VITAM - Modèle de workflow

Version 1.4.0

VITAM

juin 26, 2018

Table des matières

CHAPITRE 1

Introduction

1.1 Avertissement

Cette documentation est un travail en cours. Elle est susceptible de changer dans les prochaines releases pour tenir compte des développements de la solution logicielle Vitam.

1.2 Objectif du document

Ce document a pour objectif de présenter les différents processus employés par la solution logicielle Vitam. Il est destiné aux administrateurs aussi bien techniques que fonctionnels, aux archivistes souhaitant une connaissance plus avancée du logiciel ainsi qu'aux développeurs.

Il explicite chaque processus (appelés également « workflow »), et pour chacun leurs tâches, traitements et actions.

Ce document comprend également du matériel additionnel pour faciliter la compréhension des processus comme des fiches récapitulatives et des schémas. Il explique également la manière dont est formée la structure des fichiers de workflow.

1.3 Description d'un processus

Un workflow est un processus composé d'étapes (macro-workflow), elles-mêmes composées d'une liste de tâches et d'actions à exécuter de manière séquentielle, une seule fois ou répétées sur une liste d'éléments (micro-workflow).

Pour chacun de ces éléments, le document décrit :

- La règle générale qui s'applique à cet élément
- Les statuts de sortie possibles (OK, KO...), avec les raisons de ces sorties et les clés associées
- Des informations complémentaires, selon le type d'élément traité

Chaque étape, chaque action peuvent avoir les statuts suivants :

- OK : le traitement associé s'est passé correctement. Le workflow continue.

- WARNING : le traitement associé a généré un avertissement (par exemple le format de l'objet est mal déclaré dans le bordereau de transfert). Le workflow continue.
- KO : le traitement associé a généré une erreur métier. Le workflow s'arrête si le modèle d'exécution est bloquant (cf. ci-dessous).
- FATAL : le traitement associé a généré une erreur technique. Le workflow se met en pause.

Un workflow peut être terminé, en cours d'exécution ou être en pause. Un workflow en pause représente le processus arrêté à une étape donnée. Chaque étape peut être mise en pause : ce choix dépend du mode de versement (le mode pas à pas marque une pause à chaque étape), ou du statut (le statut FATAL met l'étape en pause). Les workflows en pause sont visibles dans l'IHM dans l'écran « Gestion des opérations ».

Lorsque le statut FATAL survient à l'intérieur d'une étape (par exemple dans une des tâches ou un des traitements de l'étape), c'est toute l'étape qui est mise en pause. Si cette étape est rejouée, les objets déjà traités avant le fatal ne sont pas traités à nouveau : le workflow reprend exactement là où il s'était arrêté et commence par rejouer l'action sur l'objet qui a provoqué l'erreur.

Chaque action peut avoir les modèles d'exécutions suivants (toutes les étapes sont par défaut bloquantes) :

- Bloquant
  - Si une action bloquante est identifiée en erreur, le workflow est alors arrêté en erreur. Seules les actions nécessaire à l'arrêt du workflow sont alors éxecutées.
- Non bloquant
  - Si une action non bloquante est identifiée en erreur, elle seule sera en erreur et le workflow continuera normalement.

1.4 Structure d'un fichier Properties du Worflow

Les fichiers **Properties** (Par exemple *DefaultIngestWorkflow.json*) permettent de définir la structure du Workflow pour les étapes, tâches et actions réalisées dans le module d'Ingest Interne, en excluant les étapes et actions réalisées dans le module d'Ingest externe.

Un Workflow est défini en JSON avec la structure suivante :

- un bloc en-tête contenant :
  - `ID` : identifiant unique du workflow,
  - `Identifier` : clé du workflow,
  - `Name` : nom du workflow,
  - `TypeProc` : catégorie du workflow,
  - `Comment` : description du workflow ou toutes autres informations utiles concernant le workflow
- une liste d'étapes dont la structure est la suivante :
  - `workerGroupId` : identifiant de famille de Workers,
  - `stepName` : nom de l'étape, servant de clé pour identifier l'étape,
  - `Behavior` : modèle d'exécution pouvant avoir les types suivants :
    - BLOCKING : le traitement est bloqué en cas d'erreur, il est nécessaire de recommencer à la tâche en erreur. Les étapes FINALLY (définition ci-dessous) sont tout de même exécutées
    - NOBLOCKING : le traitement peut continuer malgré les éventuels erreurs ou avertissements,
    - FINALLY : le traitement correspondant est toujours exécuté, même si les étapes précédentes se sont terminées en échec
  - `Distribution` : modèle de distribution, décrit comme suit :
    - `Kind` : un type pouvant être REF (un élément unique) ou LIST (une liste d'éléments hiérarchisés) ou encore LIST_IN_FILE (liste d'éléments)

    - `Element` : l'élément de distribution indiquant l'élément unique sous forme d'URI (REF) ou la liste d'éléments en pointant vers un dossier (LIST).
    - `Type` : le type des objets traités (ObjectGroup uniquement pour le moment).
    - `statusOnEmptyDistribution` : permet dans le cas d'un traitement d'une liste vide, de surcharger le statut WARNING par un statut prédéfini.
- une liste d'Actions :
    - `ActionKey` : nom de l'action
    - `Behavior` : modèle d'exécution pouvant avoir les types suivants :
        - BLOCKING : l'action est bloquante en cas d'erreur. Les actions suivantes (de la même étape) ne seront pas éxécutées.
        - NOBLOCKING : l'action peut continuer malgré les éventuels erreurs ou avertissements.
    - `In` : liste de paramètres d'entrées :
        - `Name` : nom utilisé pour référencer cet élément entre différents handlers d'une même étape,
        - `URI` : cible comportant un schéma (WORKSPACE, MEMORY, VALUE) et un path où chaque handler peut accéder à ces valeurs via le handlerIO :
            - WORKSPACE : path indiquant le chemin relatif sur le workspace (implicitement un File),
            - MEMORY : path indiquant le nom de la clef de valeur (implicitement un objet mémoire déjà alloué par un handler précédent),
            - VALUE : path indiquant la valeur statique en entrée (implicitement une valeur String).
    - `Out` : liste de paramètres de sorties :
        - `Name` : nom utilisé pour référencer cet élément entre différents handlers d'une même étape,
        - `URI` : cible comportant un schéma (WORKSPACE, MEMORY) et un path où chaque handler peut stocker les valeurs finales via le handlerIO :
            - WORKSPACE : path indiquant le chemin relatif sur le workspace (implicitement un File local),
            - MEMORY : path indiquant le nom de la clé de valeur (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",
            "behavior": "BLOCKING",
            "out": [
              {
                "name": "mapsBDOtoOG.file",
                "uri": "WORKSPACE:Maps/OG_TO_ARCHIVE_ID_MAP.json"
              }
            ]
          }
        },
        {
          "action": {
            "actionKey": "CHECK_CONSISTENCY",
            "behavior": "NOBLOCKING",
            "in": [
              {
                "name": "mapsBDOtoOG.file",
                "uri": "WORKSPACE:Maps/OG_TO_ARCHIVE_ID_MAP.json"
              }
            ]
          }
        }
      ]
    }
  ]
}
```

CHAPITRE 2

AUDIT

2.1 Workflow de contrôle d'intégrité d'un journal sécurisé

2.1.1 Introduction

Cette section décrit le processus (workflow) de contrôle d'intégrité d'un journal sécurisé mis en place dans la solution logicielle Vitam.

Celui-ci est défini dans le fichier "DefaultCheckTraceability.json" (situé ici : sources/processing/processing-management/src/main/resources/workflows).

2.1.2 Processus de contrôle d'intégrité d'un journal sécurisé (vision métier)

Le processus de contrôle d'intégrité débute lorsqu'un identifiant d'opération de sécurisation des journaux d'opération, des journaux du cycles de vie, ou du journal des écritures est soumis au service de contrôle d'intégrité des journaux sécurisés. Le service permet de récupérer le journal sécurisé, d'en extraire son contenu et de valider que son contenu n'a pas été altéré.

Pour cela, il calcule un arbre de Merkle à partir des journaux d'opérations que contient le journal sécurisé, puis en calcule un second à partir des journaux correspondants disponibles dans la solution logicielle Vitam. Une comparaison est ensuite effectuée entre ces deux arbres et celui contenu dans les métadonnées du journal sécurisé.

Ensuite, dans une dernière étape, le tampon d'horodatage est vérifié et validé.

2.1.3 Processus de préparation de la vérification des journaux sécurisés (STP_PREPARE_TRACEABILITY_CHECK)

2.1.3.1 Préparation de la vérification des journaux sécurisés PREPARE_TRACEABILITY_CHECK (PrepareTraceabilityCheckProcessActionHandler.java)

- **Règle** : vérification que l'opération donnée en entrée est de type TRACEABILITY. Récupération du zip associé à cette opération et extraction de son contenu.
- **Type** : bloquant
- **Statuts :**
    - OK : l'opération donnée en entrée est une opération de type TRACEABILITY, le zip a été trouvé et son contenu extrait (PREPARE_TRACEABILITY_CHECK.OK=Succès de la préparation de la vérification des journaux sécurisés)
    - KO : l'opération donnée en entrée n'est pas une opération de type TRACEABILITY (PREPARE_TRACEABILITY_CHECK.KO=Échec de la préparation de la vérification des journaux sécurisés)
    - FATAL : une erreur fatale est survenue lors de la préparation du processus de vérification (PREPARE_TRACEABILITY_CHECK.FATAL=Erreur fatale lors de la préparation de la vérification des journaux sécurisés)

2.1.4 Processus de vérification de l'arbre de Merkle (STP_MERKLE_TREE)

2.1.4.1 Vérification de l'arbre de MERKLECHECK_MERKLE_TREE (VerifyMerkleTreeActionHandler.java)

- **Règle** : Recalcul de l'arbre de Merkle des journaux contenus dans le journal sécurisé, calcul d'un autre arbre à partir des journaux indexés correspondants et vérification que tous deux correspondent à celui stocké dans les métadonnées du journal sécurisé
- **Type** : bloquant
- **Statuts :**
    - OK : les arbres de Merkle correspondent (CHECK_MERKLE_TREE.OK=Succès de la vérification de l'arbre de MERKLE)
    - KO : les arbres de Merkle ne correspondent pas (CHECK_Merkle_TREE.KO=Échec de la vérification de l'arbre de MERKLE)
    - FATAL : une erreur fatale est survenue lors de la vérification des arbres de Merkle (CHECK_MERKLE_TREE.FATAL=Erreur fatale lors de la vérification de l'arbre de MERKLE)

La tâche contient les traitements suivants

Comparaison de l'arbre de MERKLE avec le Hash enregistré

- **Règle** : Vérification que l'arbre de Merkle calculé à partir des journaux contenus dans le journal sécurisé est identique à celui stocké dans les métadonnées du journal sécurisé
- **Type** : bloquant
- **Statuts** :
    - OK : l'arbre de Merkle des journaux contenus dans le journal sécurisé correspond à celui stocké dans les métadonnées du journal sécurisé (CHECK_MERKLE_TREE.COMPARE_MERKLE_HASH_WITH_SAVED_HASH.OK=Succès de la comparaison de l'arbre de MERKLE avec le Hash enregistré)

    - KO : l'arbre de Merkle des journaux contenus dans le journal sécurisé ne correspond pas à celui stocké dans les métadonnées du journal sécurisé (CHECK_MERKLE_TREE.COMPARE_MERKLE_HASH_WITH_SAVED_HASH.KO=Échec de la comparaison de l'arbre de MERKLE avec le Hash enregistré)
    - FATAL : une erreur fatale est survenue lors de la comparaison de l'arbre de MERKLE avec le Hash enregistré (CHECK_MERKLE_TREE.COMPARE_MERKLE_HASH_WITH_SAVED_HASH.FATAL=Erreur fatale lors de la comparaison de l'arbre de MERKLE avec le Hash enregistré)

- Comparaison de l'arbre de MERKLE avec le Hash indexé
  - **Règle** : Vérification que l'arbre de Merkle calculé à partir des journaux indexés est identique à celui stocké dans les métadonnées du journal sécurisé
  - **Type** : bloquant
  - **Statuts** :
    - OK : l'arbre de Merkle des journaux indexés correspond à celui stocké dans les métadonnées du journal sécurisé (CHECK_MERKLE_TREE.COMPARE_MERKLE_HASH_WITH_INDEXED_HASH.OK=Succès de la comparaison de l'arbre de MERKLE avec le Hash indexé)
    - KO : l'arbre de Merkle des journaux indexés ne correspond pas à celui stocké dans les métadonnées du journal sécurisé (CHECK_MERKLE_TREE.COMPARE_MERKLE_HASH_WITH_INDEXED_HASH.KO=Échec de la comparaison de l'arbre de MERKLE avec le Hash indexé)
    - FATAL : une erreur fatale est survenue lors de la comparaison l'arbre de Merkle des journaux indexés et de celui stocké dans les métadonnées du journal sécurisé (CHECK_MERKLE_TREE.COMPARE_MERKLE_HASH_WITH_INDEXED_HASH.FATAL=Erreur fatale lors de la comparaison de l'arbre de MERKLE avec le Hash indexé)

2.1.5 Procesus de vérification de l'horodatage (STP_VERIFY_STAMP)

2.1.5.1 Vérification et validation du tampon d'horodatage VERIFY_TIMESTAMP (VerifyTimeStampActionHandler.java)

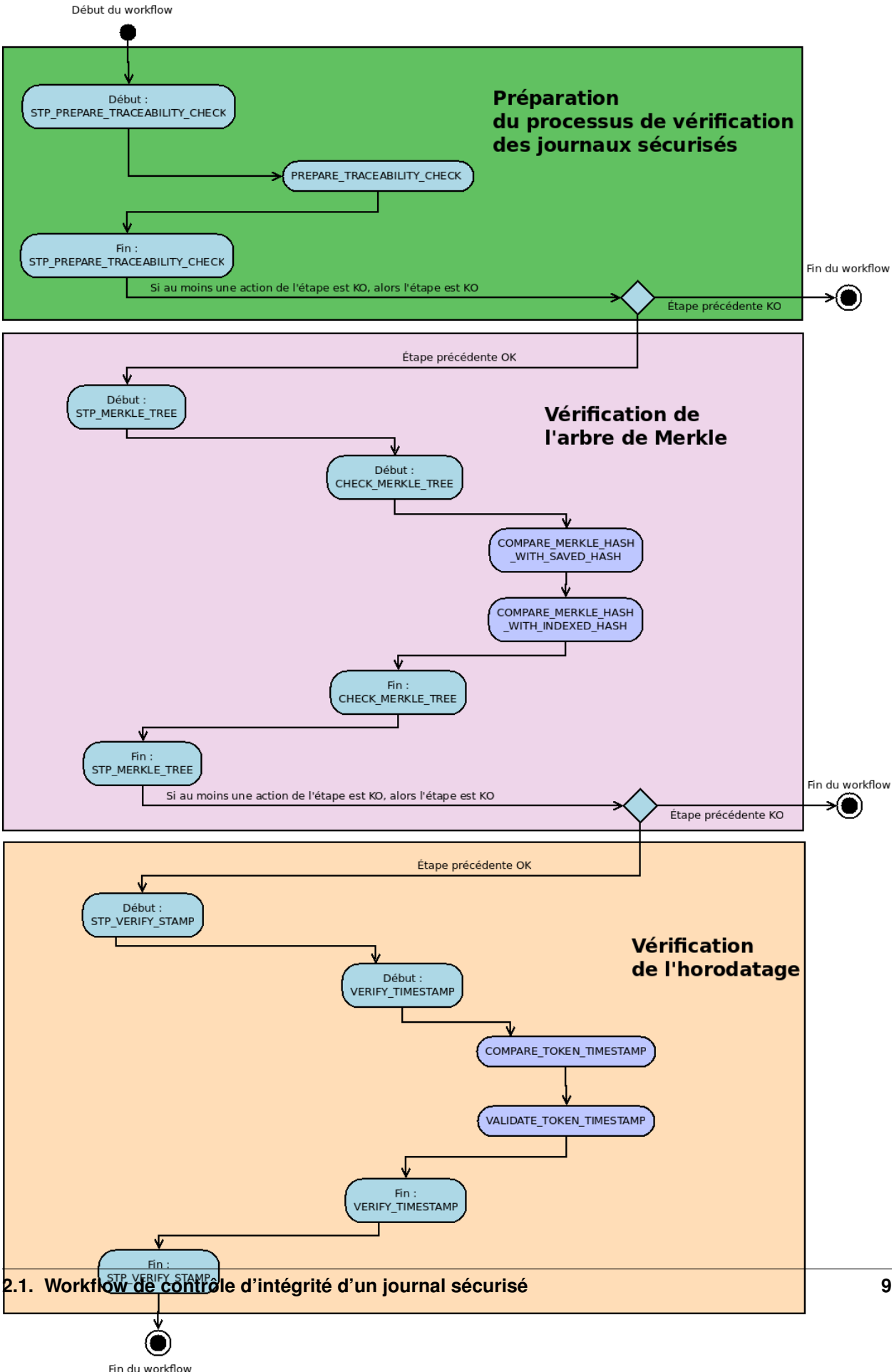
- **Règle** : Vérification et validation du tampon d'horodatage.
- **Type** : bloquant
- **Statuts** :
  - OK : le tampon d'horadatage est correct (VERIFY_TIMESTAMP.OK=Succès de la vérification de l'horodatage)
  - KO : le tampon d'horadatage est incorrect (VERIFY_TIMESTAMP.KO=Échec de la vérification de l'horodatage)
  - FATAL : une erreur fatale est survenue lors de la vérification du tampon d'horodatage (VERIFY_TIMESTAMP.FATAL=Erreur fatale lors de la vérification de l'horodatage)

La tâche contient les traitements suivants

- Comparaison du tampon du fichier (token.tsp) par rapport au tampon enregistré dans le logbook (COMPARE_TOKEN_TIMESTAMP)
  - **Règle** : Vérification que le tampon enregistré dans la collection logbookOperation est le même que celui présent dans le fichier zip généré
  - **Type** : bloquant
  - **Status** :

        - OK : les tampons sont identiques (VERIFY_TIMESTAMP.COMPARE_TOKEN_TIMESTAMP.OK=Succès de la comparaison des tampons d'horodatage)
        - KO : les tampons sont différents (VERIFY_TIMESTAMP.COMPARE_TOKEN_TIMESTAMP.KO=Échec de la comparaison des tampons d'horadatage)
        - FATAL : erreur fatale lors de la vérification des tampons (VERIFY_TIMESTAMP.COMPARE_TOKEN_TIMESTAMP.FATAL=Erreur fatale lors de la comparaison des tampons d'horadatage)
- Validation du tampon d'horodatage (VALIDATE_TOKEN_TIMESTAMP)
    - **Règle** : Vérification cryptographique du tampon et vérification de la chaîne de certification
    - **Type** : bloquant
    - **Status :**
        - OK : le tampon est validé (VERIFY_TIMESTAMP.VALIDATE_TOKEN_TIMESTAMP.OK=Succès de la validation du tampon d'horodatage)
        - KO : le tampon est invalidé (VERIFY_TIMESTAMP.VALIDATE_TOKEN_TIMESTAMP.KO=Échec de la validation du tampon d'horodatage)
        - FATAL : erreur fatale lors de la validation du tampon d'horodatage (VERIFY_TIMESTAMP.VALIDATE_TOKEN_TIMESTAMP.FATAL=Erreur fatale lors de la validation du tampon d'"“horodatage)

D'une façon synthétique, le workflow est décrit de cette façon :


Début du workflow
Début :
STP_PREPARE_TRACEABILITY_CHECK
Préparation
du processus de vérification
des journaux sécurisés
PREPARE_TRACEABILITY_CHECK
Fin :
STP_PREPARE_TRACEABILITY_CHECK
Si au moins une action de l'étape est KO, alors l'étape est KO
Étape précédente KO
Fin du workflow
Étape précédente OK
Début :
STP_MERKLE_TREE
Vérification de
l'arbre de Merkle
Début :
CHECK_MERKLE_TREE
COMPARE_MERKLE_HASH
_WITH_SAVED_HASH
COMPARE_MERKLE_HASH
_WITH_INDEXED_HASH
Fin :
CHECK_MERKLE_TREE
Fin :
STP_MERKLE_TREE
Si au moins une action de l'étape est KO, alors l'étape est KO
Étape précédente KO
Fin du workflow
Étape précédente OK
Début :
STP_VERIFY_STAMP
Vérification
de l'horodatage
Début :
VERIFY_TIMESTAMP
COMPARE_TOKEN_TIMESTAMP
VALIDATE_TOKEN_TIMESTAMP
Fin :
VERIFY_TIMESTAMP
Fin :
STP_VERIFY_STAMP
Fin du workflow

2.2 Workflow de l'audit de l'existence et de l'intégrité des fichiers

2.2.1 Introduction

Cette section décrit le processus (workflow) d'audit de l'existence des fichiers mis en place dans la solution logicielle Vitam.

Celui-ci est défini dans le fichier « DefaultAuditWorkflow.json" (situé ici : sources/processing/processing-management/src/main/resources/workflows).

2.2.2 Processus d'audit d'existence des fichiers (vision métier)

Le processus d'audit prend comme point d'entrée l'identifiant d'un tenant ou l'identifiant d'un service producteur. Il est possible de lancer un audit de l'existence des fichiers uniquement, ou de lancer un audit vérifiant l'existence et l'intégrité des fichiers en même temps.

Pour chaque objet du tenant choisi ou chaque objet appartenant au service producteur, l'audit va vérifier :

- Que la liste des offres de stockage définie dans le groupe d'objets est bien la même que celle définie dans la stratégie de stockage
- Que tous les fichiers correspondant aux objets existent sur les offres déclarées, dans un nombre de copie spécifié via la stratégie de stockage

2.2.3 Processus d'audit d'intégrité des fichiers (vision métier)

Si l'audit d'intégrité des objets est lancé, le processus va également vérifier que les empreintes des objets stockés en base de données sont bien les mêmes que les empreintes fournies par les espaces de stockage, alors recalculées à la demande de l'audit.

Dans une première étape technique, il prépare la liste des groupes d'objets à auditer afin de paralléliser la tâche. Dans un second temps, il effectue la vérification elle même. Enfin, il sécurise les journaux de cycle de vie qui ont été modifiés.

2.2.4 Processus de préparation de l'audit (STP_PREPARE_AUDIT)

2.2.4.1 Création de la liste des groupes d'objets LIST_OBJECTGROUP_ID (PrepareAuditActionHandler.java)

- **Règle** : Création de la liste des groupes d'objets à auditer
- **Type** : bloquant
- **Statuts :**
    - OK : la liste a été créée avec succès (LIST_OBJECTGROUP_ID.OK=Succès de la création de la liste des groupes d'objets à auditer)
    - FATAL : Une erreur fatale est survenue lors de la création de la liste (LIST_OBJECTGROUP_ID.FATAL=Erreur fatale lors de la création de la liste des groupes d'objets à auditer)

2.2.5 Processus d'éxecution de l'audit (STP_AUDIT)

2.2.5.1 Audit de la vérification des objets AUDIT_CHECK_OBJECT (AuditCheckObjectPlugin.java)

- **Règle** : Tâche technique pour organiser et lancer l'action d'audit de la vérification des objets. A la fin de l'audit de chaque groupe d'objets en KO, la mise à jour en base de son journal du cycle de vie est faite.
- **Type** : bloquant
- **Statuts :**
    - OK : l'action d'audit de la vérification des objets s'est terminée en OK (AUDIT_CHECK_OBJECT.OK=Succès de l'audit de la vérification des objets)
    - KO : l'action d'audit de la vérification des objets s'est terminée en KO (AUDIT_CHECK_OBJECT.KO=Échec de l'audit de la vérification des objets : au moins un objet demandé n'existe pas ou des stratégies de stockage sont incohérentes avec les offres déclarées)
    - FATAL : une erreur fatale est survenue lors du lancement de l'action d'audit (AUDIT_CHECK_OBJECT.FATAL=Erreur fatale lors de l'audit de la vérification des objets)

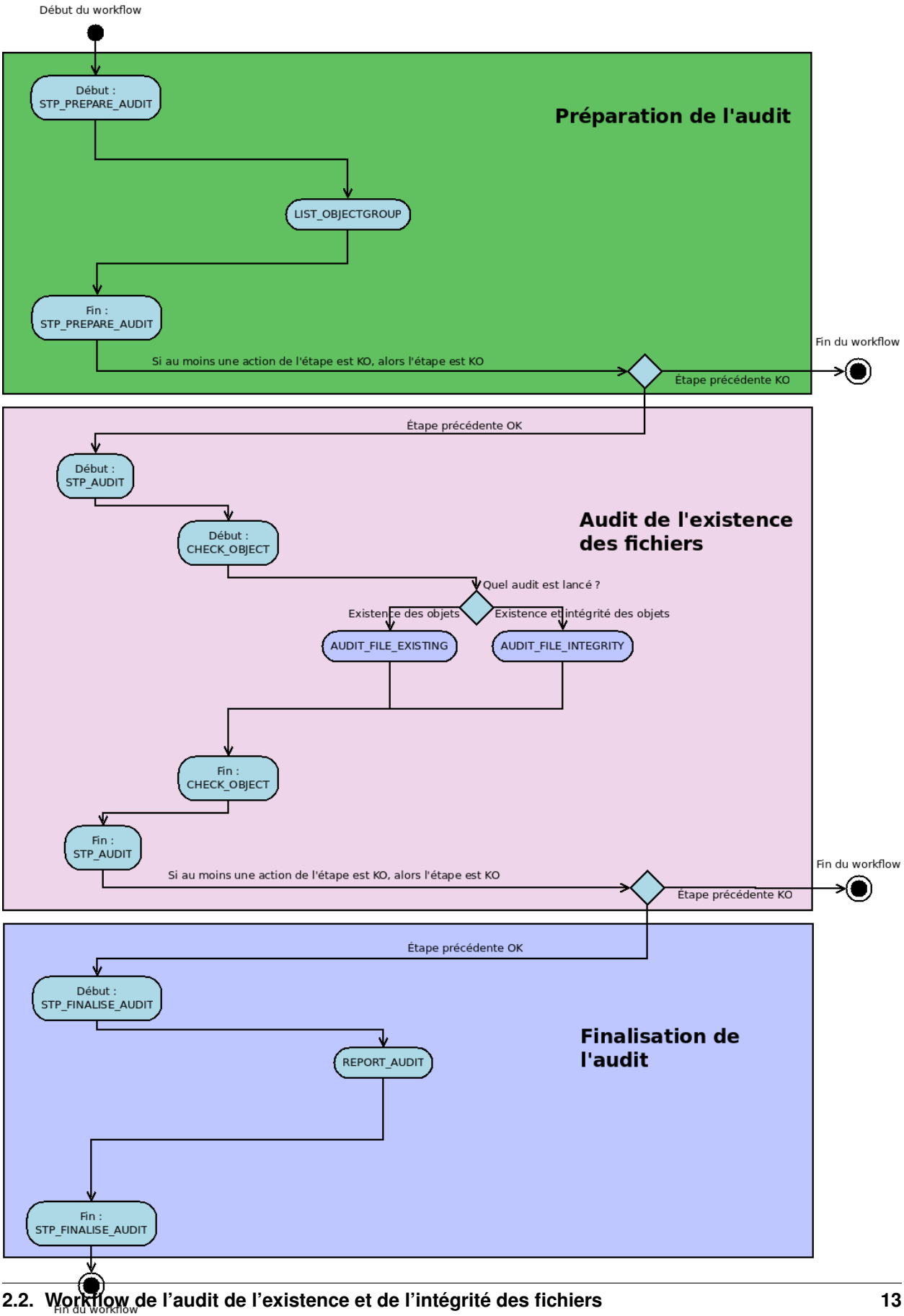
2.2.5.2 Audit de l'existence des objets AUDIT_FILE_EXISTING (CheckExistenceObjectPlugin.java)

- **Règle** [Audit de l'existence des objets permettant la vérification de chaque groupe d'objets audités]
    - La stratégie de stockage du groupe d'objets est conforme à celle du moteur de stockage
    - Les fichiers correspondant aux objets, déclarés dans le groupe d'objets, existent bien sous le même nom dans les offres de stockage
- **Type** : bloquant
- **Statuts :**
    - OK : tous les objets de tous les groupes d'objets audités existent bien sur les offres de stockage (AUDIT_CHECK_OBJECT.AUDIT_FILE_EXISTING.OK=Succès de l'audit de l'existence de fichiers)
    - KO : au moins un objet n'existe pas pour au moins un groupe objets (AUDIT_CHECK_OBJECT.AUDIT_FILE_EXISTING.KO=Echec de l'audit de l'existence de fichiers)
    - Warning : il n'y a aucun objet à auditer (cas par exemple d'un producteur sans objets) (AUDIT_CHECK_OBJECT.AUDIT_FILE_EXISTING.WARNING=Avertissement lors de l'audit de l'existence des objets : au moins un groupe d'objets n'a pas d'objet binaire à vérifier)
    - FATAL : une erreur fatale est survenue lors de l'audit de l'existence des objets (AUDIT_CHECK_OBJECT.AUDIT_FILE_EXISTING.FATAL=Erreur fatale lors de l'audit de l'existence des objets)

2.2.5.3 Audit de l'intégrité des objets AUDIT_CHECK_OBJECT.AUDIT_FILE_INTEGRITY (CheckIntegrityObjectPlugin.java)

- **Règle** [Audit de l'intégrité des objets permettant la vérification de chaque groupe d'objets audités]
    - L'objet existe bien (voir « AUDIT_FILE_EXISTING »)
    - L'empreinte de l'objet enregistrée en base de données est la même pour chaque objet que celle obtenue par l'offre de stockage, recalculée à la demande de l'audit
- **Type** : bloquant
- **Statuts :**
    - OK : tous les objets de tous les groupes d'objet audités existent bien sur les offres de stockage et leurs empreintes sont identiques entre celles enregistrées en base de données et celles recalculées par les offres de stockage (AUDIT_CHECK_OBJECT.AUDIT_FILE_INTEGRITY.OK=Succès de l'audit de l'existence et de l'intégrité des objets)

- KO : au moins un objet n'existe pas pour au moins un objet (AUDIT_CHECK_OBJECT.AUDIT_FILE_INTEGRITY.KO=Échec de l'audit de l'existence et de l'intégrité des objets)
- Warning : il n'y a aucun objet à auditer (cas par exemple d'un producteur sans objets) (AUDIT_CHECK_OBJECT.AUDIT_FILE_INTEGRITY.WARNING=Avertissement lors de l'existence et de l'intégrité des objets)
- FATAL : une erreur fatale est survenue lors de l'audit de l'existence des fichiers (AUDIT_CHECK_OBJECT.AUDIT_FILE_INTEGRITY.FATAL=Erreur fatale lors de l'existence et de l'intégrité des objets)


Début du workflow
Début : STP_PREPARE_AUDIT
Préparation de l'audit
LIST_OBJECTGROUP
Fin : STP_PREPARE_AUDIT
Si au moins une action de l'étape est KO, alors l'étape est KO
Étape précédente KO
Fin du workflow
Étape précédente OK
Début : STP_AUDIT
Audit de l'existence des fichiers
Début : CHECK_OBJECT
Quel audit est lancé ?
Existence des objets
Existence et intégrité des objets
AUDIT_FILE_EXISTING
AUDIT_FILE_INTEGRITY
Fin : CHECK_OBJECT
Fin : STP_AUDIT
Si au moins une action de l'étape est KO, alors l'étape est KO
Étape précédente KO
Fin du workflow
Étape précédente OK
Début : STP_FINALISE_AUDIT
Finalisation de l'audit
REPORT_AUDIT
Fin : STP_FINALISE_AUDIT
Fin du workflow

2.2.6 Processus de finalisation de l'audit (STP_FINALISE_AUDIT)

2.2.6.1 Notification de la fin d'audit REPORT_AUDIT (GenerateAuditReportActionHandler.java)

- **Règle** : génération du rapport d'audit
- **Type** : bloquant
- **Statuts :**
  - OK : le rapport a été créé avec succès (REPORT_AUDIT.OK=Succès de la notification de la fin de l'audit)
  - FATAL : Une erreur fatale est survenue lors de la création du rapport d'audit (REPORT_AUDIT.OK.FATAL=Erreur fatale lors de la notification de la fin de l'audit)

2.3 Rapport d'audit

Le rapport d'audit est un fichier JSON généré par la solution logicielle Vitam lorsqu'une opération d'audit se termine. Cette section décrit la manière dont ce rapport est structuré.

2.3.1 Exemple de JSON : rapport d'audit KO


