VITAM - Manuel de développement

Version 0.11.1-RC2-SNAPSHOT

VITAM

janv. 12, 2017

Table des matières

Prudence : Cette documentation est un travail en cours ; elle est susceptible de changer de manière conséquente.

Règles à respecter :

- noms de fichiers / dossiers : tout en minuscule
- les tables des matières dans un fichier _toc.rst à la racine du dossier correspondant

CHAPITRE 1

Sommaire

1.1 Détails par composant

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

1.1.1 Access

1.1.1.1 Introduction

1.1.1.1.1 But de cette documentation

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

1.1.1.2 Access

1.1.1.3 Utilisation

1.1.1.3.1 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.

1.1.1.3.2 La factory

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

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

1.1.1.3.2.1 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();
```

1.1.1.3.3 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)

1.1.1.3.4 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.

1.1.1.3.4.1 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);
```

1.1.1.3.4.2 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);
 }
```

1.1.2 Common

1.1.2.1 Introduction

1.1.2.1.1 But de cette documentation

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

1.1.2.1.2 Utilitaires Commons

1.1.2.1.2.1 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.

1.1.2.1.2.2 LocalDateUtil

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

1.1.2.1.2.3 ServerIdentity

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

ServerIdentity contient le ServerName, le ServerRole, et le Global PlatformId.

Pour une JVM, un seul ServerIdentity existe.

C'est un Common Private.

Par défaut cette classe est initialisée avec les valeurs suivantes : * ServerName : hostname ou UnknownHostname si introuvable * ServerRole : UnknownRole * PlatformId : MAC adresse partielle comme entier

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.

1.1.2.1.2.4 Usage

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

1.1.2.1.2.5 Les usages principaux

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

1.1.2.1.2.6 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

1.1.2.1.2.7 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)

1.1.2.1.2.8 BaseXXX

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

1.1.2.1.2.9 CharsetUtils

Cet utilitaire propose quelques méthodes pour la gestion des Charset.

1.1.2.1.2.10 ParametersChecker

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

1.1.2.1.2.11 SingletonUtil

Cet utilitaire propose quelques méthodes pour obtenir des Singletons.

1.1.2.1.2.12 StringUtils

Cet utilitaire propose quelques méthodes pour manipuler des String.

1.1.2.1.3 GUID

Cf chapitre dédié.

1.1.2.1.4 Logging

Cf chapitre dédié.

1.1.2.1.5 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

1.1.2.1.6 Digest

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

Cf chapitre dédié.

1.1.2.1.7 Json

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

Ce module propose une configuration par défaut pour Vitam.

1.1.2.1.8 Exception

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

1.1.2.1.9 Client

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

1.1.2.2 Global Unique Identifier (GUID) pour Vitam

1.1.2.2.1 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).

1.1.2.2.2 GUID Factory

Usage :

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

1.1.2.2.2.1 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();
```

1.1.2.2.2.2 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.

1.1.2.2.3 Attention

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

1.1.2.3 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

1.1.2.3.1 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)
```

1.1.2.4 Logging

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

1.1.2.4.1 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.

1.1.2.4.2 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

1.1.2.4.3 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
```

1.1.2.5 JunitHelper

1.1.2.5.1 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();
```

1.1.2.6 Client

1.1.2.6.1 But de cette documentation

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

1.1.2.6.2 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();
```

1.1.2.6.3 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.

1.1.2.7 DirectedCycle

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

1.1.2.7.1 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);
```

1.1.2.7.2 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”]}

}

1.1.2.7.3 Remarque

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

1.1.2.8 Graph

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

1.1.2.8.1 Initialisation

Pour initialiser un objet Graph :

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

1.1.2.8.2 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]}

1.1.2.9 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.

1.1.2.9.1 Les codes

1.1.2.9.1.1 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)).

1.1.2.9.1.2 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)).

1.1.2.9.1.3 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.

1.1.2.9.1.4 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).

1.1.2.9.2 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);
```

1.1.2.10 Common format identification

1.1.2.10.1 But de cette documentation

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

1.1.2.10.2 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 :

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 :

1.1.2.10.3 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.

1.1.2.11 Common-private

1.1.2.11.1 Génération de certificats et de keystore

1.1.2.11.2 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.

< ?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>

1.1.2.11.3 Format Identifiers

1.1.2.11.3.1 But de cette documentation

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

1.1.2.11.3.2 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.

1.1.2.11.3.3 Implémentation Mock

Implémentation simple renvoyant des réponses statiques.

