela.

Ce plugin est installé lors de déploiement du système qui associe au Node Elastichsearch Vitam, qui permet aux autres services de les appeler.

CHAPITRE 7

Gestion des bases de données

Ce document présente les points d'attention et une check list quand vous avez une modification à faire sur un schéma de données d'une base de données ou la création d'une requête particulière MongoDB.

7.1 Gestion de l'ajout d'un champ

Si ce champ n'est pas "protégé" (non préfixé par "_"), seuls les aspects indexations sont à suivre.

Si ce champ est "protégé" (préfixé par un "_"), quelques règles d'usages sont à respecter :

- Il est préfixé en base par "_" afin de ne pas être en conflit avec des métadonnées externes (notamment pour le "*content*" du Unit)
- Le nom dans la base doit être court (exemple : **_us**) afin de limiter l'empreinte mémoire et disque de ce champs tant pour les index que pour les données, et tant pour MongoDB que pour ElasticSearch
- Le nom du point de vue usage (externe et interne) doit être explicite (exemple : **allunitups**)
- Il est préfixé d'un '**#**' pour permettre son interprétation par Vitam comme un champ protégé
- Il cache l'implémentation réelle du champ

Certains de ces champs sont interdits en update/insert (depuis l'extérieur), mais autorisés en interne.

La définition d'un tel champ "protégé" s'effectue ainsi :

- common-database-vitam
  - common-database-public
    - BuilderToken.java : il contient un enum simple définisssant le champ (exemple : **ALLUNITUPS("allunitups")**)
    - VitamFieldsHelper.java : il contient des helpers pour accéder directement à la représentation formelle (précédé du '**#**') le champ (exemple : **allunitups()**)
      Le QueryBuilder interdit les champs préfixés par "_". Il impose donc l'usage de la notation '**#**'.
  - commmon-database-private
    - ParserTokens.java : il contient la copie exacte de BulderToken mais y ajoute les méthodes
      - **notAllowedOnSet()** qui interdit ou pas l'update/insert depuis l'extérieur. Ce check est réalisé par les API-internal via les VarNameAdapter.
      - **getPROJECTIONARGS()*** qui traduit du champ interne en champ externe. Cette fonction est utilisé par les deux ci-dessous.
      - **isNotAnalyzed()** qui indique si le champ n'est pas indexé
      - **isAnArray()** qui indique si le champ est un tableau
      - **isSingleProtectedVariable** désigne les variables de collections Single

    - **isAnArrayVariable** désigne les variables de collections Single ou Multiple
    - **isSingleNotAnalyzedVariable** désigne les variables de collections Single
  - VarNameAdapter.java pour Unit/ObjectGroup
  - VarNameInsertAdapter.java pour Unit/ObjectGroup
  - VarNameUpdateAdapter.java pour Unit/ObjectGroup *(devra être dupliqué en usage externe et interne : protection de certains champs)*
  - SingleVarNameAdapter.java pour les collections hors Unit/ObjectGroup pour usage interne
  - SingleVarNameAdapterExternal.java pour usage externe pour les collections hors Unit/ObjectGroup

7.1.1 metadata-core : Unit et ObjectGroup

- MongoDbVarNameAdapter.java : autorise les update/insert sur les **#protégés** et trafuit dans les champs définitifs définis dans MetadataDocument.java, Unit.java et ObjectGroup.java (exemple : **#allunitups** en **_us**)
- MongoDbMetadataResponseFilter.java : récupère la réponse et retraduit en sens inverse un champs "**_xxx**" en son correspondant "**#xxxxxxxxx**" (exemple : **_us** en **#allunitups**)
- MetadataDocument.java et Unit.java et ObjectGroup.java pour la définition des champs traduits en interne (formats courts comme "**_us**" et non "**_unitsparents**")

7.1.2 Pour les autres collections

Elle s'appuie sur SingleVarNameAdapater et devraient avoir leurs propres extensions (comme MongoDbVarNameAdapter) ainsi que pour les retours (comme MongoDbMetadataResponseFilter)

7.2 Modification d'une collection : check list

- Pour les champs protégés (préfix **#**)
  - Ajouter le champ dans les classes BuilderToken, VitamFieldsHelper, ParserTokens
  - Vérifier/Modifier les VarNameAdapter de la collection s'ils sont bien pris en compte (tant pour les cas Insert/Update interdits ou pas que pour la traduction dans le nom du champ final)
  - Modifier le ResponseFilter de la collection pour retraduire en #xxxxx la réponse
- Pour tous les champs
  - Mettre à jour le schéma Json pour prendre en compte le nouveau champ et son type
  - Si ce champ est utilisé dans des requêtes MongoDB et/ou consitue une clef primaire modifier avec l'intégration les index techniques MongoDb (optimisation et unicité)

CHAPITRE 8

Annexes

Table des figures

Liste des tableaux