```
{
   "tenant": 2,
   "auditOperationId": "aeeaaaaaakgtg6rzaahd4ak6od5brxaaaaaq",
   "auditType": "tenant",
   "objectId": "2",
   "DateTime": "2017-09-11T12:46:32.164",
   "Status": "KO",
   "outMessage": "Echec de l'audit",
   "LastEvent": "AUDIT_FILE_EXISTING",
   "source": [
      {
         "_tenant": "2",
         "OriginatingAgency": "FRAN_NP_009913",
         "evIdProc": "aedqaaaaakeuctkoabjgkak6lowhh6yaaaaq"
      },
      {
         "_tenant": "2",
         "OriginatingAgency": "RATP",
         "evIdProc": "aedqaaaaakhu4m3aaaz2aak6loy4jxqaaaaq"
      },
      {
         "_tenant": "2",
         "OriginatingAgency": "RATP",
         "evIdProc": "aedqaaaaakhu4m3aaaz2aak6lo2shsiaaaaq"
      },
      {
         "_tenant": "2",
         "OriginatingAgency": "OBJ_KO",
         "evIdProc": "aedqaaaaakhfetkwabvlcak6lso7c7aaaaaq"
      },
      {
         "_tenant": "2",
         "OriginatingAgency": "PROD_AUDIT_KO_2OBJ_1GO",
```


(suite sur la page suivante)

(suite de la page précédente)