1.1.2.11.3.4 Implémentation Siegried

Implémentation basique utilisant un client HTTP.

1.1.2.11.3.5 Format Identifier Factory

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

1.1.2.11.3.6 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.

1.1.2.11.3.7 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");
```

1.1.2.11.4 Introduction

1.1.2.11.4.1 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.

1.1.2.11.5 DAT : module Graph

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

- 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.

1.1.2.11.6 Paramètres

1.1.2.11.6.1 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.

1.1.2.11.6.2 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.

1.1.2.11.6.3 Mise en place

1.1.2.11.6.4 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;
}
```

1.1.2.11.6.5 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();
 }
```

1.1.2.11.6.6 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);
    }
}
```

1.1.2.11.6.7 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
    // ...
}
```

1.1.2.11.6.8 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..
}
```

1.1.2.11.6.9 Exemple d'utilisation dans le code Vitam

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

1.1.2.11.7 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

1.1.2.11.7.1 fonctions

UriUtils.splitUri(String uriString)

1.1.2.11.8 Implémentation du WAF dans jetty

1.1.2.11.8.1 But de cette documentation

On présente dans ce document l'implémentation de WAF dans jetty serveur

1.1.2.11.8.2 Fonctionnement général du filtre

La class WafFilter utilise XSSWrapper pour valider les en-têtes et les paramètres dans les requêtes http. Si on détecte une faille XSS, on retourne directement la réponse 406(NOT ACCEPTABLE) au client

1.1.2.11.8.3 Implémentation du WAF

1.1.3 Functional administration

1.1.3.1 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

1.1.3.2 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.

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.

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 traitement MongoDbAccess 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

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

2.2. 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.

2.2. 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.

1.1.4 IHM demo

1.1.4.1 Introduction

1.1.4.1.1 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

1.1.4.2 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

1.1.5 Ingest

1.1.5.1 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

1.1.5.2 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 - IngestInternalInternalClient-Mock.java : mock client ingest-internal - IngestInternalClientRest.java : le client ingest-internal et des fonctionnalités en se connectant au serveur REST

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 - IngestExternalexternalClient-Mock.java : mock client ingest-external - IngestExternalClientRest.java : le client ingest-external et des fonctionnalités en se connectant au serveur REST ingest-external

1.1.5.3 ingest-internal-client

1.1.5.4 Utilisation

1.1.5.4.1 Paramètres

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

1.1.5.4.2 La factory

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

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

1.1.5.4.2.1 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();
```

1.1.5.4.3 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 deux méthodes :

```
         Status status();
         UploadResponseDTO upload(String archiveMimeType,List<LogbookParameters>␣
↪logbookParametersList, InputStream inputStream);



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 :

1.1.5.5 ingest-external-client

1.1.5.6 Utilisation

1.1.5.6.1 Paramètres

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

1.1.5.6.2 La factory

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

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

1.1.5.6.2.1 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();
```

1.1.5.6.3 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 propose actuellement deux méthodes :

```
Status status();
void upload(InputStream stream)
```

1.1.5.7 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.

.....Java 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 ;
```

1.1.6 Logbook

1.1.6.1 Introduction

1.1.6.1.1 But de cette documentation

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

1.1.6.2 Logbook

1.1.6.3 Utilisation

1.1.6.3.1 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.

1.1.6.3.2 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);
```

1.1.6.3.2.1 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();
```

1.1.6.3.3 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);
```

1.1.6.3.3.1 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();
```

1.1.6.3.3.2 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();
```

1.1.6.3.3.3 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);
```

1.1.7 Metadata

1.1.7.1 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

1.1.7.2 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

1.1.7.3 Métadata

1.1.7.4 Utilisation

1.1.7.4.1 Paramètres

1.1.7.4.2 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>

1.1.7.5 Métadata : API REST Raml

1.1.7.5.1 Présentation

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

Module pour le module opération : api / rest.

1.1.7.5.2 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**

1.1.8 Processing

1.1.8.1 Introduction

1.1.8.1.1 But de cette documentation

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

1.1.8.2 Paramètres

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

1.1.8.2.1 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.

1.1.8.2.2 ParameterHelper, le helper

Utiliser le ParameterHelper afin de valider les éléments requis.

1.1.8.2.3 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

1.1.8.2.4 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).

1.1.8.2.5 DefaultWorkerParameters, l'implémentation actuelle

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

1.1.8.3 Processing Management

1.1.8.3.1 Présentation

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

2 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.