```
            "evIdProc": "aedqaaaaakhfetkwabvlcak6lsvp75aaaaaq"
        },
        {
            "_tenant": "2",
            "OriginatingAgency": "PROD_OBJ_PHYSIQUE",
            "evIdProc": "aedqaaaaakfuavsrab2diak6mdzyw6aaaaaq"
        },
        {
            "_tenant": "2",
            "OriginatingAgency": "SP_SANS_OBJ",
            "evIdProc": "aedqaaaaakfuavsrab2diak6mdz7rraaaaaq"
        }
    ],
    "auditKO": [
        {
            "IdOp": "aeeaaaaaakgtg6rzaahd4ak6od5brxaaaaaq",
            "IdGOT": "aebaaaaaaifemvtkabtmsak6lso7pdiaaaaq",
            "IdObj": "aeaaaaaaaafemvtkabtmsak6lso7pcyaaaaq",
            "Usage": "BinaryMaster_1",
            "OriginatingAgency": "OBJ_KO",
            "OutDetail": "LFC.AUDIT_FILE_EXISTING.KO"
        },
        {
            "IdOp": "aeeaaaaaakgtg6rzaahd4ak6od5brxaaaaaq",
            "IdGOT": "aebaaaaaaifemvtkabtmsak6lsvqfkiaaaba",
            "IdObj": "aeaaaaaaaafemvtkabtmsak6lsvqflqaaaaq",
            "Usage": "TextContent_1",
            "OriginatingAgency": "PROD_AUDIT_KO_2OBJ_1GO",
            "OutDetail": "LFC.AUDIT_FILE_EXISTING.KO"
        },
        {
            "IdOp": "aeeaaaaaakgtg6rzaahd4ak6od5brxaaaaaq",
            "IdGOT": "aebaaaaaaifemvtkabtmsak6lsvqfjiaaaaq",
            "IdObj": "aeaaaaaaaafemvtkabtmsak6lsvqfjaaaaaq",
            "Usage": "BinaryMaster_1",
            "OriginatingAgency": "PROD_AUDIT_KO_2OBJ_1GO",
            "OutDetail": "LFC.AUDIT_FILE_EXISTING.KO"
        }
    ],
    "auditWarning": [
        "SP_SANS_OBJ"
    ]
}
```


2.3.2 Partie « Master »

La partie « master », c'est à dire le bloc à la racine du rapport (sans indentation) est composé des champs suivants :

- « tenant » : le tenant sur lequel l'opération d'audit a été lancée
- « auditOperationId » : l'identifiant de l'opération d'audit
- « auditType » : l'élément sur lequel l'audit a été lancé. Celui ci peut être par « tenant », ou par « originatingagency »
- « objectId » : l'identifiant de l'élément (tenant ou service producteur)
- « DateTime » : la date du rapport

- « Status » : la statut final du rapport, OK (l'audit n'a pas détecté d'anomalie), Warning (l'audit a détecté quelque chose de singulier qui n'a pas été considéré comme une anomalie), KO (l'audit a détecté une anomalie)
- « outMessage » : le message final de l'audit, repris du journal des opérations
- « LastEvent » : la clé correspondant au type d'audit. Par exemple pour l'audit de l'existence des fichiers il s'agit de « AUDIT_FILE_EXISTING »

Mais aussi : - « source » : la liste des opérations auditées - « auditKO » : la liste des anomalies détectées qui ont provoqué le KO de l'audit - « auditWarning » : la liste des éléments singuliers détectés qui ont provoqué un warning de l'audit

2.3.3 Liste des opérations auditées (« source »)

La liste des opérations auditées est une liste d'identifiant d'opérations d'ingest. Il s'agit des opérations à l'origine de la création des groupes d'objets qui ont été audités. Chaque groupe n'a par nature qu'une et une seule opération à l'origine de sa création. En partant de ces opérations, il est donc possible de retrouver l'ensemble des groupes d'objets qui ont été audités.

Au travers de ces identifiants d'opérations, cette liste recense exhaustivement les groupes d'objets audités et ne présume en rien le succès ou l'échec de l'audit par rapport à ceux-ci.

Cette partie est construite autour des champs suivants :

- « #tenant » : identifiant du tenant sur lequel l'opération s'est déroulée
- « OriginatingAgency » : identifiant du service producteur relatif à cette opération
- « evIdProc » : identifiant de l'opération étant à l'origine de la création du groupe d'objets audité

2.3.4 Liste des anomalies détectées générant un KO (« auditKO »)

Cette liste détaille l'ensemble des objets qui ont rencontré un KO lors de l'audit KO. Chaque objet possède son bloc, ayant les champs suivants :

- « IdOp » : identifiant de l'opération étant à l'origine de la création du groupe d'objets auquel appartient l'objet KO audité
- « IdGOT » : identifiant du groupe d'objets audité, possédant l'objet KO
- « IdObj » : identifiant de l'objet KO
- « Usage » : usage de l'objet KO dans son groupe d'objets
- « OriginatingAgency » : service producteur de référence de l'objet
- « OutDetail » : clé correspondant à l'audit qui a déclenché le KO, reprise du journal des opérations. Par exemple pour un audit de l'existence des fichiers, la clé est « LFC.AUDIT_FILE_EXISTING.KO »

2.3.5 Liste des éléments singuliers générant un avertissement (« auditWarning »)

Cette liste décrit les identifiants des services producteurs ayant généré un avertissement. Dans le cas de l'audit de l'existence des fichiers, une alerte correspond au fait qu'un service producteur n'a aucun objet à auditer. Cette liste est donc l'ensemble des services producteurs concernés par l'audit mais dont il n'existe aucun objet à auditer.

2.4 Workflow de l'audit de cohérence des fichiers

2.4.1 Introduction

Cette section décrit le processus (workflow) d'audit de cohérence des fichiers mis en place dans la solution logicielle Vitam.

Celui-ci est défini dans le fichier « EvidenceAuditWorkflow.json" (situé ici : sources/processing/processing-management/src/main/resources/workflows).

2.4.2 Processus d'audit de cohérence des fichiers (vision métier)

Le processus d'audit de cohérence permet de vérifier la cohérence entre les signatures calculées pour chaque objet, en comparant celle présente dans le journal sécurisé, avec celle présente dans la base de donnée, et celle de l'offre de stockage.

L'audit s'applique au niveau des unités archivistiques, des objets, et des groupes d'objets.

2.4.3 Processus de préparation de l'audit (STP_EVIDENCE_AUDIT_PREPARE)

2.4.3.1 Création de la liste à auditer EVIDENCE_AUDIT_LIST_OBJECT (EvidenceAuditPrepare.java)

- **Règle** : Création de la liste à auditer
- **Type** : bloquant
- **Statuts :**
    - OK : la liste a été créée avec succès (EVIDENCE_AUDIT_LIST_OBJECT.OK=Création de la liste à auditer)
    - KO : Echec de la création de la liste à auditer (EVIDENCE_AUDIT_LIST_OBJECT.KO=Echec lors de la création de la liste à auditer)
    - FATAL : Une erreur fatale est survenue lors de la création de la liste (EVIDENCE_AUDIT_LIST_OBJECT.FTAL=Erreur fatale lors de la création de la liste à auditer)

2.4.4 Processus de récupération des données de la base (STP_EVIDENCE_AUDIT_CHECK_DATABASE)

2.4.4.1 Récupération des données dans la base de donnée EVIDENCE_AUDIT_CHECK_DATABASE (EvidenceAuditDatabaseCheck.java)

- **Règle** : Tâche consistant à récupérer les informations nécéssaires à l'audit dans la base de données.
- **Type** : bloquant
- **Statuts** :
    - OK : La récupération des données dans la base de données est un succès (EVIDENCE_AUDIT_CHECK_DATABASE.OK=Succès de la récupération des données dans la base de données)
    - KO : La récupération des données dans la base de donnée est un échec (EVIDENCE_AUDIT_CHECK_DATABASE.KO=Echec de la récupération des données dans la base de données)

- FATAL : Une erreur fatale est survenue dans la récupération des données dans la base de données (EVIDENCE_AUDIT_CHECK_DATABASE.FATAL=Erreur fatale lors de la récupération des données dans la base de données)
- WARNING : Avertissement lors de la récupération des données dans la base de données (EEVIDENCE_AUDIT_CHECK_DATABASE.WARNING=Avertissement lors de la récupération des données dans la base de données)

2.4.5 Processus de préparation des signatures à partir des fichiers sécurisés (STP_EVIDENCE_AUDIT_LIST_SECURED_FILES_TO_DOWNLOAD)

2.4.5.1 Préparation de la liste des signatures dans les fichiers sécurisés EVIDENCE_AUDIT_LIST_SECURED_FILES_TO_DOWNLOAD (EvidenceAuditListSecuredFiles.java)

- **Règle** : Tâche consistant à préparer la liste des signatures des objets, groupes d'objets ou unités archivistiques archivées, dans les fichiers sécurisés.
- **Type** : bloquant
- **Statuts** :
  - OK : La préparation de la liste des signatures dans les fichiers sécurisés est un succès (EVIDENCE_AUDIT_LIST_SECURED_FILES_TO_DOWNLOAD.OK=Succès de la préparation de la liste des signatures dans les fichiers sécurisés)
  - KO : La préparation de la liste des signatures dans les fichiers sécurisés est un échec (EVIDENCE_AUDIT_LIST_SECURED_FILES_TO_DOWNLOAD.KO=Echec de la préparation de la liste des signatures dans les fichiers sécurisés)
  - WARNING : Avertissement lors de la préparation de la liste des signatures (EVIDENCE_AUDIT_LIST_SECURED_FILES_TO_DOWNLOAD.WARNING=Avertissement lors de la préparation de la liste des signatures dans les fichiers sécurisés)
  - FATAL : une erreur fatale est survenue lors de la préparation de la liste des signatures dans les fichiers sécurisés (EVIDENCE_AUDIT_LIST_SECURED_FILES_TO_DOWNLOAD.FATAL=Erreur fatale lors de la préparation de la liste des signatures dans les fichiers sécurisés)

2.4.6 Processus d'extraction des signatures à partir des fichiers sécurisés (STP_EVIDENCE_AUDIT_EXTRACT_ZIP_FILE)

2.4.6.1 Extraction des signatures à partir des fichiers sécurisés EVIDENCE_AUDIT_EXTRACT_ZIP_FILE (EvidenceAuditExtractFromZip.java)

- **Règle** : Tâche consistant à extraire les signatures des objets, groupes d'objets ou unités archivistiques archivées, dans les fichiers sécurisés.
- **Type** : bloquant
- **Statuts** :
  - OK : L'extraction des signatures à partir des fichiers sécurisés est un succès (EVIDENCE_AUDIT_EXTRACT_ZIP_FILE.OK=Succès de l'extraction des signatures à partir des fichiers sécurisés)
  - KO : L'extraction des signatures à partir des fichiers sécurisés est un échec (EVIDENCE_AUDIT_EXTRACT_ZIP_FILE.KO=Echec de l'extraction des signatures à partir des fichiers sécurisés)

- WARNING : Avertissement lors de l'extraction des signatures à partir des fichiers sécurisés (STP_EVIDENCE_AUDIT_EXTRACT_ZIP_FILE.WARNING=Avertissement lors de l'extraction des signatures à partir des fichiers sécurisés)
- FATAL : Une erreur fatale est survenue lors de l'extraction des signatures à partir des fichiers sécurisés (EVIDENCE_AUDIT_EXTRACT_ZIP_FILE.FATAL=Erreur fatale lors de l'extraction des signatures à partir des fichiers sécurisés)

2.4.7 Processus de préparation des rapports pour chaque objet, groupe d'objets ou unité audité (STP_EVIDENCE_AUDIT_PREPARE_GENERATE_REPORTS)

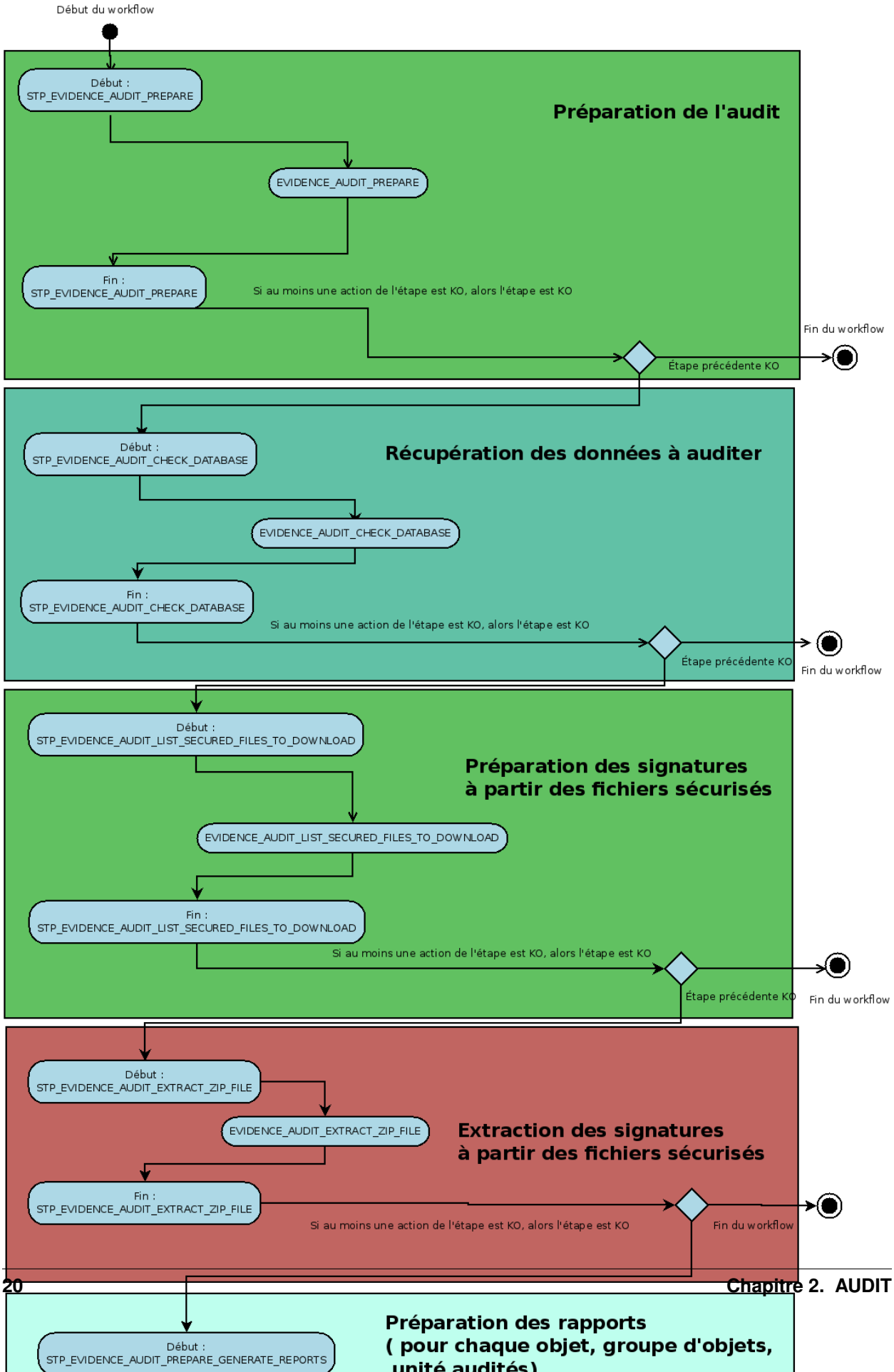
2.4.7.1 Création du rapport pour chaque unité archivistique ou objet ou groupe d'objets EVIDENCE_AUDIT_PREPARE_GENERATE_REPORTS (EvidenceAuditGenerateReports.java)

- **Règle** : Tâche consistant à créer le rapport pour chaque unité archivistique, objet ou groupe d'objets audité
- **Type** : bloquant
- **Statuts :**
    - OK : La création du rapport pour chaque unité archivistique ou objet ou groupe d'objets est un succès (EVIDENCE_AUDIT_PREPARE_GENERATE_REPORTS.OK=Succès de la création du rapport pour chaque unité archivistique ou objet ou groupe d'objets)
    - KO : La création du rapport pour chaque unité archivistique ou objet ou groupe d'objets est un échec (EVIDENCE_AUDIT_PREPARE_GENERATE_REPORTS.KO=Echec de la création du rapport pour chaque unité archivistique ou objet ou groupe d'objets)
    - FATAL : une erreur fatale est survenue de la création du rapport pour chaque unité archivistique ou objet ou groupe d'objets (EVIDENCE_AUDIT_PREPARE_GENERATE_REPORTS.FATAL=Erreur fatale lors de la création du rapport pour chaque unité archivistique ou objet ou groupe d'objets)
    - WARNING : Avertissement lors de la création du rapport pour chaque unité archivistique ou objet ou groupe d'objets (EVIDENCE_AUDIT_PREPARE_GENERATE_REPORTS.WARNING=Avertissement lors de la création du rapport pour chaque unité archivistique ou objet ou groupe d'objets)

2.4.8 Processus de finalisation de l'audit et génération du rapport final (STP_EVIDENCE_AUDIT_FINALIZE)

2.4.8.1 Création du rapport de l'audit de cohérence EVIDENCE_AUDIT_FINALIZE (EvidenceAuditFinalize.java)

- **Règle** : Tâche consistant à créer le rapport permettant de comparer les signatures extraites des fichiers sécurisés avec les données de la base de données et de l'offre de stockage.
- **Type** : bloquant
- **Statuts :**
    - OK : La création du rapport d'audit de cohérence est un succès (EVIDENCE_AUDIT_FINALIZE.OK=Succès de la création du rapport de l'audit de cohérence)
    - KO : La création du rapport d'audit de cohérence est un échec (EVIDENCE_AUDIT_FINALIZE.KO=Echec de la création du rapport de l'audit de cohérence)
    - FATAL : une erreur fatale est survenue lors de la création du rapport d'audit de cohérence) (EVIDENCE_AUDIT_FINALIZE.FATAL=Erreur fatale lors de la création du rapport d'audit de cohérence)


Début du workflow
Début :
STP_EVIDENCE_AUDIT_PREPARE
Préparation de l'audit
EVIDENCE_AUDIT_PREPARE
Fin :
STP_EVIDENCE_AUDIT_PREPARE
Si au moins une action de l'étape est KO, alors l'étape est KO
Étape précédente KO
Fin du workflow
Début :
STP_EVIDENCE_AUDIT_CHECK_DATABASE
Récupération des données à auditer
EVIDENCE_AUDIT_CHECK_DATABASE
Fin :
STP_EVIDENCE_AUDIT_CHECK_DATABASE
Si au moins une action de l'étape est KO, alors l'étape est KO
Étape précédente KO
Fin du workflow
Début :
STP_EVIDENCE_AUDIT_LIST_SECURED_FILES_TO_DOWNLOAD
Préparation des signatures
à partir des fichiers sécurisés
EVIDENCE_AUDIT_LIST_SECURED_FILES_TO_DOWNLOAD
Fin :
STP_EVIDENCE_AUDIT_LIST_SECURED_FILES_TO_DOWNLOAD
Si au moins une action de l'étape est KO, alors l'étape est KO
Étape précédente KO
Fin du workflow
Début :
STP_EVIDENCE_AUDIT_EXTRACT_ZIP_FILE
EVIDENCE_AUDIT_EXTRACT_ZIP_FILE
Extraction des signatures
à partir des fichiers sécurisés
Fin :
STP_EVIDENCE_AUDIT_EXTRACT_ZIP_FILE
Si au moins une action de l'étape est KO, alors l'étape est KO
Fin du workflow
Début :
STP_EVIDENCE_AUDIT_PREPARE_GENERATE_REPORTS
Préparation des rapports
(pour chaque objet, groupe d'objets,
unité audités)

2.5 Rapport d'audit de cohérence

Le rapport d'audit de cohérence est un fichier JSON généré par la solution logicielle Vitam lorsqu'une opération d'audit se termine. Cette section décrit la manière dont ce rapport est structuré.

2.5.1 Exemple de JSON : rapport d'audit


```
aeaqaaaaaahlm6sdabkeoaldcyfb3fyaaaaq:
    Identifier:                     "aeaqaaaaaahlm6sdabkeoaldcyfb3fyaaaaq"
    EvidenceStatus:                 "OK"
    ObjectType:                     "Unit"
aebaaaaaaahlm6sdabkeoaldcyeoe6qaaaaq:
    Identifier:                     "aebaaaaaaahlm6sdabkeoaldcyeoe6qaaaaq"
    EvidenceStatus:                 "OK"
    ObjectType:                     "ObjectGroup"
  ObjectsReports:
    0:
        Identifier:                 "aeaaaaaaaahlm6sdabkeoaldcyeoe6iaaaaq"
        EvidenceStatus:             "OK"
```


2.5.2 Détails du rapport

Chaque section du rapport correspond au résultat de l'audit de cohérence pour chaque objet ou groupe d'objets ou unité archivistique audité. On y trouve les informations suivantes :

- « Identifier » : Identifiant de l'objet ou groupe d'objets ou unité archivistique audité.
- « EvidenceStatus » : Résultat de l'audit : OK si toutes les signatures des fichiers sécurisés, des offres de stockage et de la base de données sont présentes et identiques. Le résultat est KO si au moins une des signatures est absente, ou bien si une des 3 est différente.
- « ObjectType » : type de l'objet audit : objet ou groupe d'objets ou unité archivistique.

Note : pour les audits exécutés sur des groupes d'objets, le rapport comporte aussi une section « ObjectReports » qui donne les informations de chaque audit réalisé sur les objets du groupe. On y retrouve aussi les identifiants (dans le champ « Identifier »), le résultat (dans le champ « EvidenceStatus »).

CHAPITRE 3

DIP

3.1 Workflow d'export d'un DIP

3.1.1 Introduction

Cette section décrit le processus (workflow) d'export, utilisé lors de l'export d'un Dissemination Information Package (DIP) dans la solution logicielle Vitam.

Toutes les étapes et actions sont journalisées dans le journal des opérations. Les étapes et actions associées ci-dessous décrivent le processus d'export de DIP (clé et description de la clé associée dans le journal des opérations) tel qu'implémenté dans la version actuelle de la solution logicielle Vitam.

3.1.2 Processus de création du bordereau de mise à disposition (STP_CREATE_MANIFEST)

3.1.2.1 Création du Bordereau CREATE_MANIFEST (CreateManifest.java)

- **Règle** : Création d'un bordereau contenant les unités archivistiques soumises au service d'export de DIP, ainsi que le groupes d'objets techniques et objets-données qui leurs sont associés
- **Type** : bloquant
- **Statuts** :
  - OK : le bordereau contenant les descriptions des unités archivistiques, groupes d'objets techniques et objets-données a été créé avec succès (CREATE_MANIFEST.OK=Succès de la création du bordereau de mise à disposition)
  - KO : la création du bordereau contenant les descriptions des unités archivistiques, groupes d'objets techniques et objets-données a échouée car des informations étaient manquantes, érronées ou inconnues (CREATE_MANIFEST.KO=Échec de la création du bordereau de mise à disposition)
  - FATAL : une erreur fatale est survenue lors de la création du bordereau (CREATE_MANIFEST.FATAL=Erreur fatale lors de la création du bordereau de mise à disposition)

3.1.3 Processus de déplacement des objets binaires vers le workspace (STP_PUT_BINARY_ON_WORKSPACE)

3.1.3.1 Déplacement des objets binaires vers le workspace PUT_BINARY_ON_WORKSPACE (PutBinaryOnWorkspace.java)

- **Règle** : Déplacement des objets-données mentionnées dans le bordereau vers le workspace
- **Type** : bloquant
- **Statuts** :
  - OK : les objets-données ont été déplacés vers le workspace avec succès (PUT_BINARY_ON_WORKSPACE.OK=Succès du déplacement des objets binaires de l'offre de stockage vers le workspace)
  - KO : le déplacement des objet-données vers le workspace a échoué car un ou plusieurs de ces objets étaient introuvables (PUT_BINARY_ON_WORKSPACE.KO=Échec du déplacement des objets binaires de l'offre de stockage vers le workspace)
  - FATAL : une erreur fatale est survenue lors du déplacement des objets binaires de stockage vers le workspace (PUT_BINARY_ON_WORKSPACE.FATAL=Erreur fatale lors du déplacement des objets binaires de l'offre de stockage vers le workspace)

3.1.4 Processus de création du DIP et de son déplacement vers l'offre de stockage (STP_STORE_MANIFEST)

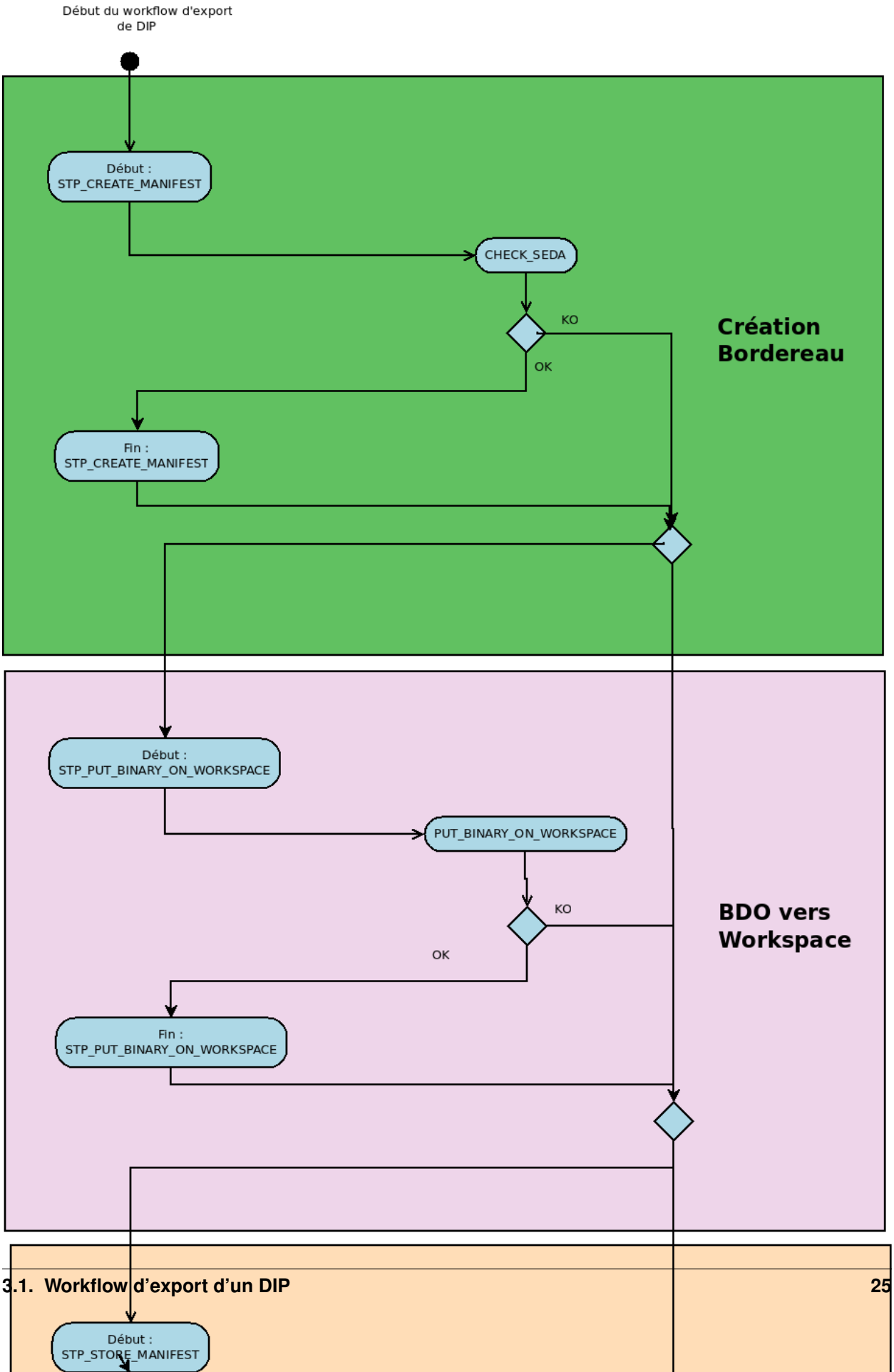
3.1.4.1 Stockage du bordereau compressé (STORE_MANIFEST - StoreDIP.java)

- **Règle** : Création du DIP et de son déplacement vers l'offre de stockage
- **Type** : bloquant
- **Statuts** :
  - OK : le DIP a été créé et stocké sur les offres de stockages avec succès (STORE_MANIFEST.OK=Succès du processus de la création du DIP et de son déplacement vers l'offre de stockage)
  - FATAL : une erreur fatale est survenue lors de la création et de l'enregistrement du DIP sur les offres de stockage déplacement des objets binaires de stockage vers workspace (STORE_MANIFEST.FATAL=Erreur fatale lors de la création du DIP et de son déplacement vers l'offre de stockage)

3.1.5 Structure du Workflow d'export de DIP

Le workflow d'export de DIP actuel mis en place dans la solution logicielle Vitam est défini dans l'unique fichier "ExportUnitWorkflow.json". Ce fichier est disponible dans /sources/processing/processing-management/src/main/resources/workflows.

D'une façon synthétique, le workflow est décrit de cette façon :


Début du workflow d'export de DIP
Début : STP_CREATE_MANIFEST
CHECK_SEDA
KO
OK
Création Bordereau
Fin : STP_CREATE_MANIFEST
Début : STP_PUT_BINARY_ON_WORKSPACE
PUT_BINARY_ON_WORKSPACE
KO
OK
BDO vers Workspace
Fin : STP_PUT_BINARY_ON_WORKSPACE
Début : STP_STORE_MANIFEST

CHAPITRE 4

INGEST

4.1 Workflow d'entrée

4.1.1 Introduction

Cette section décrit le processus (workflow) d'entrée, utilisé lors du transfert d'un Submission Information Package (SIP) dans la solution logicielle Vitam. Ce workflow se décompose en deux grandes catégories : le processus d'entrée externe dit « ingest externe » et le processus d'entrée interne dit « ingest interne ». Le premier prend en charge le SIP et effectue des contrôles techniques préalables, tandis que le second débute dès le premier traitement métier.

Toutes les étapes, tâches et traitements sont journalisés dans le journal des opérations. Ces derniers associées ci-dessous décrivent le processus d'entrée (clé et description de la clé associée dans le journal des opérations) tel qu'implémenté dans la version actuelle de la solution logicielle Vitam.

Le processus d'entrée externe comprend l'étape : STP_SANITY_CHECK_SIP (Contrôle sanitaire du SIP). Les autres étapes font partie du processus d'entrée interne.

4.1.2 Processus des contrôles préalables à l'entrée (STP_SANITY_CHECK_SIP)

4.1.2.1 Contrôle sanitaire du SIP SANITY_CHECK_SIP (IngestExternalImpl.java)

- **Règle** : Vérification de l'absence de virus dans le SIP
- **Type** : bloquant
- **Statuts** :
  - OK : aucun virus n'est détecté dans le SIP (SANITY_CHECK_SIP.OK = Succès du processus des contrôles préalables à l'entrée)
  - KO : un ou plusieurs virus ont été détectés dans le SIP (SANITY_CHECK_SIP.KO = Échec du processus des contrôles préalables à l'entrée)
  - FATAL : une erreur fatale est survenue lors de la vérification de la présence de virus dans le SIP (SANITY_CHECK_SIP.FATAL = Erreur fatale lors du processus des contrôles préalables à l'entrée)

4.1.2.2 Contrôle du format du conteneur du SIP CHECK_CONTAINER (IngestExternalImpl.java)

- **Règle** : Vérification du format du SIP via un outil d'identification de format qui se base sur le référentiel des formats qu'il intègre
- **Formats acceptés** : .zip, .tar, .tar.gz, .tar.bz2 et tar.gz2
- **Type** : bloquant
- **Statuts** :
  - OK : le conteneur du SIP est au bon format (CHECK_CONTAINER.OK = Succès du contrôle du format du conteneur du SIP)
  - KO : le conteneur du SIP n'est pas au bon format (CHECK_CONTAINER.KO = Échec du contrôle du format du conteneur du SIP)
  - FATAL : une erreur fatale est survenue lors de la vérification du format du conteneur du SIP, liée à l'outil d'identification des formats (CHECK_CONTAINER.FATAL = Erreur fatale lors du contrôle du format du conteneur du SIP)

4.1.3 Processus de réception du SIP dans Vitam STP_UPLOAD_SIP (IngestInternalResource.java)

- **Règle** : Vérification de la bonne réception du SIP dans l'espace de travail interne (« workspace »)
- **Type** : bloquant
- **Statuts** :
  - OK : le SIP a été réceptionné sur l'espace de travail interne (STP_UPLOAD_SIP.OK = Succès du processus de réception du SIP)
  - KO : le SIP n'a pas été réceptionné sur l'espace de travail interne (STP_UPLOAD_SIP.KO = Échec du processus de réception du SIP)
  - FATAL : une erreur fatale est survenue lors de la réception du SIP dans la solution logicielle Vitam, par exemple une indisponibilité du serveur (STP_UPLOAD_SIP.FATAL = Erreur fatale lors du processus de réception du SIP)

4.1.4 Processus de contrôle du SIP (STP_INGEST_CONTROL_SIP)

4.1.4.1 Préparation des informations de stockage PREPARE_STORAGE_INFO (PrepareStorageInfoActionHandler.java)

- **Règle** : Récupération des informations liées aux offres de stockage à partir de la stratégie
- **Type** : bloquant
- **Statuts** :
  - OK : Succès de la préparation des informations de stockage (PREPARE_STORAGE_INFO.OK = Succès de la préparation des informations de stockage)
  - KO : Echec de la préparation des informations de stockage (PREPARE_STORAGE_INFO.KO = Echec de la préparation des informations de stockage)
  - FATAL :erreur fatale est survenue lors de la préparation des informations de stockage (PREPARE_STORAGE_INFO.FATAL = Erreur fatale lors de la préparation des informations de stockage)

4.1.4.2 Vérification globale du SIP CHECK_SEDA (CheckSedaActionHandler.java)

- **Règle** : Vérification de la cohérence physique du SIP reçu par rapport au modèle de SIP accepté
- **Type de SIP accepté** : le bordereau de transfert, obligatoire dans le SIP, doit être nommé manifest.xml, doit être conforme au schéma xsd par défaut fourni avec le standard SEDA v. 2.1, doit satisfaire les exigences du document « Structuration des SIP » et doit posséder un répertoire unique nommé « Content »
- **Type** : bloquant
- **Statuts** :
  - OK : le SIP est présent, nommé manifest.xml et conforme au schéma xsd par défaut fourni avec le standard SEDA v.2.1. (CHECK_SEDA.OK = Succès de la vérification globale du SIP)
  - KO :
    - Cas 1 : le bordereau de transfert est introuvable dans le SIP ou n'est pas au format XML (CHECK_SEDA.NO_FILE.KO = Échec de la vérification globale du SIP : absence du bordereau de transfert ou bordereau de transfert au mauvais format)
    - Cas 2 : le bordereau de transfert n'est pas au format XML (CHECK_SEDA.NOT_XML_FILE.KO = Échec de la vérification globale du SIP : bordereau de transfert non conforme aux caractéristiques d'un fichier xml)
    - Cas 3 : le bordereau de transfert ne respecte pas le schéma par défaut fourni avec le standard SEDA 2.1 (CHECK_SEDA.NOT_XSD_VALID.KO = Échec de la vérification globale du SIP : bordereau de transfert non conforme au schéma SEDA 2.1)
    - Cas 4 : le SIP contient plus d'un dossier « Content » (CHECK_SEDA.CONTAINER_FORMAT.DIRECTORY.KO = Le SIP contient plus d'un dossier ou un dossier dont le nommage est invalide)
    - Cas 5 : le SIP contient plus d'un seul fichier à la racine (CHECK_SEDA.CONTAINER_FORMAT.FILE.KO = Le SIP contient plus d'un fichier à sa racine)
  - FATAL : une erreur fatale est survenue lors de du contrôle de cohérence (CHECK_SEDA.FATAL = Erreur fatale lors de la vérification globale du SIP)

4.1.4.3 Vérification de l'en-tête du bordereau de transfert CHECK_HEADER (CheckHeaderActionHandler.java)

- **Règles** : Vérification des informations générales du bordereau de transfert (nommées « header » dans le fichier « manifest.xml ») et de l'existence du service producteur (OriginatingAgencyIdentifier)
- **Type** : bloquant
- **Statuts** :
  - OK : les informations du bordereau de transfert sont conformes et le service producteur est déclaré (CHECK_HEADER.OK = Succès de la vérification générale du bordereau de transfert)
  - KO : les informations du bordereau de transfert ne sont pas conformes ou il n'y a pas de service producteur déclaré (CHECK_HEADER.KO = Échec de la vérification générale du bordereau de transfert)
  - FATAL : une erreur fatale est survenue lors des contrôles sur les informations générales du bordereau de transfert (CHECK_HEADER.FATAL = Erreur fatale lors de la vérification générale du bordereau de transfert)

4.1.4.3.1 La tâche check_header contient les traitements suivants :

4.1.4.3.2 Vérification de la présence et contrôle des services agents (CHECK_AGENT).

Cette tâche est exécutée si la valeur IN de checkOriginatingAgency est true.

- **Règle** : Vérification du service producteur ainsi que du service versant déclarés dans le SIP par rapport au référentiel des services agents présent dans la solution logicielle Vitam
- **Type** : bloquant
- **Statuts** :
  - OK : le service producteur et/ou le service versant déclaré dans le SIP est valide (service agent existant dans le référentiel des services agents)(CHECK_HEADER.CHECK_AGENT.OK=Succès de la vérification de la présence et du contrôle des services agents)
  - KO :
    - Cas 1 : aucun service producteur n'est déclaré dans la balise dédiée dans le bordereau de transfert (CHECK_HEADER.CHECK_AGENT.EMPTY_REQUIRED_FIELD.KO= Échec de la vérification de la présence et du contrôle des services agents : champ obligatoire vide)
    - Cas 2 : le service producteur et/ou le service versant déclaré dans le SIP n'est pas connue du référentiel des services agents (CHECK_HEADER.CHECK_AGENT.UNKNOWN.KO= Échec de la vérification de la présence et du contrôle des services agents : services agents inconnus du référentiel des services agents)
    - Cas 3 : la balise permettant de déclarer un service producteur est absente du bordereau de tranfert (CHECK_HEADER.CHECK_AGENT.KO=Échec de la vérification de la présence et du contrôle des services agents)
  - FATAL : une erreur fatale est survenue lors de la vérification de la présence et du contrôle des services agents (CHECK_HEADER.CHECK_AGENT.FATAL=Erreur fatale lors de la vérification de la présence et du contrôle des services agents)

4.1.4.3.3 Vérification de la présence et contrôle du contrat d'entrée (CHECK_CONTRACT_INGEST)

Cette tâche est exécutée si la valeur IN de checkContract est true.

- **Règle** : Vérification du contrat d'entrée déclaré dans le SIP par rapport au référentiel des contrats d'entrée présent dans la solution logicielle Vitam
- **Type** : bloquant
- **Statuts** :
  - OK : le contrat déclaré dans le SIP est valide (contrat existant dans le référentiel des contrats et dont le statut est actif)(CHECK_HEADER.CHECK_CONTRACT_INGEST.OK=Succès de la vérification de la présence et du contrôle du contrat d'entrée)
  - KO :
    - Cas 1 : le contrat déclaré dans le SIP est inexistant (CHECK_HEADER.CHECK_CONTRACT_INGEST.CONTRACT_ du contrôle de la présence du contrat d'entrée : contrat d'entrée inconnu du référentiel des contrats d'entrée)
    - Cas 2 : le contrat déclaré dans le SIP est inactif (CHECK_HEADER.CHECK_CONTRACT_INGEST.CONTRACT_INA du contrôle du caractère actif du contrat d'entrée)
    - Cas 3 : Aucun contrat d'entrée trouvé dans le manifest (CHECK_HEADER.CHECK_CONTRACT_INGEST.CONTRACT_NOT_IN_MANIFEST.KO=Échec du contrôle de la présence du contrat d'entrée dans le bordereau de transfert)
    - Cas 4 : le contrat déclaré dans le SIP n'existe pas dans le contexte applicatif (CHECK_HEADER.CHECK_CONTRACT_INGEST.CONTRACT_NOT_IN_CONTEXT.KO=Échec du contrôle de la présence du contrat d'entrée dans le contexte applicatif)
    - Cas 5 : le contexte applicatif est inexistant (CHECK_HEADER.CHECK_CONTRACT_INGEST.CONTEXT_UNKNO du contrôle de la présence du contexte applicatif : contexte inconnu du référentiel des contextes)
    - Cas 6 : le contexte applicatif est inactif (CHECK_HEADER.CHECK_CONTRACT_INGEST.CONTEXT_INACTIVE.I du contrôle du caractère actif du contexte applicatif)

- Cas 7 : Erreur lors de la récupération du contexte applicatif (CHECK_HEADER.CHECK_CONTRACT_INGEST.CONTEXT_CHECK_ERROR.KO=Échec de la vérification de la présence et du contrôle du contexte applicatif)

- FATAL : une erreur fatale est survenue lors de la vérification de la présence et du contrôle du contrat d'entrée ou du contexte applicatif (CHECK_HEADER.CHECK_CONTRACT_INGEST.FATAL=Erreur fatale lors de la vérification de la présence et du contrôle du contrat d'entrée ou du contexte applicatif)

4.1.4.3.4 Vérification de la relation entre le contrat d'entrée et le profil d'archivage (CHECK_IC_AP_RELATION)

Cette tâche est exécutée si la valeur IN de checkProfile est true.

- **Règle** : le profil d'archivage déclaré dans le contrat d'entrée du SIP doit être le même que celui déclaré dans son bordereau de transfert.
- **Statuts** :
  - OK : le profil d'archivage déclaré dans le contrat d'entrée et celui déclaré dans le bordereau de transfert sont les mêmes (CHECK_HEADER.CHECK_IC_AP_RELATION.OK = Succès de la vérification de la relation entre le contrat d'entrée et le profil)
  - KO :
    - Cas 1 : le profil déclaré dans le SIP est inexistant (CHECK_HEADER.CHECK_IC_AP_RELATION.UNKNOWN.KO=E du contrôle de la présence du profil d'archivage dans le référentiel des profils d'archivage)
    - Cas 2 : le profil déclaré dans le SIP est inactif (CHECK_HEADER.CHECK_IC_AP_RELATION.INACTIVE.KO=Échec du contrôle du caractère actif du profil d'archivage)
    - Cas 3 : le profil déclaré dans le contrat d'entrée et celui déclaré dans le bordereau de transfert ne sont pas les mêmes (CHECK_HEADER.CHECK_IC_AP_RELATION.DIFF.KO=Échec du contrôle de cohérence entre le profil d'archivage déclaré dans le bordereau de transfert et celui déclaré dans le contrat d'entrée)
  - FATAL : une erreur fatale est survenue lors de la vérification de la relation entre le contrat d'entrée et le profil d'archivage (CHECK_HEADER.CHECK_IC_AP_RELATION.FATAL = Erreur fatale lors de la vérification de la relation entre le contrat d'entrée et le profil d'archivage)

4.1.4.3.5 Vérification de la conformité du bordereau de transfert par le profil d'archivage (CHECK_ARCHIVEPROFILE)

- **Règle** : le bordereau de transfert du SIP doit être conforme aux exigences du profil d'archivage. Si aucun profil SEDA ne s'applique au SIP, ce traitement est ignoré.
- **Type** : bloquant
- **Statuts** :
  - OK : le bordereau de transfert est conforme aux exigences du profil d'archivage (CHECK_HEADER.CHECK_ARCHIVEPROFILE.OK = Succès de la vérification de la conformité au profil d'archivage)
  - KO : le bordereau de transfert n'est pas conforme aux exigences du profil d'archivage (CHECK_HEADER.CHECK_ARCHIVEPROFILE.KO = Échec de la vérification de la conformité au profil d'archivage)
  - FATAL : une erreur fatale est survenue lors de la vérification du bordereau de transfert par le profil d'archivage (CHECK_HEADER.CHECK_ARCHIVEPROFILE.FATAL = Erreur fatale lors de la vérification de la conformité au profil d'archivage)

4.1.4.4 Vérification du contenu du bordereau CHECK_DATAOBJECTPACKAGE (CheckDataObjectPackageActionHandler.java)

- **Règles** : Vérification du contenu du bordereau de transfert et de sa cohérence.
- **Type** : bloquant.

4.1.4.4.1 La tâche CHECK_DATAOBJECTPACKAGE contient plusieurs traitements.

4.1.4.4.2 Vérification des usages des groupes d'objets CHECK_DATAOBJECTPACKAGE.CHECK_MANIFEST_DATAO (CheckVersionActionHandler.java)

- **Règle** : Tous les objets décrits dans le bordereau de transfert du SIP doivent déclarer un usage conforme à la liste des usages acceptés dans la solution logicielle Vitam ainsi qu'un numéro de version respectant la norme de ce champ
- **Types d'usages acceptés** : original papier (PhysicalMaster), original numérique (BinaryMaster), diffusion (Dissemination), vignette (Thumbnail), contenu brut (TextContent). Les numéros de versions sont optionnels, il s'agit d'un entier positif ou nul (0, 1, 2…). La grammaire est : « usage_version ». Exemples : « BinaryMaster_2 », « TextContent_10 » ou sans numéro de versions « PhysicalMaster ».
- **Statuts** :
  - OK : les objets contenus dans le SIP déclarent tous dans le bordereau de transfert un usage cohérent avec ceux acceptés et optionnellement un numéro de version respectant la norme de ce champ usage, par exemple « BinaryMaster_2 » (CHECK_MANIFEST_DATAOBJECT_VERSION.OK = Succès de la vérification des usages des objets)
  - KO :
    - Cas 1 : un ou plusieurs BinaryMaster sont déclarées dans un ou plusieurs objets physiques (CHECK_DATAOBJECTPACKAGE.CHECK_MANIFEST_DATAOBJECT_VERSION.PDO_DATAOBJECTIONVE = L'objet physique déclare un usage « BinaryMaster ». Cet usage n'est pas autorisé pour les objets physiques
    - Cas 2 : un ou plusieurs PhysicalMaster sont déclarés dans un ou plusieurs objets binaires (CHECK_DATAOBJECTPACKAGE.BDO_DATAOBJECTIONVERSION_PHYSICALMASTER.KO=Au moins un objet binaire déclare un usage « PhysicalMaster ». Cet usage n'est pas autorisé pour les objets binaires)
    - Cas 3 : un ou plusieurs objets contenus dans le SIP déclarent dans le bordereau de transfert un usage ou un numéro de version incohérent avec ceux acceptés (CHECK_DATAOBJECTPACKAGE.CHECK_MANIFEST_DATAOBJECT_VERSION.INVALID_DATAOBJECTVE objet déclare un usage incorrect. L'usage doit s'écrire sous la forme [usage] ou [usage]_[version]. « Usage » doit être parmi l'énumération DataObjectVersion définie pour Vitam, « version » doit être un entier positif)
    - Cas 4 : une ou plusieurs URI sont vides (CHECK_DATAOBJECTPACKAGE.CHECK_MANIFEST_DATAOBJECT_V existe au moins un champ non renseigné dont la valeur est obligatoire)
  - FATAL : une erreur fatale est survenue lors du contrôle des usages déclarés dans le bordereau de transfert pour les objets contenus dans le SIP (CHECK_MANIFEST_DATAOBJECT_VERSION.FATAL = Erreur fatale lors de la vérification des usages des objets)

4.1.4.4.3 Vérification du nombre d'objets CHECK_MANIFEST_OBJECTNUMBER (CheckObjectsNumberActionHandler.java)

- **Règle** : Le nombre d'objets binaires reçus dans la solution logicielle Vitam doit être strictement égal au nombre d'objets binaires déclaré dans le manifeste du SIP

- **Type** : bloquant.
- **Statuts** :
  - OK : le nombre d'objets reçus dans la solution logicielle Vitam est strictement égal au nombre d'objets déclaré dans le bordereau de transfert du SIP (CHECK_MANIFEST_OBJECTNUMBER.OK = Succès de la vérification du nombre d'objets)
  - KO :
    - Cas 1 : le nombre d'objets reçus dans la solution logicielle Vitam est supérieur au nombre d'objets déclaré dans le bordereau de transfert du SIP (CHECK_DATAOBJECTPACKAGE.CHECK_MANIFEST_OBJECTNUMBER.MANIFEST_INFERIOR_BDO.KO= bordereau de transfert déclare moins d'objets binaires qu'il n'en existe dans le répertoire Content du SIP)
    - Cas 2 : le nombre d'objets reçus dans la solution logicielle Vitam est inférieur au nombre d'objets déclaré dans le bordereau de transfert du SIP (CHECK_DATAOBJECTPACKAGE.CHECK_MANIFEST_OBJECTNUMBER.MANIFEST_SUPERIOR_BDO.KO= bordereau de transfert déclare plus d'objets binaires qu'il n'en existe dans le répertoire Content du SIP)
    - Cas 3 : une ou plusieurs balises URI déclarent un chemin invalide (CHECK_DATAOBJECTPACKAGE.CHECK_MANIFEST_OBJECTNUMBER.INVALID_URI.KO=Au moins un objet déclare une URI à laquelle ne correspond pas de fichier ou déclare une URI déjà utilisée par un autre objet)
  - FATAL : une erreur fatale est survenue lors de la vérification du nombre d'objets (CHECK_DATAOBJECTPACKAGE.CHECK_MANIFEST_OBJECTNUMBER.FATAL = Erreur fatale lors de la vérification du nombre d'objets)

4.1.4.4.4 Vérification de la cohérence du bordereau de transfert CHECK_MANIFEST (ExtractSedaActionHandler.java)

- **Règle** : Création des journaux du cycle de vie des unités archivistiques et des groupes d'objets, extraction des unités archivistiques, objets binaires et objets physiques, vérification de la présence de récursivités dans les arborescences des unités archivistiques et création de l'arbre d'ordre d'indexation, extraction des métadonnées contenues dans la balise ManagementMetadata du bordereau de transfert pour le calcul des règles de gestion, vérification de la validité du rattachement des unités du SIP aux unités présentes dans la solution logicielle Vitam si demandé, détection des problèmes d'encodage dans le bordereau de transfert et vérification que les objets ne font pas référence directement à des unités si ces objets possèdent des groupes d'objets.