1.1.8.3.1.1 Processing-management

1.1.8.3.2 Rest API

Dans le module Processing-management (package rest) : | http://server/processing/v1 | POST /operations -> **POST Soumettre un workflow à éxécution** | GET /status -> **statut du logbook**

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**

1.1.8.3.3 Core

Dans la partie Core, la classe ProcessManagementImpl propose une méthode : submitWorkflow. Elle permet de lancer un workflow.

1.1.8.3.3.1 Processing-management-client

1.1.8.3.4 Utilisation

Le client propose une méthode principale : executeVitamProcess. Deux autres méthodes ont été ajoutées : registerWorker et unregisterWorker.

- executeVitamProcess permet de traiter les opérations de "workflow".
- registerWorker : permet d'ajouter un nouveau worker à la liste des workers.
- unregisterWorker : permet de supprimer un worker à la liste des workers.

1.1.8.3.4.1 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.

1.1.8.4 Processing Distributor

1.1.8.4.1 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.

1.1.8.4.1.1 Processing-distributor

1.1.8.4.2 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.

1.1.8.4.3 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.

1.1.8.4.3.1 Processing-distributor-client

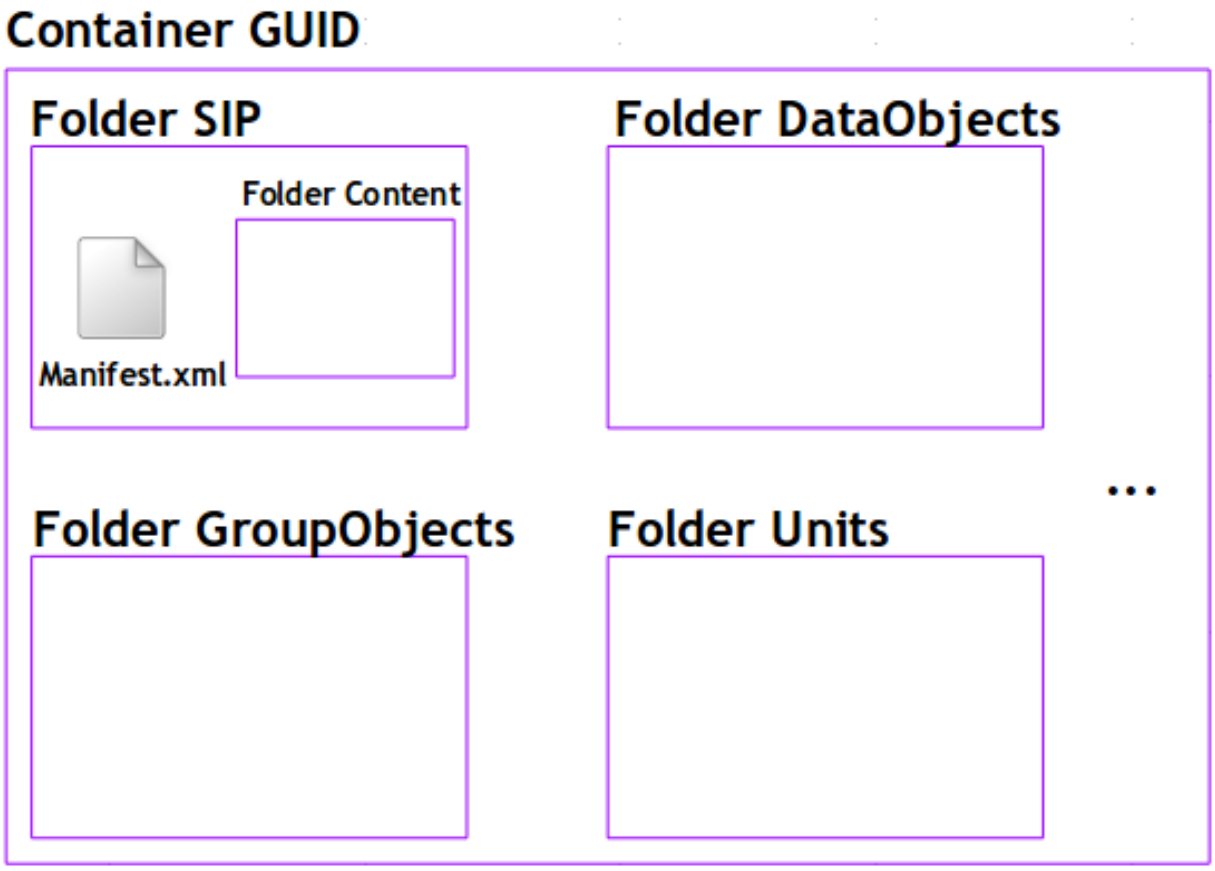
Pour le momentn le module est vide, car la partie client permettant d'appeler les méthodes register / unregister est portée par le module processing-management-client. A terme, il sera souhaité d'avoir 2 clients séparés.

1.1.8.5 Etudes en cours

1.1.8.5.1 Workspace

1.1.8.5.1.1 Arborescence

- ** example d'arborescence d'un container dans le workspace ** :



- ** détails ** :

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

1.1.8.5.2 Workflow

1.1.8.5.2.1 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

```
{
  "id": "DefaultIngestWorkflow",
  "comment": "Default Ingest Workflow V6",
  "steps": [
    {
      "workerGroupId": "DefaultWorker",
      "stepName": "STP_INGEST_CONTROL_SIP",
      "behavior": "BLOCKING",
      "distribution": {
        "kind": "REF",
        "element": "SIP/manifest.xml"
      },
      "actions": [
        {
          "action": {
            "actionKey": "CHECK_SEDA",
            "behavior": "BLOCKING"
          }
        },
        {
          "action": {
            "actionKey": "CHECK_MANIFEST_DATAOBJECT_VERSION",
            "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": "mapsBDOtoOG.file",
                "uri": "WORKSPACE:Maps/BDO_TO_OBJECT_GROUP_ID_MAP.json"
              },
              {
                "name": "mapsBDO.file",
                "uri": "WORKSPACE:Maps/BINARY_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": "mapsBDOtoVersionBDO.file",
```

```
                "uri": "WORKSPACE:Maps/BDO_TO_VERSION_BDO_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_CONSISTENCY",
            "behavior": "NOBLOCKING",
            "in": [
              {
                "name": "mapsBDOtoOG.file",
                "uri": "WORKSPACE:Maps/OG_TO_ARCHIVE_ID_MAP.json"
              },
              {
                "name": "mapsBDOtoOG.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": "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": "mapsBDO.file",
                "uri": "WORKSPACE:Maps/OBJECT_GROUP_ID_TO_GUID_MAP.json"
              },
              {
                "name": "mapsBDO.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": "mapsBDO.file",
                "uri": "WORKSPACE:Maps/BINARY_DATA_OBJECT_ID_TO_GUID_MAP.json",
                "optional": "true"
              },
              {
                "name": "mapsBDOtoOG.file",
                "uri": "WORKSPACE:Maps/BDO_TO_OBJECT_GROUP_ID_MAP.json",
                "optional": "true"
              },
              {
                "name": "mapsBDOtoVersionBDO.file",
                "uri": "WORKSPACE:Maps/BDO_TO_VERSION_BDO_MAP.json",
                "optional": "true"
              },
              {
                "name": "globalSEDAParameters.file",
                "uri": "WORKSPACE:ATR/globalSEDAParameters.json",
                "optional": "true"
              }
            ],
            "out": [
              {
                "name": "atr.file",
                "uri": "WORKSPACE:ATR/responseReply.xml"
              }
            ]
          }
        }
      ]
    }
  ]
}
```


- **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_CONSISTENCY : - 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_CONSISTENCY_POST : 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_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 4** - 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 5** - STP_ACCESSION_REGISTRATION : Alimentation du registre de fond
    - ACCESSION_REGISTRATION : enregistrement des archives prises en charge dans le Registre des Fonds
- **Step 6 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

1.1.8.5.2.2 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).

1.1.9 Storage

1.1.9.1 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.

1.1.9.1.1 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.

1.1.9.1.1.1 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
}
```

1.1.9.1.1.2 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)) {
    // Requête contant le tenantId afin de récupérer la capacité (objet permettant d
↪'être enrichi dan le futur)
    StorageCapacityRequest request = new StorageCapacityRequest();
    request.setTenantId("tenantId");
    // Récupération de la capacité
    StorageCapacityResult capacity = myConnection.getStorageCapacity(request);
    // 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
}
```

1.1.9.1.1.3 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");

// 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)) {
    PutObjectRequest request = new PutObjectRequest();
    request.setDataStream(new FileInputStream(PropertiesUtils.findFile("digitalObject.
↪pdf")));
    request.setDigestAlgorithm(DigestType.MD5.getName());
    request.setGuid("GUID");
    request.setTenantId("0");
    request.setFolder(DataCategory.OBJECT.getFolder());
    PutObjectResult 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
}
```

1.1.9.2 Storage Engine

1.1.9.3 Storage Engine Client

1.1.9.3.1 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.

1.1.9.3.1.1 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
StorageClientFactory.changeMode(null)
  // Récupération explicite du client mock
  StorageClient client = StorageClientFactory.getInstance().getClient();
```