- **Type** : bloquant.
- **Statuts** :
  - OK : les journaux du cycle de vie des unités archivistiques et des groupes d'objets ont été créés avec succès, aucune récursivité n'a été détectée dans l'arborescence des unités archivistiques, la structure de rattachement déclarée existe (par exemple, un SIP peut être rattaché à un plan de classement, mais pas l'inverse), le type de structure de rattachement est autorisé, aucun problème d'encodage détecté et les objets avec groupe d'objets ne référencent pas directement les unités. L'extraction des unités archivistiques, objets binaires et physiques, la création de l'arbre d'indexation et l'extraction des métadonnées des règles de gestion ont été effectuées avec succès. (CHECK_MANIFEST.OK = Succès du contrôle de cohérence du bordereau de transfert). L'extraction des unités archivistiques, objets binaires et physiques, la création de l'arbre d'indexation et l'extraction des métadonnées des règles de gestion ont été effectuées avec succès.
  - KO :
    - Cas 1 : une ou plusieurs balises de rattachement vers un GOT existant déclarent autre chose que le GUID d'un GOT existant (CHECK_DATAOBJECTPACKAGE.CHECK_MANIFEST.EXISTING_OG_NOT_DECLARED.KO=Une unité archivistique déclare un objet à la place du groupe d'objets correspondant)

- Cas 2 : une ou plusieurs balises de rattachement vers une AU existant déclarent autre chose que le GUID d'une AU existante (CHECK_DATAOBJECTPACKAGE.CHECK_MANIFEST.CHECK_MANIFEST_WRONG_ATTACHMENT.KO=Le bordereau de transfert procède à un rattachement en utilisant des éléments inexistants dans le système)
- Cas 3 : il y a un problème lors du contrôle à un noeud de rattachement CHECK_DATAOBJECTPACKAGE.CHECK_MANIFEST.CHECK_MANIFEST_WRONG_ATTACHMENT_LINK.K bordereau de transfert procède à un rattachement en utilisant des éléments hors périmètre.
- Cas 4 : Une récursivité a été détectée dans l'arborescence des unités archivistiques (CHECK_DATAOBJECTPACKAGE.CHECK_MANIFEST.CHECK_MANIFEST_LOOP.KO=Le bordereau de transfert présente une récursivité dans l'arborescence de ses unités archivistiques)
- Cas 5 : il y a un problème d'encodage ou des objets référencent directement des unités archivistiques (CHECK_DATAOBJECTPACKAGE.CHECK_MANIFEST.KO = Échec du contrôle de cohérence du bordereau de transfert)

- FATAL : une erreur fatale est survenue lors de la vérification de la cohérence du bordereau, par exemple les journaux du cycle de vie n'ont pu être créés (CHECK_MANIFEST.FATAL = Erreur fatale lors du contrôle de cohérence du bordereau de transfert)

4.1.4.4.5 Vérification de la cohérence entre objets, groupes d'objets et unités archivistiques CHECK_CONSISTENCY (CheckObjectUnitConsistencyActionHandler.java)

- **Règle** : Vérification que chaque objet ou groupe d'objets est référencé par une unité archivistique, rattachement à un groupe d'objets pour les objets sans groupe d'objets mais référencés par une unité archivistique, création de la table de concordance (MAP) pour les identifiants des objets et des unités archivistiques du SIP et génération de leurs identifiants Vitam (GUID)
- **Type** : bloquant.
- **Statuts** :
  - OK : aucun objet ou groupe d'objets n'est orphelin (c'est à dire non référencé par une unité archivistique) et tous les objets sont rattachés à un groupe d'objets. La table de concordance est créée et les identifiants des objets et unités archivistiques ont été générés. (CHECK_CONSISTENCY.OK = Succès de la vérification de la cohérence entre objets, groupes d'objets et unités archivistiques)
  - KO : au moins un objet ou groupe d'objets est orphelin (c'est-à-dire non référencé par une unité archivistique) (CHECK_CONSISTENCY.KO = Échec de la vérification de la cohérence entre objets, groupes d'objets et unités archivistiques)
  - FATAL : une erreur fatale est survenue lors de la vérification de la cohérence entre objets, groupes d'objets et unités archivistiques (CHECK_CONSISTENCY.FATAL = Erreur fatale lors de la vérification de la cohérence entre objets, groupes d'objets et unités archivistiques)

4.1.5 Processus de contrôle et traitement des objets (STP_OG_CHECK_AND_TRANSFORME)

4.1.5.1 Vérification de l'intégrité des objets CHECK_DIGEST (CheckConformityActionPlugin.java)

- **Règle** : Vérification de la cohérence entre l'empreinte de l'objet binaire calculée par la solution logicielle Vitam et celle déclarée dans le bordereau de transfert. Si l'empreinte déclarée dans le bordereau de transfert n'a pas été calculée avec l'algorithme SHA-512, alors l'empreinte est recalculée avec cet algorithme. Elle sera alors enregistrée dans la solution logicielle Vitam.
- **Algorithmes autorisés en entrée** : MD5, SHA-1, SHA-256, SHA-512
- **Type** : bloquant
- **Statuts** :

- OK : tous les objets binaires reçus sont identiques aux objets binaires attendus. Tous les objets binaires disposent désormais d'une empreinte calculée avec l'algorithme SHA-256 (CHECK_DIGEST.OK = Succès de la vérification de l'empreinte des objets)
- KO :
  - Cas 1 : au moins un objet reçu n'a pas d'empreinte dans le bordereau (CHECK_DIGEST.EMPTY.KO=Échec lors de la vérification de l'empreinte des objets : Il existe au moins un objet dont l'empreinte est absente dans le bordereau de transfert)
  - Cas 2 : au moins une empreinte d'un objet reçu n'est pas conforme à son empreinte dans le bordereau (CHECK_DIGEST.INVALID.KO=Échec lors de la vérification de l'empreinte des objets : Il existe au moins un objet dont l'empreinte est invalide dans le bordereau de transfert)
  - Cas 3 : le SIP soumis à la solution logicielle Vitam contient à la fois le cas 1 et le cas 2 (CHECK_DIGEST.KO=Échec de la vérification de l'empreinte des objets)
- FATAL : une erreur fatale est survenue lors de la vérification de l'intégrité des objets binaires, par exemple lorsque l'algorithme est inconnu (CHECK_DIGEST.FATAL = Erreur fatale lors de la vérification de l'empreinte des objets)

4.1.5.1.1 Identification des formats (OG_OBJECTS_FORMAT_CHECK - FormatIdentificationActionPlugin.java)

- **Règle** : Identification des formats de chaque objet binaire présent dans le SIP, afin de garantir une information homogène. Cette action met en œuvre un outil d'identification prenant l'objet en entrée et fournissant des informations de format en sortie. Ces informations sont comparées avec les formats enregistrés dans le référentiel des formats interne à la solution logicielle Vitam et avec celles déclarées dans le bordereau de transfert. En cas d'incohérence entre la déclaration dans le SIP et le format identifié, le SIP sera accepté, générant un avertissement. La solution logicielle Vitam se servira alors des informations qu'elle a identifiées et non de celles fournies dans le SIP
- **Type** : bloquant
- **Statuts** :
  - OK : l'identification s'est bien passée, les formats identifiés sont référencés dans le référentiel interne et les informations sont cohérentes avec celles déclarées dans le manifeste (OG_OBJECTS_FORMAT_CHECK.OK = Succès de la vérification des formats)
  - KO :
    - Cas 1 : au moins un objet reçu a un format qui n'a pas été trouvé (OG_OBJECTS_FORMAT_CHECK.KO = Échec de l'identification des formats)
    - Cas 2 : au moins un objet reçu a un format qui n'est pas référencé dans le référentiel interne (OG_OBJECTS_FORMAT_CHECK.UNCHARTED.KO=Échec lors de l'identification des formats, le format de ou des objet(s) est identifié mais est inconnu du référentiel des formats)
    - Cas 3 : le SIP soumis à la solution logicielle Vitam contient à la fois le cas 1 et le cas 2 (OG_OBJECTS_FORMAT_CHECK.KO = Échec de l'identification des formats)
  - FATAL : une erreur fatale est survenue lors de l'indentification des formats (OG_OBJECTS_FORMAT_CHECK.FATAL = Erreur fatale lors de l'identification des formats)
  - WARNING : l'identification s'est bien passée, les formats identifiés sont référencés dans le référentiel interne mais les informations ne sont pas cohérentes avec celles déclarées dans le manifeste (OG_OBJECTS_FORMAT_CHECK.WARNING = Avertissement lors de la vérification des formats)

4.1.6 Processus de contrôle et traitement des unités archivistiques (STP_UNIT_CHECK_AND_PROCESS)

4.1.6.1 Vérification globale de l'unité archivistique CHECK_UNIT_SCHEMA (CheckArchiveUnitSchemaActionPlugin.java)

- **Règle** : Contrôle additionnel sur la validité des champs de l'unité archivistique par rapport au schéma prédéfini dans la solution logicielle Vitam. Par exemple, les champs obligatoires, comme les titres des unités archivistiques, ne doivent pas être vides. Lorsque le manifeste déclare une personne (Person) et non une société (Entity), alors au moins un champ entre « Firstname » et « Birthname » est obligatoire. En plus du contrôle par le schéma, cette tâche vérifie pour les dates extrêmes que la date de fin est bien supérieure ou égale à la date de début de l'unité archivistique.
- **Type** : bloquant
- **Statuts** :
  - OK : tous les champs de l'unité archivistique sont conformes à ce qui est attendu (CHECK_UNIT_SCHEMA.OK = Succès de la vérification globale de l'unité archivistique)
  - KO :
    - Cas 1 : au moins un champ d'une unité archivistique dont le schéma n'est pas conforme par rapport au schéma prédéfini du référentiel Vitam. (CHECK_UNIT_SCHEMA.INVALID_UNIT.KO=Échec lors de la vérification globale de l'unité archivistique : champs non conformes)
    - Cas 2 : au moins un champ obligatoire d'une unité archivistique est vide(CHECK_UNIT_SCHEMA.EMPTY_REQUIRED_FIELD.KO=Échec lors de la vérification globale de l'unité archivistique : champs obligatoires vides)
    - Cas 3 : au moins un champ date d'une unité archivistique est supérieur à 9000 (titre vide, date incorrecte…) ou la date de fin des dates extrêmes est strictement inférieure à la date de début (CHECK_UNIT_SCHEMA.RULE_DATE_THRESHOLD.KO=Échec du calcul des dates d'échéance, la date ne peut être gérée)
    - Cas 4 : Échec du calcul des dates : au moins un champ date d'une unité archivistique possède un format non conforme (CHECK_UNIT_SCHEMA.RULE_DATE_FORMAT.KO=Échec du calcul des dates d'échéance, la date ne peut être gérée)
    - Cas 5 : Au moins une valeur de l'unité archivistique n'est pas conforme à son schéma en raison d'un problème de cohérence entre champs. Par exemple, la valeur contenue dans le champs « StartDate » est postérieure à la date définie dans la « EndDate » (CHECK_UNIT_SCHEMA.CONSISTENCY.KO=Au moins une unité archivistique n'est pas conforme à son schéma en raison d'un problème de cohérence entre champs)
  - FATAL : une erreur fatale est survenue lors de la vérification de l'unité archivistique (CHECK_UNIT_SCHEMA.FATAL=Erreur fatale lors de la vérification globale de l'unité archivistique)

4.1.6.2 Vérification du profil d'unité archivistique - si celui-ci est déclaré CHECK_ARCHIVE_UNIT_PROFILE (CheckArchiveUnitProfileActionPlugin.java)

- **Règle** : Vérification de la conformité au niveau des unités archivistiques : si celles ci font référence à un profil d'unité archivistique, présent dans la balise « ArchiveUnitProfile »
- **Type** : non bloquant
- **Statuts** :
  - OK : les unités archivistiques versées et ayant un profil d'unité archivistique de référence bien conformes au schéma décrit dans le profil d'unité archivistique, et ceux ci existent bien dans le système (CHECK_ARCHIVE_UNIT_PROFILE.OK = Succès de la vérification de la conformité aux profils d'unité archivistique)

- KO : au moins une unité archivistique n'est pas conforme au schéma décrit dans le profil d'unité archivistique associé (CHECK_ARCHIVE_UNIT_PROFILE.KO = Echec de la vérification de la conformité au profil d'unité archivistique)
- PROFILE NOT FOUND : au moins une unité archivistique est déclarée en lien avec un profil d'unité archivistique via la balise ArchiveUnitProfile , et ce référentiel n'existe pas dans le système (CHECK_ARCHIVE_UNIT_PROFILE.PROFILE_NOT_FOUND.KO=Échec de la vérification de la conformité au profil d'unité archivistique : profil d'unité archivistique non trouvé)
- INVALID UNIT : au moins une unité archivistique n'est pas conforme au schéma décrit dans le profil d'unité archivistique associé (CHECK_ARCHIVE_UNIT_PROFILE.INVALID_UNIT.KO = Échec de la vérification de la conformité au profil d'unité archivistique : champs non conformes)
- INVALID AU : le profil d'unité archivistique cité dans le référentiel est mal formaté (CHECK_ARCHIVE_UNIT_PROFILE.INVALID_AU_PROFILE.KO=Échec de la vérification de la conformité aux documents type : document type non conforme)

4.1.6.3 Vérification du niveau de classification CHECK_CLASSIFICATION_LEVEL (CheckClassificationLevelActionPlugin.java)

- **Règle** : Vérification des niveaux de classification associés, s'il existe, aux unités archivistiques. Ces niveaux doivent exister dans la liste des niveaux de classifications autorisés par la plateforme (paramètre configuré dans la configuration des workers). Pour les unités archivistiques sans niveau de classification, la vérification contrôle que la plateforme autorise le versement d'unités archivistiques sans niveau de classification.
- **Type** : bloquant
- **Statuts** :
  - OK : les unités archivistiques versées ont un niveau de classification autorisé par la plateforme. S'il existe dans le SIP des unités archivistiques sans niveau de classification, c'est que la plateforme autorise le versement d'unités archivistiques sans niveau de classification. (CHECK_CLASSIFICATION_LEVEL.OK=Succès de la vérification du niveau de classification)
  - KO : au moins une unité archivistique du SIP possède un niveau de classification qui n'est pas un niveau de classification autorisé par la plateforme, ou une unité archivistique n'a pas de niveau de classification alors que la plateforme requiert que toutes les unités archivistiques possèdent un niveau de classification. (CHECK_CLASSIFICATION_LEVEL.KO=Échec de la vérification du niveau de classification, non autorisés par la plateforme : le bordereau de transfert déclare un niveau de classification non autorisé par la plateforme)
  - FATAL : une erreur fatale est survenue lors de la vérification des niveaux de classifications (CHECK_CLASSIFICATION_LEVEL.FATAL=Erreur fatale lors de la vérification du niveau de classification)

4.1.6.3.1 Application des règles de gestion et calcul des dates d'échéances UNITS_RULES_COMPUTE (UnitsRulesComputePlugin.java)

- **Règle** : Calcul des dates d'échéances des unités archivistiques du SIP. Pour les unités racines, c'est à dire les unités déclarées dans le SIP et n'ayant aucun parent dans l'arborescence, la solution logicielle Vitam utilise les règles de gestion incluses dans le bloc Management de chacune de ces unités ainsi que celles présentes dans le bloc ManagementMetadata. La solution logicielle Vitam effectue également ce calcul pour les autres unités archivistiques du SIP possédant des règles de gestion déclarées dans leurs balises Management, sans prendre en compte le ManagementMetadata. Le référentiel utilisé pour ces calculs est le référentiel des règles de gestion de la solution logicielle Vitam.
- **Type** : bloquant
- **Statuts** :

- OK : les règles de gestion sont référencées dans le référentiel interne et ont été appliquées avec succès (UNITS_RULES_COMPUTE.OK = Succès de l'application des règles de gestion et du calcul des dates d'échéance)
- KO :
  - Cas 1 : au moins une règle de gestion déclarée dans le manifeste n'est pas référencée dans le référentiel interne ou au moins une règle est incohérent avec sa catégorie (UNITS_RULES_COMPUTE.UNKNOWN.KO=Échec lors de l'application des règles de gestion et du calcul des dates d'échéance : règle de gestion inconnue)
  - Cas 2 : une balise RefnonRuleId a un identifiant d'une règle d'une autre catégorie que la sienne (UNITS_RULES_COMPUTE.REF_INCONSISTENCY.KO=Échec lors de l'application des règles de gestion et du calcul des dates d'échéance : exclusion d'héritage incohérente)
- FATAL : une erreur fatale est survenue lors du calcul des dates d'échéances (UNITS_RULES_COMPUTE.FATAL = Erreur fatale lors de l'application des règles de gestion et du calcul des dates d'échéance)

4.1.7 Processus de vérification préalable à la prise en charge (STP_STORAGE_AVAILABILITY_CHECK)

4.1.7.1 Vérification de la disponibilité de toutes les offres de stockage (STORAGE_AVAILABILITY_CHECK - CheckStorageAvailabilityActionHandler.java)

- **Règle** : Vérification de la disponibilité des offres de stockage et de l'espace disponible pour y stocker le contenu du SIP compte tenu de la taille des objets à stocker
- **Type** : bloquant
- **Statuts** :
  - OK : les offres de stockage sont accessibles et disposent d'assez d'espace pour stocker le contenu du SIP (STORAGE_AVAILABILITY_CHECK.OK = Succès de la vérification de la disponibilité de toutes les offres de stockage)
  - KO :
    - Cas 1 : les offres de stockage ne sont pas disponibles (STORAGE_AVAILABILITY_CHECK.STORAGE_OFFER_KO_UNAVAILABLE.KO= Au moins une offre de stockage n'est pas disponible)
    - Cas 2 : les offres ne disposent pas d'assez d'espace pour stocker le contenu du SIP (STORAGE_AVAILABILITY_CHECK.STORAGE_OFFER_SPACE_KO.KO= Au moins une offre de stockage est insuffisante)
  - FATAL : une erreur fatale est survenue lors de la vérification de la disponibilité de l'offre de stockage (STORAGE_AVAILABILITY_CHECK.FATAL = Erreur fatale lors de la vérification de la disponibilité d'au moins une offre de stockage)

4.1.7.2 Vérification de la disponibilité de l'offre de stockage STORAGE_AVAILABILITY_CHECK.STORAGE_AVAILABILITY_CHECK (CheckStorageAvailabilityActionHandler.java)

- **Règle** : Vérification de la disponibilité de l'offres de stockage et de l'espace disponible pour y stocker le contenu du SIP compte tenu de la taille des objets à stocker
- **Type** : bloquant
- **Statuts** :

- OK : l'offre de stockage est accessible et dispose d'assez d'espace pour stocker le contenu du SIP (STORAGE_AVAILABILITY_CHECK.STORAGE_AVAILABILITY_CHECK.OK=Succès de la vérification de la disponibilité de l'offre de stockage)
- KO :
  - Cas 1 : l'offre de stockage n'est pas disponible (STORAGE_AVAILABILITY_CHECK.STORAGE_AVAILABILITY_CHECK.STORAGE_OFFER_KO_UNAVAILABLE.K de stockage n'est pas disponible)
  - Cas 2 : l'offre de stockage ne dispose pas d'assez d'espace pour stocker le contenu du SIP (STORAGE_AVAILABILITY_CHECK.STORAGE_AVAILABILITY_CHECK.STORAGE_OFFER_SPACE_KO.KO=Dispo de l'offre de stockage insuffisante)
- FATAL : une erreur fatale est survenue lors de la vérification de la disponibilité de l'offre de stockage (STORAGE_AVAILABILITY_CHECK.STORAGE_AVAILABILITY_CHECK.FATAL=Erreur fatale lors de la vérification de la disponibilités de l'offre de stockage)

4.1.8 Processus d'écriture et indexation des objets et groupes d'objets (STP_OBJ_STORING)

4.1.8.1 Ecriture des objets sur l'offre de stockage OBJ_STORAGE (StoreObjectActionHandler.java)

- **Règle** : Ecriture des objets contenus dans le SIP sur les offres de stockage en fonction de la stratégie de stockage applicable
- **Type** : Bloquant
- **Statuts** :
  - OK : tous les objets binaires contenus dans le SIP ont été écrits sur les offres de stockage (OBJ_STORAGE.OK = Succès de l'écriture des objets et des groupes d'objets sur les offres de stockage)
  - KO : au moins un des objets binaires contenus dans le SIP n'a pas pu être écrit sur les offres de stockage (OBJ_STORAGE.KO = Échec de l'écriture des objets et des groupes d'objets sur les offres de stockage)
  - WARNING : le SIP ne contient pas d'objet (OBJECTS_LIST_EMPTY.WARNING = Avertissement lors de l'établissement de la liste des objets : il n'y a pas d'objet pour cette étape)
  - FATAL : une erreur fatale est survenue lors de l'écriture des objets binaires sur les offres de stockage (OBJ_STORAGE.FATAL = Erreur fatale lors de l'écriture des objets et des groupes d'objets sur les offres de stockage)

4.1.8.2 Indexation des métadonnées des groupes d'objets (OG_METADATA_INDEXATION - IndexObjectGroupActionPlugin.java)

- **Règle** : Indexation des métadonnées des groupes d'objets dans les bases internes de la solution logicielle Vitam, comme la taille des objets, les métadonnées liées aux formats (Type MIME, PUID, etc.), l'empreinte des objets, etc.
- **Type** : bloquant
- **Statuts** :
  - OK : les métadonnées des groupes d'objets ont été indexées avec succès (OG_METADATA_INDEXATION.OK = Succès de l'indexation des métadonnées des objets et des groupes d'objets)
  - KO : les métadonnées des groupes d'objets n'ont pas été indexées (OG_METADATA_INDEXATION.KO = Échec de l'indexation des métadonnées des objets et des groupes d'objets)
  - FATAL : une erreur fatale est survenue lors de l'indexation des métadonnées des groupes d'objets (OG_METADATA_INDEXATION.FATAL = Erreur fatale lors de l'indexation des métadonnées des objets et des groupes d'objets)

4.1.9 Processus d'indexation des unités archivistiques (STP_UNIT_METADATA)

4.1.9.1 Indexation des métadonnées des unités archivistiques (UNIT_METADATA_INDEXATION - IndexUnitActionPlugin.java)

- **Règle** : Indexation des métadonnées des unités archivistiques dans les bases internes de la solution logicielle Vitam, c'est à dire le titre des unités, leurs descriptions, leurs dates extrêmes, etc.
- **Type** : bloquant
- **Statuts** :
  - OK : les métadonnées des unités archivistiques ont été indexées avec succès (UNIT_METADATA_INDEXATION.OK = Succès de l'indexation des métadonnées de l'unité archivistique)
  - KO : les métadonnées des unités archivistiques n'ont pas été indexées (UNIT_METADATA_INDEXATION.KO = Échec de l'indexation des métadonnées de l'unité archivistique)
  - FATAL : une erreur fatale est survenue lors de l'indexation des métadonnées des unités archivistiques (UNIT_METADATA_INDEXATION.FATAL = Erreur fatale lors de l'indexation des métadonnées de l'unité archivistique)

4.1.10 Processus d'enregistrement et écriture des métadonnées des objets et groupes d'objets(STP_OG_STORING)

4.1.10.1 Enregistrement des journaux du cycle de vie des groupes d'objets COMMIT_LIFE_CYCLE_OBJECT_GROUP (CommitLifeCycleObjectGroupActionHandler.java)

- **Règle** : Sécurisation en base des journaux du cycle de vie des groupes d'objets (avant cette étape, les journaux du cycle de vie des groupes d'objets sont dans une collection temporaire afin de garder une cohérence entre les métadonnées indexées et les journaux lors d'une entrée en succès ou en échec)(Pas d'évènements créées dans le journal du cycle de vie)
- **Type** : bloquant
- **Statuts** :
  - OK : la sécurisation des journaux du cycle de vie s'est correctement déroulée (COMMIT_LIFE_CYCLE_OBJECT_GROUP.OK = Succès de l'enregistrement des journaux du cycle de vie des groupes d'objets)
  - FATAL : une erreur fatale est survenue lors de la sécurisation du journal du cycle de vie (COMMIT_LIFE_CYCLE_OBJECT_GROUP.FATAL = Erreur fatale lors de l'enregistrement des journaux du cycle de vie des groupes d'objets)

4.1.10.2 Ecriture des métadonnées du groupe d'objets sur l'offre de stockage OG_METADATA_STORAGE (StoreMetaDataObjectGroupActionPlugin)

- **Règle** : Sauvegarde des métadonnées liées aux groupes d'objets ainsi que leurs journaux de cycle de vie sur les offres de stockage en fonction de la stratégie de stockage
- **Type** : bloquant
- **Statuts** :
  - OK : les métadonnées des groupes d'objets ont été sauvegardées avec succès (OG_METADATA_STORAGE.OK = Succès de l'écriture des métadonnées des objets et groupes d'objets sur l'offre de stockage)
  - KO : les métadonnées des groupes d'objets n'ont pas été sauvegardées (OG_METADATA_STORAGE.KO = Échec de l'écriture des métadonnées des objets et groupes d'objets sur l'offre de stockage)

4.1.11 Processus d'enregistrement et écriture des unités archivistiques (STP_UNIT_STORING)

4.1.11.1 Enregistrement du journal du cycle de vie des unités archivistiques COMMIT_LIFE_CYCLE_UNIT (AccessInternalModuleImpl.java)

- **Règle** : Sécurisation en base des journaux du cycle de vie des unités archivistiques (avant cette étape, les journaux du cycle de vie des unités archivistiques sont dans une collection temporaire afin de garder une cohérence entre les métadonnées indexées et les journaux lors d'une entrée en succès ou en échec)
- **Type** : bloquant
- **Statuts** :
  - OK : le différentiel est créé et est enregistré dans l'evDetData (OBJECT_GROUP_UPDATE.OK = Succès de l'enregistrement des journaux du cycle de vie des groupes d'objets)
  - FATAL : une erreur fatale est survenue (OBJECT_GROUP_UPDATE.FATAL =)

4.1.11.2 Ecriture des métadonnées de l'unité archivistique sur l'offre de stockage UNIT_METADATA_STORAGE (AccessInternalModuleImpl.java)

- **Règle** : Sauvegarde des métadonnées et des journaux de cycle de vie des unités archivistiques sur les offres de stockage en fonction de la stratégie de stockage.(Pas d'évènements stockés dans le journal de cycle de vie
- **Type** : bloquant
- **Statuts** :
  - OK : la sécurisation des journaux du cycle de vie s'est correctement déroulée (COMMIT_LIFE_CYCLE_OBJECT_GROUP.OK = Succès de l'enregistrement des journaux du cycle de vie des groupes d'objets)
  - FATAL : une erreur fatale est survenue lors de la sécurisation du journal du cycle de vie (COMMIT_LIFE_CYCLE_OBJECT_GROUP.FATAL = Erreur fatale lors de l'enregistrement des journaux du cycle de vie des groupes d'objets)

4.1.12 Processus de mise à jour du groupe d'objets (STP_UPDATE_OBJECT_GROUP)

4.1.12.1 Création du différentiel OBJECT_GROUP_UPDATE (UpdateObjectGroupPlugin.java)

- **Règle** : création du différentiel pour le groupe d'objets.
- **Type** : bloquant
- **Statuts** :
  - OK : le différentiel est créé et est enregistré dans l'evDetData (OBJECT_GROUP_UPDATE.OK = Succès lors du processus de mise à jour du groupe d'objets)
  - FATAL : une erreur fatale est survenue (OBJECT_GROUP_UPDATE.FATAL = Erreur fatale lors du processus de mise à jour du groupe d'"'objets)
  - FATAL : une erreur fatale est survenue (OBJECT_GROUP_UPDATE.FATAL =)

4.1.12.2 Etablissement de la liste des objets (OBJECTS_LIST_EMPTY)

- **Règle** : Etablissement de la liste des objets pré existante dans le groupe d'objet technique avant le rattachement à l'unité archivistique. Si aucun rattachement n'est déclaré dans le bordereau de transfert, alors cette tâche est OK.

- **Type** : bloquant
- **Statuts** :
  - OK : La liste des objets a été créé avec succès (STP_UPDATE_OBJECT_GROUP.OK = Succès lors de l'' établissement de la liste des objets)
  - FATAL : La liste des objets n'a pas été créée (STP_UPDATE_OBJECT_GROUP.FATAL = Erreur fatale lors de l'' établissement de la liste des objets)

4.1.12.3 Alimentation du registre des fonds ACCESSION_REGISTRATION (AccessionRegisterActionHandler.java)

- **Règle** : Enregistrement dans le registre des fonds des informations concernant la nouvelle entrée (nombre d'objets, volumétrie). Ces informations viennent s'ajouter aux informations existantes pour un même service producteur. Si le service producteur n'était pas déjà présent pas le registre des fonds, alors cette entrée est enregistrée et le service producteur est créé dans le registre des fonds.
- **Type** : bloquant
- **Statuts** :
  - OK : le registre des fonds est correctement alimenté (ACCESSION_REGISTRATION.OK = Succès de l'alimentation du registre des fonds)
  - KO : le registre des fonds n'a pas pu être alimenté (ACCESSION_REGISTRATION.KO = Échec de l'alimentation du registre des fonds)
  - FATAL : une erreur fatale est survenue lors de l'alimentation du registre des fonds (ACCESSION_REGISTRATION.FATAL = Erreur fatale lors de l'alimentation du registre des fonds)

4.1.13 Processus de finalisation de l'entrée (STP_INGEST_FINALISATION)

4.1.13.1 Notification de la fin de l'opération d'entrée ATR_NOTIFICATION (TransferNotificationActionHandler.java)

- **Règle** : Génération de la notification de réponse (ArchiveTransferReply ou ATR) une fois toutes les étapes passées, en succès, avertissement ou échec, puis écriture de cette notification dans l'offre de stockage et envoi au service versant.
- **Type** : non bloquant
- **Statuts** :
  - OK : Le message de réponse a été correctement généré, écrit sur l'offre de stockage et envoyé au service versant (ATR_NOTIFICATION.OK = Succès de la notification de la fin de l'opération d'entrée à l'opérateur de versement)
  - KO : Le message de réponse n'a pas été correctement généré, écrit sur l'offre de stockage ou reçu par le service versant (ATR_NOTIFICATION.KO = Échec de la notification de la fin de l'opération d'entrée à l'opérateur de versement)
  - FATAL : une erreur fatale est survenue lors de la notification de la fin de l'opération (ATR_NOTIFICATION.FATAL = Erreur fatale lors de la notification de la fin de l'opération d'entrée à l'opérateur de versement)

4.1.13.2 Mise en cohérence des journaux du cycle de vie ROLL_BACK (RollBackActionHandler.java)

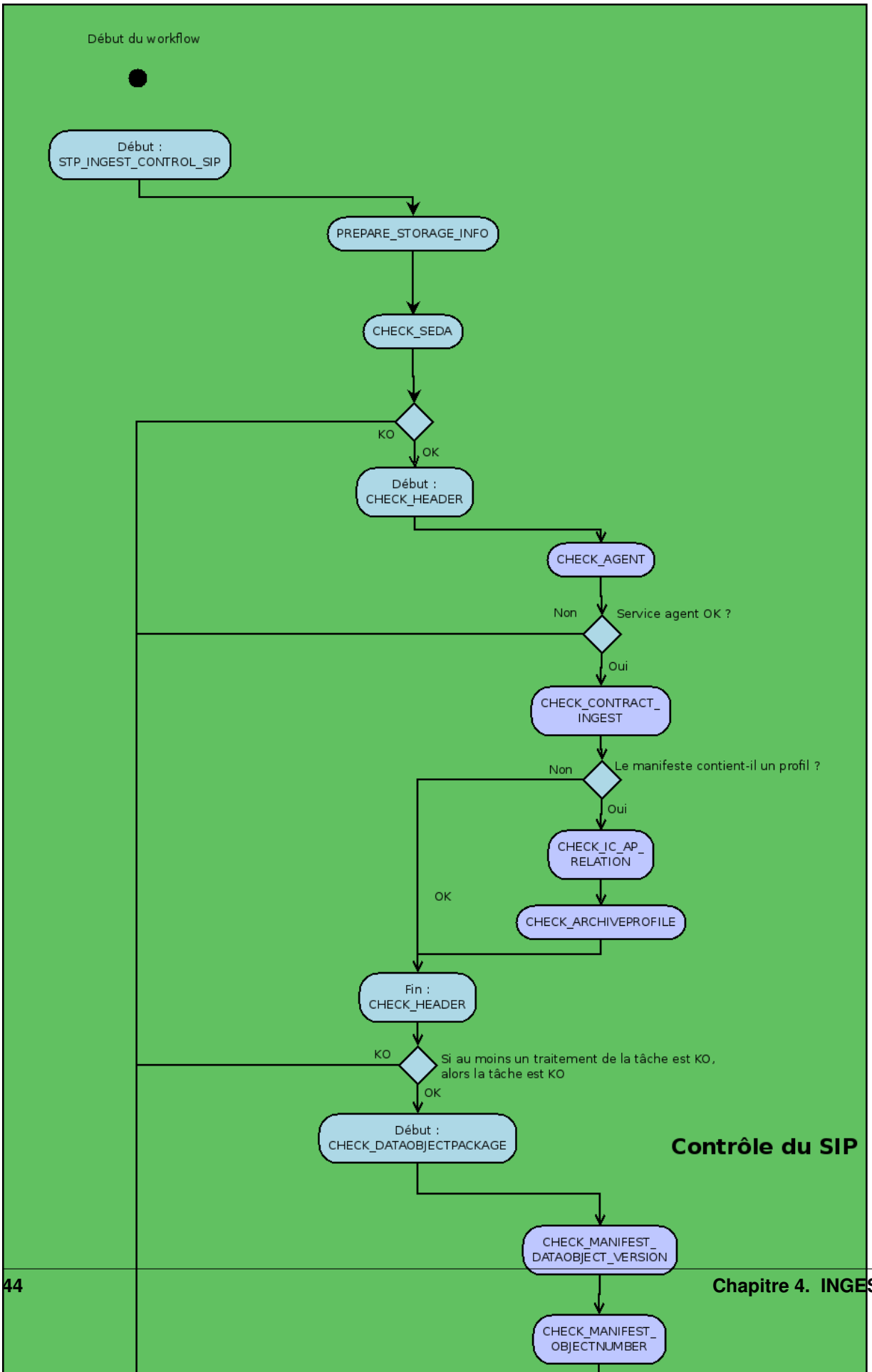
- **Règle** : Purge des collections temporaires des journaux du cycle de vie
- **Type** : bloquant

- **Statuts** :
  - OK : la purge s'est correctement déroulée (ROLL_BACK.OK = Succès de la mise en cohérence des journaux du cycle de vie)
  - FATAL : une erreur fatale est survenue lors de la purge (ROLL_BACK.FATAL = Erreur fatale lors de la mise en cohérence des journaux du cycle de vie)

4.1.14 Structure du Workflow

Le workflow mis en place dans la solution logicielle Vitam est défini dans l'unique fichier « DefaultIngestWorkflow.json ». Ce fichier est disponible dans /sources/processing/processing-management/src/main/resources/workflows. Il décrit le processus d'entrée (hors Ingest externe) pour entrer un SIP, indexer les métadonnées et stocker les objets contenus dans le SIP.

D'une façon synthétique, le workflow est décrit de cette façon :


Début du workflow
Début : STP_INGEST_CONTROL_SIP
PREPARE_STORAGE_INFO
CHECK_SEDA
KO
OK
Début : CHECK_HEADER
CHECK_AGENT
Non
Service agent OK ?
Oui
CHECK_CONTRACT_ INGEST
Non
Le manifeste contient-il un profil ?
Oui
CHECK_IC_AP_ RELATION
OK
CHECK_ARCHIVEPROFILE
Fin : CHECK_HEADER
KO
Si au moins un traitement de la tâche est KO, alors la tâche est KO
OK
Début : CHECK_DATAOBJECTPACKAGE
Contrôle du SIP
CHECK_MANIFEST_ DATAOBJECT_VERSION
CHECK_MANIFEST_ OBJECTNUMBER

4.1.15 Le cas du processus d'entrée « test à blanc »

Il est possible de procéder à un versement dit « à blanc », pour tester la conformité du SIP par rapport à la forme attendue par la solution logicielle Vitam sans pour autant le prendre en charge. Dans ce cas, le processus d'entrée à blanc diffère du processus d'entrée « classique » en ignorant un certain nombre d'étapes.

Les étapes non exécutées dans le processus d'entrée à blanc sont les suivantes :

- Ecriture et indexation des objets et groupes d'objets (STP_OBJ_STORING)
- Indexation des unités archivistiques (STP_UNIT_METADATA)
- Enregistrement et écriture des métadonnées des objets et groupes d'objets (STP_OG_STORING)
- Enregistrement et écriture des unités archivistiques (STP_UNIT_STORING)
- Registre des fonds (STP_ACCESSION_REGISTRATION)

Les tâches et traitements relatifs à toutes ces étapes sont donc également ignorées.

4.2 Workflow d'entrée d'un plan de classement

4.2.1 Introduction

Cette section décrit le processus d'entrée d'un plan de classement dans la solution logicielle Vitam. La structure d'un plan de classement diffère de celle d'un SIP par l'absence d'objet et de vérification par rapport à un profil d'archivage. Il s'agit plus simplement d'une arborescence représentée par des unités archivistiques. Ce processus partage donc certaines étapes avec celui du transfert d'un SIP classique, en ignore certaines et rajoute des tâches additionnelles.

4.2.2 Processus d'entrée d'un plan de classement (vision métier)

Le processus d'entrée d'un plan est identique au workflow d'entrée d'un SIP. Il débute lors du lancement du téléchargement d'un plan de classement dans la solution logicielle Vitam. Toutes les étapes et traitements sont journalisées dans le journal des opérations.

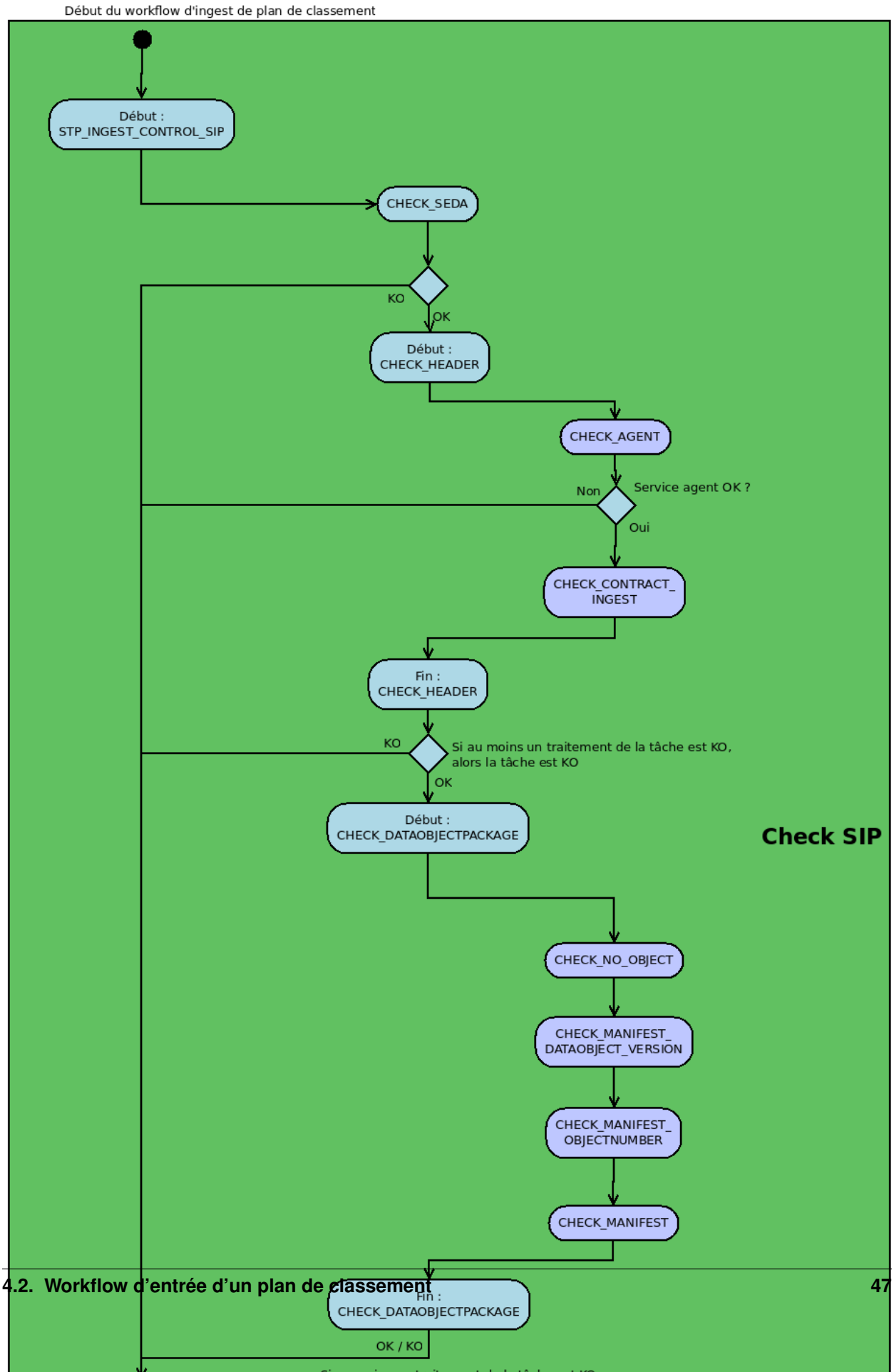
Les étapes et traitements associées ci-dessous décrivent le processus d'entrée d'un plan (clé et description de la clé associée dans le journal des opérations), non encore abordées dans la description de l'entrée d'un SIP.

4.2.2.1 Traitement additionnel dans la tâche CHECK_DATAOBJECTPACKAGE

- Vérification de la non existence d'objets (CHECK_NO_OBJECT)
  - **Règle** : vérification qu'il n'y a pas d'objet numérique dans le bordereau de transfert du plan
  - **Statuts** :
  - OK : aucun objet numérique n'est présent dans le bordereau de transfert (CHECK_DATAOBJECTPACKAGE.CHECK_NO_OBJECT.OK=Succès de la vérification de l'absence d'objet)
  - KO : des objets numériques sont présent dans le bordereau de transfert (CHECK_DATAOBJECTPACKAGE.CHECK_NO_OBJECT.KO=Échec de la vérification de l'absence d'objet : objet(s) trouvé(s))
  - FATAL : une erreur fatale est survenue lors de la vérification de la non existence d'objet numérique (CHECK_DATAOBJECTPACKAGE.CHECK_NO_OBJECT.FATAL=Erreur fatale lors de la vérification de l'absence d'objet)

Le workflow actuel mis en place dans la solution logicielle Vitam est défini dans le fichier « DefaultFilingSchemeWorkflow.json ». Ce fichier est disponible dans : sources/processing/processing-management/src/main/resources/workflows.

D'une façon synthétique, le workflow est décrit de cette façon :


Début du workflow d'ingest de plan de classement
Début : STP_INGEST_CONTROL_SIP
CHECK_SEDA
KO
OK
Début : CHECK_HEADER
CHECK_AGENT
Non
Service agent OK ?
Oui
CHECK_CONTRACT_ INGEST
Fin : CHECK_HEADER
KO
Si au moins un traitement de la tâche est KO, alors la tâche est KO
OK
Début : CHECK_DATAOBJECTPACKAGE
Check SIP
CHECK_NO_OBJECT
CHECK_MANIFEST_ DATAOBJECT_VERSION
CHECK_MANIFEST_ OBJECTNUMBER
CHECK_MANIFEST
Fin : CHECK_DATAOBJECTPACKAGE
OK / KO

CHAPITRE 5

MASTERDATA

5.1 Introduction

Cette section décrit les processus (workflows) d'administration des différents référentiels de la solution logicielle Vitam. Ceux-ci se construisent sur la base de fichiers à importer. La structure de ces fichiers et la description de leurs contenus sont décrites dans la documentation relative au modèle de données. Si un des fichiers importés contient des balises HTML, son contenu sera considéré comme dangereux et l'import sera rejeté. Ce rejet ne fera pas l'objet d'une opération et ne sera donc pas enregistré dans le journal des opérations. En revanche, une alerte de sécurité sera émise dans un log de sécurité de la solution logicielle Vitam, pour en informer l'administrateur de cette tentative.

5.2 Workflow d'import d'un arbre de positionnement

5.2.1 Introduction

Cette section décrit le processus permettant d'importer un arbre de positionnement dans la solution logicielle Vitam. La structure d'un arbre de positionnement diffère de celle d'un SIP en plusieurs points.

Un arbre ne doit pas avoir d'objet, ni de service producteur, ni de contrat. Il s'agit plus simplement d'une arborescence représentée par des unités archivistiques. Ce processus partage donc certaines étapes avec celui du transfert d'un SIP classique, en ignore certaines et rajoute des tâches additionnelles.

5.2.2 Processus d'import d'un arbre (HOLDINGSCHEME - vision métier)

Le processus d'import d'un arbre est identique au workflow d'entrée d'un SIP. Il débute lors du lancement du téléchargement de l'arbre dans la solution logicielle Vitam. Par ailleurs, toutes les étapes, tâches et traitements sont journalisés dans le journal des opérations.

La fin du processus peut prendre plusieurs statuts :

- **Statuts** :

- OK : l'arbre de positionnement a été importé (HOLDINGSCHEME.OK = Succès de l'import de l'arbre de positionnement)
- KO : l'arbre de positionnement n'a pas été importé (HOLDINGSCHEME.KO = Échec de l'import de l'arbre de positionnement)
- FATAL : une erreur fatale est survenue lors de l'import de l'arbre de positionnement (HOLDINGSCHEME.FATAL = Erreur fatale lors de l'import de l'arbre de positionnement)

Les étapes et tâches associées ci-dessous décrivent ce processus d'import (clé et description de la clé associée dans le journal des opérations), non encore abordées dans la description de l'entrée d'un SIP.

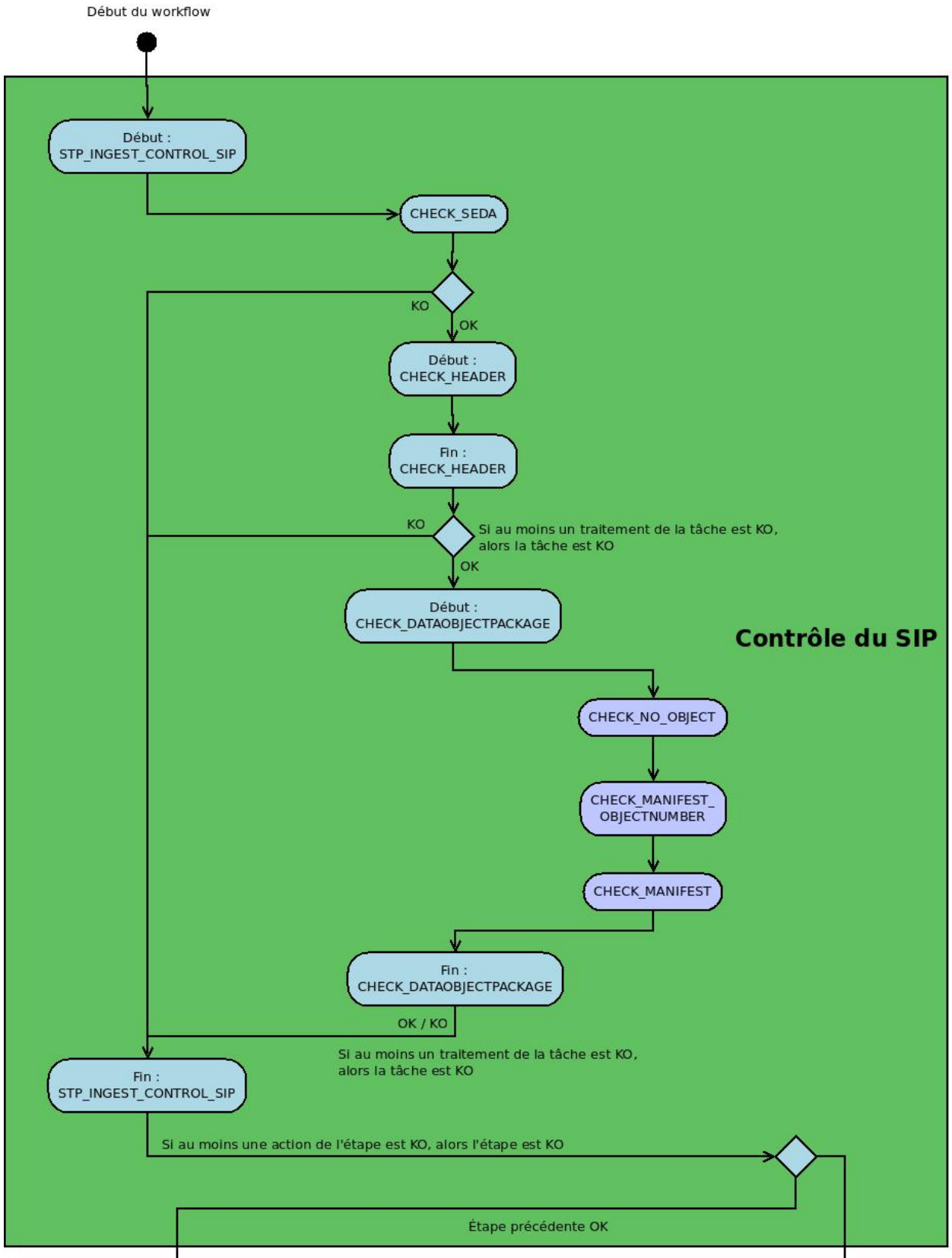
5.2.2.1 Traitement additionnel dans la tâche CHECK_DATAOBJECTPACKAGE (CheckDataObjectPackageActionHandler.java)

5.2.2.1.1 Vérification de la non existence d'objets CHECK_NO_OBJECT (CheckDataObjectPackageActionHandler)

- **Règle** : Vérification qu'il n'y a pas d'objet numérique dans le bordereau de transfert de l'arbre de positionnement.
  - **Statuts** :
    - OK : aucun objet numérique n'est présent dans le bordereau de transfert (CHECK_DATAOBJECTPACKAGE.CHECK_NO_OBJECT.OK=Succès de la vérification de l'absence d'objet)
    - KO : des objets numériques sont présent dans le bordereau de transfert (CHECK_DATAOBJECTPACKAGE.CHECK_NO_OBJECT.KO=Échec de la vérification de l'absence d'objet : objet(s) trouvé(s))
    - FATAL : une erreur fatale est survenue lors de la vérification de la non existence d'objet numérique (CHECK_DATAOBJECTPACKAGE.CHECK_NO_OBJECT.FATAL=Erreur fatale lors de la vérification de l'absence d'objet)

5.2.3 Structure du Workflow

Le workflow mis en place dans la solution logicielle Vitam est défini dans le fichier « DefaultHoldingSchemeWorkflow.json ». Ce fichier est disponible à sources/processing/processing-management/src/main/resources/workflows.


Début du workflow
Début : STP_INGEST_CONTROL_SIP
CHECK_SEDA
KO
OK
Début : CHECK_HEADER
Fin : CHECK_HEADER
KO
Si au moins un traitement de la tâche est KO, alors la tâche est KO
OK
Début : CHECK_DATAOBJECTPACKAGE
Contrôle du SIP
CHECK_NO_OBJECT
CHECK_MANIFEST_ OBJECTNUMBER
CHECK_MANIFEST
Fin : CHECK_DATAOBJECTPACKAGE
OK / KO
Si au moins un traitement de la tâche est KO, alors la tâche est KO
Fin : STP_INGEST_CONTROL_SIP
Si au moins une action de l'étape est KO, alors l'étape est KO
Étape précédente OK
Début : STP_UNIT_CHECK_AND_PROCESS
CHECK_UNIT_SCHEMA
OK / KO
Fin :
Contrôle et traitements des units

5.3 Workflow d'administration d'un référentiel de règles de gestion

5.3.1 Introduction

Cette section décrit le processus (workflow) permettant d'importer et de mettre à jour un référentiel de règles de gestions dans la solution logicielle Vitam.

5.3.2 Processus d'administration d'un référentiel de règles de gestion (STP_IMPORT_RULES)

L'import d'un référentiel de règles de gestion permet de vérifier le formalisme de ce dernier, notamment que les données obligatoires sont bien présentes pour chacune des règles. Tous les éléments réalisés au cours de ce processus sont exécutés dans une seule étape. Cet import concerne aussi bien l'import initial (aucune règles de gestion pré-existantes) que la mise à jour du référentiel.

Ce processus d'import débute lors du lancement du téléchargement du fichier CSV contenant le référentiel dans la solution logicielle Vitam. Par ailleurs, toutes les étapes, tâches et traitements sont journalisés dans le journal des opérations.

La fin du processus peut prendre plusieurs statuts :

- **Statuts** :
  - OK : le référentiel des règles de gestion a été importé (STP_IMPORT_RULES.OK = Succès du processus d'import du référentiel des règles de gestion)
  - Warning : le référentiel des règles de gestion a été importé et ce nouvel import modifie des règles de gestions préalablement utilisées par des unités archivistique dans la solution logicielle Vitam (STP_IMPORT_RULES.WARNING = Avertissement lors du processus d'import des règles de gestion, des règles de gestions ont été modifiées et sont utilisées par des unités archivistiques existantes)
  - KO : le référentiel des règles de gestion n'a pas été importé (STP_IMPORT_RULES.KO = Échec du processus d'import du référentiel des règles de gestion)
  - FATAL : une erreur fatale est survenue lors de l'import du référentiel des règles de gestion (STP_IMPORT_RULES.FATAL = Erreur fatale lors du processus d'import du référentiel des règles de gestion)

5.3.2.1 Création du rapport RULES_REPORT (RulesManagerFileImpl.java)

- **Règle** : création du rapport d'import des règles
- **Type** : bloquant
- **Statuts** :
  - OK : le rapport est généré (RULES_REPORT.OK = Succès de la génération du rapport d'analyse du référentiel des règles de gestion)
  - KO : pas de cas KO
  - FATAL : une erreur fatale est survenue lors de la création du rapport (RULES_REPORT.FATAL = Erreur fatale lors de la génération du rapport d'analyse du référentiel des règles de gestion)

5.3.2.2 Contrôle des règles de gestion CHECK_RULES (UnitsRulesComputePlugin.java)

- **Règle** : contrôle qu'aucune règle supprimée du référentie n'est utilisé par une unité archivistique. Contrôle des règles modifiées utilisées par des unités archivistiques. Vérification que les informations obligatoires minimales ont bien été remplies pour chacune des règles, conformément aux exigences du référentiel des règles de gestion. La liste de ces exigences est décrite dans le document modèle de données.

De plus le fichier rempli les conditions suivantes :

- il est au format CSV
- les informations suivantes sont toutes décrites dans cet ordre
  - RuleId
  - RuleType
  - RuleValue
  - RuleDescription
  - RuleDuration
  - RuleMeasurement
  - Aucune règle supprimée n'est utilisée par une unité archivistique
  - Aucune opération d'import de référentiel de règle de gestion n'a lieu en même temps
  - Si le tenant définit des durées minimales pour des catégories de règles de gestion (configuration de sécurité), les règles de gestion importées doivent avoir des durées supérieures ou égales à ces durées minimales de sécurité
- **Type** : bloquant
- **Statuts** :
  - OK : les règles ci-dessus sont respectées (CHECK_RULES = Contrôle de la conformité du fichier des règles de gestion)
  - WARNING : une règle modifiée par l'import du référentiel est actuellement utilisée par une unité archivistique (CHECK_RULES.WARNING = Avertissement lors du contrôle de la conformité du fichier des règles de gestion)
  - KO :
    - Cas 1 : une des règles ci-dessus n'est pas respectée. Le détail des erreurs est inscrit dans le rapport d'import du référentiel (CHECK_RULES.KO = Échec du contrôle de la conformité du fichier des règles de gestion)
    - Cas 2 : le fichier CSV n'est pas reconnu comme un CSV valide (CHECK_RULES.INVALID_CSV.KO=Échec du contrôle de la conformité du fichier des règles de gestion : fichier CSV invalide)
    - Cas 3 : une opération de mise à jour du référentiel est déjà en cours (CHECK_RULES.IMPORT_IN_PROCESS.KO=L'import est impossible car une mise à jour du référentiel est déjà en cours
    - Cas 4 : au moins une règle de gestion a une durée qui est inférieure à la durée minimale requise sur ce tenant. Selon la configuration de la solution logicielle Vitam, ce cas peut provoquer une alerte de sécurité, enregistrée dans les logs de sécurité. (CHECK_RULES.MAX_DURATION_EXCEEDS.KO=Echec lors du contrôle de sécurité des règles de gestion. Les durées des règles de gestion doivent être supérieures ou égales aux durées minimales requises par le tenant)
  - Dans le rapport de l'import du référentiel des règles de gestion, des clés plus détaillées peuvent y être inscrites, selon les erreurs rencontrées :
    - Le fichier importé n'est pas au format CSV (STP_IMPORT_RULES_NOT_CSV_FORMAT.KO = Le fichier importé n'est pas au format CSV)
    - Il existe plusieurs fois le même RuleId (STP_IMPORT_RULES_RULEID_DUPLICATION.KO = Il existe plusieurs fois le même RuleId. Ce RuleId doit être unique dans l'ensemble du référentiel)
    - Au moins une RuleType est incorrecte (STP_IMPORT_RULES_WRONG_RULETYPE_UNKNOW.KO = Au moins une RuleType est incorrecte. RuleType autorisés : AppraisalRule, AccessRule, StorageRule, DisseminationRule, ReuseRule, ClassificationRule)

- Au moins une valeur obligatoire est manquante (STP_IMPORT_RULES_MISSING_INFORMATION.KO = Au moins une valeur obligatoire est manquante. Valeurs obligatoires : RuleID, RuleType, RuleValue, RuleDuration, RuleMeasurement)
- Des valeurs de durée sont incorrectes pour RuleMeasurement (STP_IMPORT_RULES_WRONG_RULEMEASUREMENT.KO = Au moins un champ RuleDuration a une valeur incorrecte. La valeur doit être un entier positif ou nul, ou être indiquée unlimited)
- Au moins un champs RuleDuration a une valeur incorrecte (STP_IMPORT_RULES_WRONG_RULEDURATION.KO = Au moins un champ RuleDuration a une valeur incorrecte. La valeur doit être un entier positif ou nul, ou être indiquée unlimited)
- L'association de RuleDuration et de RuleMeasurement doit être inférieure ou égale à 999 ans (STP_IMPORT_RULES_WRONG_TOTALDURATION.KO = L'association de RuleDuration et de RuleMeasurement doit être inférieure ou égale à 999 ans)
- Des règles supprimées sont actuellement utilisées (STP_IMPORT_RULES_DELETE_USED_RULES.KO = Des régles supprimées sont actuellement utilisées)
- Des durées sont inférieures ou égales aux durées minimales autorisées dans la configuration de la plateforme (STP_IMPORT_RULES_RULEDURATION_EXCEED.KO = Echec lors du contrôle de sécurité des règles de gestion. Les durées des règles de gestion doivent être supérieures ou égales aux durées minimales requises par le tenant)

- FATAL : une erreur fatale est survenue lors du contrôle des règles de gestion (CHECK_RULES.FATAL=Erreur fatale lors du contrôle de la conformité du fichier de règles de gestion)

5.3.2.3 Persistance des données en base COMMIT_RULES (RulesManagerFileImpl.java)

- **Règle** : enregistrement des données en base
- **Type** : bloquant
- **Statuts** :
  - OK : les données sont persistées en base (COMMIT_RULES=OK=Succès de la persistance des données en base
  - FATAL : une erreur fatale est survenue lors de la persistance des données en base (COMMIT_RULES.FATAL=Erreur fatale lors de la persistance des données en base)

5.3.2.4 Processus d'enregistrement du fichier d'import du référentiel des règles de gestion STP_IMPORT_RULES_BACKUP_CSV (RulesManagerFileImpl.java)

- **Règle** : enregistrement du CSV d'import du référentiel des règles de gestion
- **Type** : bloquant
- **Statuts** :
  - OK : le CSV d'import est enregistré (STP_IMPORT_RULES_BACKUP_CSV.OK=Succès du processus d'enregistrement du fichier d'import du référentiel des règles de gestion)
  - KO : pas de cas KO
  - FATAL : une erreur fatale est survenue lors de l'enregistrement du CSV d'import (STP_IMPORT_RULES_BACKUP_CSV.FATAL = Erreur fatale lors du processus d'enregistrement du fichier d'import du référentiel des règles de gestion)

5.3.2.5 Sauvegarde du JSON STP_IMPORT_RULES_BACKUP (RulesManagerFileImpl.java)

- **Règle** : enregistrement d'une copie de la base de données sur le stockage
- **Type** : bloquant
- **Statuts** :
  - OK : une copie de la base de donnée nouvellement importée est enregistrée (STP_IMPORT_RULES_BACKUP.OK = Succès du Processus de sauvegarde du référentiel des règles de gestion)
  - KO : pas de cas KO
  - FATAL : une erreur fatale est survenue lors de la copie de la base de donnée nouvellement importée (STP_IMPORT_RULES_BACKUP.FATAL=Erreur fatale lors du processus de sauvegarde du référentiel des règles de gestion)

5.3.3 Structure du rapport d'administration du référentiel des règles de gestion

Lorsqu'un nouveau référentiel est importé, la solution logicielle Vitam génère un rapport de l'opération. Ce rapport est en 3 parties :

- « Operation » contient :
  - evType : le type d'opération. Dans le cadre de ce rapport, il s'agit toujours de « STP_IMPORT_RULES ».
  - evDateTime : la date et l'heure de l'opération d'import.
  - evId : l'identifiant de l'opération.
  - outMessg : message final de l'opération (Succès/Avertissement/Échec du processus d'import du référentiel des règles de gestion)
- « Error » : détail les erreurs en indiquant :
  - line : le numéro de la ligne du rapport CSV générant l'erreur
  - Code : le code d'erreur
  - Message : le message associée à l'erreur
  - Information additionnelle : une précision sur l'erreur, comme par exemple le contenu du champ qui l'a provoquée
  - « usedDeletedRules » : contient l'intégralité des règles en cours d'utilisation dont la suppression a été demandée lors de la mise à jour du référentiel des règles de gestion. Chaque détail précise en plus la date de création de la règle, sa dernière mise à jour et sa version.
  - « usedUpdatedRules » : contient l'intégralité des règles en cours d'utilisation dont une mise à jour a été effectuée. Chaque détail précise en plus la date de création de la règle, sa dernière mise à jour et sa version.

5.3.3.1 Exemples

Exemple 1 : import initial d'un référentiel

Le rapport généré est :