1.1.9.3.1.2 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();
```

1.1.9.3.2 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 du status est également disponible :

```
StatusMessage status = client.getStatus();
```

1.1.10 Technical administration

1.1.11 Worker

1.1.11.1 Introduction

1.1.11.1.1 But de cette documentation

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

1.1.11.2 Worker

1.1.11.2.1 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.

1.1.11.2.2 2. Worker-server

1.1.11.2.2.1 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**

1.1.11.2.2.2 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.

1.1.11.2.3 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 - IndexObjectGroupActionHandler - IndexUnitActionHandler - StoreObjectGroupActionHandler - FormatIdentificationActionHandler - AccessionRegisterActionHandler - TransferNotificationActionHandler - UnitsRulesCompteHandler - DummyHandler

La classe WorkerImpl permet de lancer ces différents handlers.

1.1.11.2.3.1 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
```

1.1.11.2.3.2 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 :

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(BDO_TO_BDO_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
```

1.1.11.2.3.3 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.rst

1.1.11.2.4 4. Details des Handlers

1.1.11.2.4.1 4.1 Détail du handler : CheckConformityActionHandler

1.1.11.2.4.2 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

1.1.11.2.4.3 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

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.

1.1.11.2.4.4 4.1.3 journalisation :

1.1.11.2.4.5 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

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

1.1.11.2.4.6 4.1.5 modules utilisés

processing, worker, workspace et logbook

1.1.11.2.4.7 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

1.1.11.2.4.8 4.2 Détail du handler : CheckObjectsNumberActionHandler

1.1.11.2.4.9 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.

1.1.11.2.4.10 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 :* .. code-block : : java .....