```
{"Operation":{"evType":"STP_IMPORT_RULES","evDateTime":"2017-11-02T13:50:22.389"},
↪"error":{},"usedDeletedRules":[],"usedUpdatedRules":[]}
```

Exemple 2 : mise à jour d'un référentiel existant

Dans cette exemple, la mise à jour :

- Essaye de modifier une RuleType d'une règle en lui mettant « AccessRules » au lieu de « AccessRule »

- Met à jour une règle de gestion en cours d'utilisation

Le rapport généré est :


```
{
      "Operation": {
              "evType": "STP_IMPORT_RULES",
              "evDateTime": "2017-11-02T14:03:53.326"
      },
      "error": {
              "line 6": [{
                      "Code": "STP_IMPORT_RULES_WRONG_RULETYPE_UNKNOW.KO",
                      "Message": "Au moins une RuleType est incorrecte. RuleType␣
↪autorisés : AppraisalRule, AccessRule, StorageRule, DisseminationRule, ReuseRule,␣
↪ClassificationRule",
                      "Information additionnelle": "AccessRulez"
              }]
      },
      "usedDeletedRules": [],
      "usedUpdatedRules": ["id=null, tenant=0, ruleId=APP-00001,␣
↪ruleType=AppraisalRule, ruleValue=Dossier individuel d'agent civil,␣
↪ruleDescription=Durée de conservation des dossiers individuels d'agents. L'échéance␣
↪est calculée à partir de la date de naissance de l'agent, ruleDuration=70,␣
↪ruleMeasurement=YEAR, creationDate=2017-11-02T14:03:52.374, updateDate=2017-11-
↪02T14:03:52.374, version=0"]
}
```


5.4 Workflow d'import d'un référentiel des formats

5.4.1 Introduction

Cette section décrit le processus (workflow) permettant d'importer un référentiel des formats

5.4.2 Processus d'import d'un référentiel de formats (vision métier)

Le processus d'import du référentiel des formats vise à contrôler que les informations sont formalisées de la bonne manière dans le fichier soumis à la solution logicielle Vitam et que chaque format contient bien le nombre d'informations minimales attendues. Tous les éléments réalisés au cours de ce processus sont exécutés dans une seule étape.

5.4.2.1 Import d'un référentiel de formats STP_REFERENTIAL_FORMAT_IMPORT (ReferentialFormatFileImpl)

- Vérification du fichier de référentiel des formats
  - **Type** : bloquant
  - **Règle** : le fichier doit être au format xml et respecter le formalisme du référentiel PRONOM publié par the National Archives (UK)
  - **Statuts** :
    - OK : les informations correspondant à chaque format sont décrites comme dans le référentiel PRONOM (STP_REFERENTIAL_FORMAT_IMPORT.OK=Succès du processus d'import du référentiel des formats)
    - KO : la condition ci-dessus n'est pas respectée (STP_REFERENTIAL_FORMAT_IMPORT.KO=Échec du processus d'import du référentiel des formats)

- FATAL : une erreur fatale est survenue lors de l'import du référentiel des formats (STP_REFERENTIAL_FORMAT_IMPORT.FATAL=Erreur fatale lors du processus d'import du référentiel des formats)

5.5 Workflow d'administration d'un référentiel des services agent

5.5.1 Introduction

Cette section décrit le processus (workflow) permettant d'importer un référentiel de services agents

5.5.2 Processus d'import et mise à jour d'un référentiel de services agents (STP_IMPORT_AGENCIES)

L'import d'un référentiel de services agent permet de vérifier le formalisme de ce dernier, notamment que les données obligatoires sont bien présentes pour chacun des agents. Tous les éléments réalisés au cours de ce processus sont exécutés dans une seule étape. Cet import concerne aussi bien l'import initial (pas de services agents pré-existant) que la mise à jour du référentiel.

5.5.2.1 Import d'un référentiel de services agents STP_IMPORT_AGENCIES (AgenciesService.java)

- **Règle** : le fichier remplit les conditions suivantes :
  - il est au format CSV
  - les informations suivantes sont toutes décrites dans l'ordre exact pour chacun des services agents :
  - Identifier
  - Name
  - Description (optionnel)
  - l'identifiant doit être unique
- **Type** : bloquant
- **Status** :
  - OK : le fichier respecte les règles (STP_IMPORT_AGENCIES.OK=Succès du processus d'import du référentiel des services agents)
  - KO :
    - Cas 1 : une information concernant les services agents est manquante (Identifier, Name, Description) (STP_IMPORT_AGENCIES.KO=Échec du processus d'import du référentiel des services agents). De plus le rapport d'import du référentiel contiendra la clé « STP_IMPORT_AGENCIES_MISSING_INFORMATIONS ».
    - Cas 2 : un service agent qui était présent dans la base a été supprimé (STP_IMPORT_AGENCIES.DELETION.KO=Échec du processus d'import du référentiel des services agents : Des services agents supprimés sont présents dans le référentiel des services agents)
  - FATAL : une erreur fatale est survenue lors de l'import du référentiel des services agents (STP_IMPORT_AGENCIES.FATAL=Erreur fatale lors du processus d'import du référentiel des service agents)

5.5.2.2 Vérification des contrats utilisés STP_IMPORT_AGENCIES.USED_CONTRACT

- **Règle** : contrôle des contrats utilisant des services agents modifiés
- **Type** : bloquant
- **Status** :
  - OK : aucun des services agents utilisés par des contrats d'accès n'a été modifié (STP_IMPORT_AGENCIES.USED_CONTRACT.OK=Succès du processus de vérification des services agents utilisés dans les contrats d'accès)
  - WARNING : un ou plusieurs services agents utilisés par des contrats d'accès ont été modifiés (STP_IMPORT_AGENCIES.USED_CONTRACT.WARNING=Avertissement lors du processus de vérification des services agents utilisés dans les contrats d'accès)
  - KO : pas de cas KO
  - FATAL : une erreur fatale est survenue lors de la vérification des services agents utilisés dans les contrats d'accès (STP_IMPORT_AGENCIES.USED_CONTRACT.FATAL=Erreur fatale lors du processus de vérification des services agents utilisés dans les contrats d'accès)

5.5.2.3 Vérification des unités archivistiques STP_IMPORT_AGENCIES.USED_AU

- **Règle** : contrôle des unités archivistiques référençant des serivces agents modifiés
- **Type** : bloquant
- **Status** :
  - OK : aucun service agent référencé par les unités archivistiques n'a été modifié (STP_IMPORT_AGENCIES.USED_AU.OK=Succès du processus de vérification des services agents utilisés par les unités archivistiques)
  - WARNING : au moins un service agent référencé par une unité archivistique a été modifié (STP_IMPORT_AGENCIES.USED_AU.WARNING=Avertissement lors du processus de vérification des services agents utilisés par les unités archivistiques)
  - KO : pas de cas KO
  - FATAL : une erreur fatale est survenue lors de la vérification des services agents utilisés par les unités archivistiques (STP_IMPORT_AGENCIES.USED_AU.FATAL=Erreur fatale lors du processus de vérification des services agents utilisés par les unités archivistiques)

5.5.2.4 Création du rapport au format JSON STP_AGENCIES_REPORT (AgenciesService.java)

- **Règle** : création du rapport d'import de référentiel des services agent
- **Type** : bloquant
- **Status** :
  - OK : le rapport d'import du référentiel des services agents a bien été créé (STP_AGENCIES_REPORT.OK=Succès du processus de génération du rapport d'import du référentiel des services agents)
  - KO : pas de cas KO
  - FATAL : une erreur fatale est survenue lors de la création du rapport d'import du référentiel des services agents (STP_AGENCIES_REPORT.FATAL=Erreur fatale lors du processus de génération du rapport d'import du référentiel des services agents)

5.5.2.5 Sauvegarde du CSV d'import STP_IMPORT_AGENCIES_BACKUP_CSV (AgenciesService.java)

- **Règle** : sauvegarde de fichier d'import de référentiel des services agent
- **Type** : bloquant
- **Status** :
  - OK : le fichier d'import du référentiel des services agent a bien été sauvegardé (STP_IMPORT_AGENCIES_BACKUP_CSV.OK=Succès du processus de sauvegarde du fichier d'"import de référentiel des services agents)
  - KO : pas de cas KO
  - FATAL : une erreur fatale est survenue lors de la sauvegarde de fichier d'import de référentiel des services agent (STP_AGENCIES_REPORT.FATAL=Erreur fatale lors du processus de sauvegarde du fichier d'"import de référentiel des services agents)

5.5.2.6 Sauvegarde d'une copie de la base de donnée STP_BACKUP_AGENCIES (AgenciesService.java)

- **Règle** : création d'une copie de la base de données contenant le référentiel des services agents
- **Type** : bloquant
- **Status** :
  - OK : la copie de la base de donnée contenant le référentiel des services agents a été crée avec succès (STP_BACKUP_AGENCIES.OK = Succès du processus de sauvegarde du référentiel des services agents)
  - KO : pas de cas KO
  - FATAL : une erreur fatale est survenue lors de la création d'une copie de la base de données contenant le référentiel des services agent (STP_BACKUP_AGENCIES.FATAL = Erreur fatale lors du processus de sauvegarde du référentiel des services agents)

5.5.3 Structure du rapport d'administration du référentiel des services agents

Lorsqu'un nouveau référentiel est importé, la solution logicielle Vitam génère un rapport de l'opération. Ce rapport est en plusieurs parties :

- « Operation » contient :
  - evType : le type d'opération. Dans le cadre de ce rapport, il s'agit toujours de « STP_IMPORT_AGENCIES »
  - evDateTime : la date et l'heure de l'opération d'import
  - evId : l'identifiant de l'opération
- « AgenciesToImport » : contient la liste des identifiants contenue dans le fichier
- « InsertAgencies » : contient les identifiants des services agents ajoutés
- « UpdatedAgencies » : liste les identifiants des services agents modifiés
- « UsedAgencies By Contrat » : liste les identifiants des services agents modifiés qui sont utilisés par des contrats d'accès
- « UsedAgencies By AU » : liste les identifiants des services agents modifiés qui sont utilisés dans des unités archivistiques
- « UsedAgencies to Delete » : liste les identifiants des services agents supprimés qui sont utilisés dans des unités archivistiques

Exemple 1 : modification et ajout d'un service agent

Le rapport généré est :