```
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, HAN-
              DLER_ID, LogbookTypeProcess.INGEST, objectGroupToGuid-
              StoredMap.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 ....
    }
}
```

1.1.11.2.4.11 4.4 Détail du handler : CheckSedaActionHandler

TODO

1.1.11.2.4.12 4.4 Détail du handler : CheckStorageAvailabilityActionHandler

TODO

1.1.11.2.4.13 4.5 Détail du handler : CheckVersionActionHandler

TODO

1.1.11.2.4.14 4.6 Détail du handler : ExtractSedaActionHandler

1.1.11.2.4.15 4.6.1 description

Ce handler permet d'extraire le contenu du SEDA. Il y a : - extraction des BinaryDataObject - 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

1.1.11.2.4.16 4.6.2 Détail des différentes maps utilisées :

Map<String, String> binaryDataObjectIdToGuid contenu : cette map contient l'id du BDO 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 lecture, get : saveObjectGroupsToWorkspace, getObjectGroupQualifiers, suppression : c'est un clean en fin d'execution du handler

Map<String, String> binaryDataObjectIdToObjectGroupId : contenu : cette map contient l'id du BDO 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 lecture, get : lecture de la map dans mapNewTechnicalDataObjectGroupToBDO, getNewGdoIdFromGdoByUnit, completeBinaryObjectToObjectGroupMap, checkArchiveUnitIdReference et writeBinaryDataObjectInLocal suppression : c'est un clean en fin d'execution du handler

Map<String, GotObj> binaryDataObjectIdWithoutObjectGroupId : contenu : cette map contient l'id du BDO relié à un groupe d'objet technique instanciés lors du parcours des objets binaires. création : elle est créé lors de la création du handler MAJ, put : elle est populée lors du parcours des BDO dans mapNewTechnicalDataObjectGroupToBDO et extractArchiveUnitToLocalFile. Dans extractArchiveUnitToLocalFile, quand on découvre un DataObjectReferenceId et que cet Id se trouve dans binaryDataObjectIdWithoutObjectGroupId alors on récupère l'objet et on change le statut isVisited à true. lecture, get : lecture de la map dans mapNewTechnicalDataObjectGroupToBDO, 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 BDO dans writeBinaryDataObjectInLocal et mapNewTechnicalDataObjectGroupToBDO lors de la création du groupe d'objet technique lecture, get : lecture de la map dans checkArchiveUnitIdReference, writeBinaryDataObjectInLocal, 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 writeBinaryDataObjectInLocal lecture, get : lecture de la map dans writeBinaryDataObjectInLocal suppression : c'est un clean en fin d'execution du handler

Map<String, List<String>> objectGroupIdToBinaryDataObjectId contenu : cette map contient l'id du groupe d'objet relié à son ou ses BDO création : elle est créé lors de la création du handler MAJ, put : elle est populée lors du parcours des BDO dans writeBinaryDataObjectInLocal quand il y a une balise DataObjectGroupId ou DataObjectGroupReferenceId et qu'il n'existe pas dans objectGroupIdToBinaryDataObjectId. lecture, get : lecture de la map dans le parcours des BDO dans writeBinaryDataObjectInLocal. La lecture est faite pour ajouter des BDO 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, BinaryObjectInfo> objectGuidToBinaryObject contenu : cette map contient le guid du binary data object et BinaryObjectInfo création : elle est créé lors de la création du handler MAJ, put : elle est populer lors de l'extraction des infos du binary data object vers le workspace lecture, get : elle permet de récupérer les infos binary data object pour sauver l'object group sur le worskapce et 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 BDO dans writeBinaryDataObjectInLocal 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 BDO dans writeBinaryDataObjectInLocal 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 (binaryDataObjectIdToObjectGroupId, objectGroupIdToGuid) dans le workspace

1.1.11.2.4.17 4.7 Détail du handler : IndexObjectGroupActionHandler

1.1.11.2.4.18 4.7.1 description

Indexation des objets groupes en récupérant les objets groupes du workspace. Il y a utilisation d'un client metadata. TODO

1.1.11.2.4.19 4.8 Détail du handler : IndexUnitActionHandler

1.1.11.2.4.20 4.8.1 description

Indexation des units en récupérant les units du workspace. Il y a utilisation d'un client metadata. TODO

1.1.11.2.4.21 4.9 Détail du handler : StoreObjectGroupActionHandler

1.1.11.2.4.22 4.9.1 description

Persistence des objets dans l'offre de stockage depuis le workspace.

TODO

1.1.11.2.4.23 4.10 Détail du handler : FormatIdentificationActionHandler

1.1.11.2.4.24 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.

1.1.11.2.4.25 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.

1.1.11.2.4.26 4.10.3 exécution

Ce Handler est exécuté dans l'étape "Contrôle et traitements des objets", juste après le Handler de vérification des empreintes.

1.1.11.2.4.27 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)

1.1.11.2.4.28 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).

1.1.11.2.4.29 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

1.1.11.2.4.30 4.11 Détail du handler : TransferNotificationActionHandler

1.1.11.2.4.31 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.

1.1.11.2.4.32 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> binaryDataObjectSystemGuid 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.

1.1.11.2.4.33 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 er

- un fichier globalSedaParameters.file contenant des informations sur le manifest (messageIdentifier).

- un fichier mapsUnits.file : présentant une map d'archive unit
- un fichier mapsBDO.file : présentant la liste des binary data objects
- un fichier mapsBDOtoOG=.file : mappant le binary 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.

1.1.11.2.4.34 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.

1.1.11.2.4.35 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.

1.1.11.2.4.36 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

1.1.11.2.4.37 4.12 Détail du handler : AccessionRegisterActionHandler

1.1.11.2.4.38 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.

1.1.11.2.4.39 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> bdoToBdoInfo contenu : cette map contient l'id du binary data object relié à son information

1.1.11.2.4.40 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 – 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)

1.1.11.2.5 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.

1.1.11.2.6 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.

1.1.11.3 Worker Client

1.1.11.3.1 La factory

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

```
WorkerClientFactory.changeMode(WorkerClientConfiguration configuration)

// Récupération du client
WorkerClient client = WorkerClientFactory.getInstance().getClient();
```

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.

1.1.11.3.1.1 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().geClient();
```

1.1.11.3.1.2 Le mode de production

Pour instancier son client en mode Production :

```
// Changer la configuration du Factory
WorkerClientFactory.changeMode(WorkerClientConfiguration configuration);
// Récupération explicite du client
WorkerClient client = WorkerClientFactory.getInstance().getClient();
```

1.1.11.3.2 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 :

1.1.12 Workspace

1.1.12.1 Introduction

1.1.12.1.1 But de cette documentation

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

1.1.12.2 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.

1.1.12.2.1 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).

1.1.12.2.2 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);
```

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.

1.2 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

1.3 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>
```

1.4 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>
```

1.5 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 2

Index & Tables

- genindex
- search