```
{
      "Operation": {
              "evType": "STP_IMPORT_AGENCIES",
              "evDateTime": "2017-11-02T15:28:34.523",
              "evId": "aecaaaaaacevq6lcaamxsak7pvmsdbqaaaaq"
      },
      "InsertAgencies": ["Identifier1"],
      "UpdatedAgencies": ["Identifier0"],
      "UsedAgencies By Contrat": ["Identifier0"],
      "UsedAgencies By AU": []
}
```


Exemple 2 : ajout en erreur d'un service agent, causé par un champ obligatoire qui est manquant

Le rapport généré est :


```
{
      "Operation": {
    "evId":"aecaaaaaacflvhgbabrs6alb6vdoehyaaaaq",
              "evType": "STP_IMPORT_AGENCIES",
              "evDateTime": "2017-11-02T15:36:03.976"
      },
      "AgenciesToImport": ["AG-TNR0002"],
      "UsedAgencies to Delete":["AG-TNR0002"]
}
```


5.6 Workflow d'administration d'un référentiel des contrats d'entrée

5.6.1 Introduction

Cette section décrit le processus (workflow) permettant d'importer un contrat d'entrée.

5.6.2 Processus d'import et mise à jour d'un contrat d'entrée (vision métier)

Le processus d'import d'un contrat d'entrée permet à la fois de vérifier qu'il contient les informations minimales obligatoires, de vérifier la cohérence de l'ensemble des informations, et de lui affecter des élements peuplés automatiquement.

Tous les élements réalisés au cours de ce processus sont exécutés dans une seule étape.

5.6.2.1 Import d'un contrat d'entrée (STP_IMPORT_INGEST_CONTRACT)

- Vérification de la présence des informations minimales, de la cohérence des informations et affectation des données aux champs peuplés par la solution logicielle Vitam.
  - **Type** : bloquant
  - **Règle** : vérification et enregistrement du contrat :
    - Le champ « Name » est peuplé d'une chaîne de caractères
    - Si le tenant concerné est en mode « esclave », le champ « Identifier » doit être rempli. Sinon, il est automatiquement complété par la solution logicielle Vitam
    - Les données suivantes optionnelles si elles sont remplies le sont en respectant les règles énnonées pour chacune :

- Le champ « Description » est peuplé avec une chaîne de caractères
- Le champ « Status » est peuplé avec de la valeur ACTIVE ou la valeur INACTIVE
- Le champ « ArchiveProfile » est peuplé avec un tableau d'une ou plusieurs chaînes de caractère. Chacune de ces chaînes de caractère doit correspondre au champ « Identifier » d'un profil d'archivage contenu dans le référentiel des profils
- Le champ « CheckParentLink » : est peuplé avec de la valeur ACTIVE ou la valeur INACTIVE
- Le champ « LinkedParentId » est peuplé par une chaîne de caractères devant correspondre au GUID d'une AU de plan de classement ou d'arbre de positionnement pris en charge par la solution logicielle Vitam sur le même tenant

- **Statuts** :
  - OK : le contrat répond aux exigences des règles (STP_IMPORT_INGEST_CONTRACT.OK=Succès du processus d'import du contrat d'entrée)
  - KO : une des règles ci-dessus n'a pas été respectée (STP_IMPORT_INGEST_CONTRACT.KO=Échec du processus d'import du contrat d'entrée)
  - FATAL : une erreur fatale est survenue lors de la vérification de l'import du contrat (STP_IMPORT_INGEST_CONTRACT.FATAL=Erreur fatale du processus d'import du contrat d'entrée)
  - WARNING : Avertissement lors du processus d'import du contrat d'"‘entrée (STP_IMPORT_INGEST_CONTRACT.WARNING=Avertissement lors du processus d'import du contrat d'entrée)
  - DUPLICATION : L'identifiant utilisé existe déjà (STP_IMPORT_INGEST_CONTRACT.IDENTIFIER_DUPLICATIO de l'import : l'identifiant est déjà utilisé)
  - EMPTY REQUIRED FIELD : un champ obligatoire n'est pas renseigné (STP_IMPORT_INGEST_CONTRACT.EMPTY_REQUIRED_FIELD.KO=Échec de l'import : au moins un des champs obligatoires n'est pas renseigné)
  - PROFILE NOT FOUND : Le profil d'archivage mentionné n" existe pas (STP_IMPORT_INGEST_CONTRACT.PROFILE_NOT_FOUND.KO=Échec de l'import : profil d'archivage non trouvé)

5.6.2.2 Mise à jour d'un contrat d'entrée (STP_UPDATE_INGEST_CONTRACT)

La modification d'un contrat d'entrée doit suivre les mêmes règles que celles décrites pour la création. La clé de l'événement est « STP_UPDATE_INGEST_CONTRACT », entraînant des statuts STP_UPDATE_INGEST_CONTRACT.OK, STP_UPDATE_INGEST_CONTRACT.KO et STP_UPDATE_INGEST_CONTRACT.FATAL sur les mêmes contrôles que l'import.

5.6.2.3 Sauvegarde du JSON (STP_BACKUP_INGEST_CONTRACT)

Cette tâche est appellée que ce soit en import initial ou en modification.

- **Règle** : enregistrement d'une copie de la base de données des contrats d'entrée sur le stockage
- **Type** : bloquant
- **Statuts** :
  - OK : une copie de la base de donnée nouvellement importée est enregistrée (STP_BACKUP_INGEST_CONTRACT.OK = Succès du processus de sauvegarde des contrats d'entrée)
  - KO : pas de cas KO
  - FATAL : une erreur fatale est survenue lors de la copie de la base de donnée nouvellement importée (STP_BACKUP_INGEST_CONTRACT.FATAL = Erreur fatale lors du processus de sauvegarde des contrats d'entrée)

5.7 Workflow d'administration d'un référentiel des contrats d'accès

5.7.1 Introduction

Cette section décrit le processus (workflow) permettant d'importer un contrat d'accès.

5.7.2 Processus d'import et mise à jour d'un contrat d'accès (vision métier)

Le processus d'import d'un contrat d'accès permet à la fois de vérifier qu'il contient les informations minimales obligatoires, de vérifier la cohérence de l'ensemble des informations et de lui affecter des éléments peuplés automatiquement.

Tous les éléments réalisés au cours de ce processus sont exécutés dans une seule étape.

5.7.2.1 Import d'un contrat d'accès (STP_IMPORT_ACCESS_CONTRACT)

- Vérification de la présence des informations minimales obligatoires, de la cohérence des informations et affecter des données aux champs peuplés par la solution logicielle Vitam.
  - **Type** : bloquant
  - **Règle** : vérification et enregistrement du contrat
  - Les données suivantes sont obligatoirement remplies :
    - Le champ « Name » est peuplé d'une chaîne de caractères
    - Le champ « Identifier » est peuplé d'une chaîne de caractères si le référentiel des contrats d'accès est configuré en mode esclave sur le tenant sélectionné
  - Les données suivantes optionnelles, si elles sont remplies, le sont en respectant les règles énoncées pour chacune :
    - Le champ « Description » est peuplé avec une chaîne de caractères
    - Le champ « Status » est peuplé avec la valeur ACTIVE ou INACTIVE
    - Le champ « DataObjectVersion » est soit vide, soit peuplé avec un tableau d'une ou plusieurs chaînes de caractères. Chacune de ces chaînes de caractères doit correspondre à un des usages définis dans les groupe d'objets techniques pris en charge dans la solution logicielle Vitam.
    - Le champ « OriginatingAgencies » est soit vide soit peuplé avec un tableau d'une ou plusieurs chaînes de caractères. Chacune de ces chaînes de caractères doit correspondre au champ « Identifier » d'un service agent contenu dans le référentiel des services agents.
    - Le champ « WritingPermission » doit être à « true » ou « false »
    - Le champ « EveryOriginatingAgency » doit être à « true » ou « false »
    - Le champ « EveryDataObjectVersion » doit être à « true » ou « false »
    - Le champ « RootUnit » est soit vide, soit peuplé avec un tableau d'une ou plusieurs chaînes de caractère. Chacune des chaînes de caractère doit correspondre au GUID d'une unité archivistique prise en charge dans la solution logicielle Vitam.
  - **Type** : bloquant
  - **Statuts** :
    - OK : le contrat répond aux exigences des règles (STP_IMPORT_ACCESS_CONTRACT.OK=Succès du processus d'import du contrat d'accès)
    - KO : une des règles ci-dessus n'a pas été respectée (STP_IMPORT_ACCESS_CONTRACT.KO=Échec du processus d'import du contrat d'accès)

- FATAL : une erreur fatale est survenue lors de la vérification de l'import du contrat d'accès (STP_IMPORT_ACCESS_CONTRACT.FATAL=Erreur fatale lors du processus d'import du contrat d'accès)
- WARNING : Avertissement lors du processus d'import du contrat d'accès (STP_IMPORT_ACCESS_CONTRACT.WARNING=Avertissement lors du processus d'import du contrat d'accès)
- DUPLICATION : l'identifiant du contrat est déjà utilisé (STP_IMPORT_ACCESS_CONTRACT.IDENTIFIER_DUPLICATION.KO=Échec de l'import : l'identifiant est déjà utilisé)
- EMPTY REQUIRED FIELD : au moins un des champs obligatoires n'est pas renseigné (STP_IMPORT_ACCESS_CONTRACT.EMPTY_REQUIRED_FIELD.KO=Échec de l'import : au moins un des champs obligatoires n'est pas renseigné)
- AGENCY NOT FOUND : Service producteur inconnu (STP_IMPORT_ACCESS_CONTRACT.AGENCY_NOT_FOU de l'import : au moins un service producteur est inconnu)
- VALIDATION ERROR : Erreur de validation du contrat (STP_IMPORT_ACCESS_CONTRACT.VALIDATION_ERROR.KO=Échec de l'import : erreur de validation du contrat)

5.7.2.2 Mise à jour d'un contrat d'accès STP_UPDATE_ACCESS_CONTRACT (AdminExternalClientRest.java)

La modification d'un contrat d'accès doit suivre les mêmes règles que celles décrites pour la création.

- OK : le contrat répond aux exigences des règles (STP_UPDATE_ACCESS_CONTRACT.OK=Succès du processus de mise à jour du contrat d'accès)
- KO : une des règles ci-dessus n'a pas été respectée (STP_UPDATE_ACCESS_CONTRACT.KO=Échec du processus de mise à jour du contrat d'accès)
- FATAL : une erreur fatale est survenue lors de la vérification de l'import du contrat d'accès (STP_UPDATE_ACCESS_CONTRACT.FATAL=Erreur fatale lors du processus de mise à jour du contrat d'"“accès)

5.7.2.3 Sauvegarde du JSON STP_BACKUP_INGEST_CONTRACT (IngestContractImpl.java)

Cette tâche est appellée que ce soit en import initial ou en modification.

- **Règle** : enregistrement d'une copie de la base de données des contrats d'accès sur le stockage
- **Type** : bloquant
- **Statuts** :
  - OK : une copie de la base de donnée nouvellement importée est enregistrée (STP_BACKUP_ACCESS_CONTRACT.OK = Succès du processus de sauvegarde des contrats d'accès)
  - KO : pas de cas KO
  - FATAL : une erreur fatale est survenue lors de la copie de la base de donnée nouvellement importée (STP_BACKUP_ACCESS_CONTRACT.FATAL = Erreur fatale lors du processus de sauvegarde des contrats d'accès)

5.8 Workflow d'import d'un référentiel des profils

5.8.1 Introduction

Cette section décrit le processus (workflow) permettant d'importer un profil d'archivage.

5.8.2 Processus d'import et mise à jour d'un profil (vision métier)

Le processus d'import d'un profil d'archivage permet à la fois de vérifier qu'il contient les informations minimales obligatoires, de vérifier la cohérence de l'ensemble des informations, et de lui affecter des élements peuplés automatiquement.

Tous les éléments réalisés au cours de ce processus sont exécutés dans une seule étape.

5.8.2.1 Import des métadonnées d'un profil d'archivage STP_IMPORT_PROFILE_JSON (ProfileServiceImpl.java)

- **Règle** : le profil d'archivage répond aux exigences suivantes :

- Vérification de la présence des informations minimales, de la cohérence des informations et affectation des données aux champs peuplés par la solution logicielle Vitam.
  - Les données suivantes sont obligatoirement remplies :
    - Le champ « Name » est peuplé d'une chaîne de caractères
    - Le champs « Identifier » est peuplé d'une chaîne de caractère si le référentiel des profils d'archivage est configuré en mode esclave sur le tenant séléctionné
    - Le champ « Format » doit être renseigné avec la valeur RNG ou XSD
  - Les données suivantes optionnelles si elles sont remplies le sont en respectant les règles énoncées pour chacune :
    - Le champ « Description » est peuplé avec une chaîne de caractères
    - Le champ « Status » est peuplé avec la valeur ACTIVE ou la valeur INACTIVE
  - **Type** : bloquant
  - **Statuts** :
    - OK : les règles ci-dessus sont respectées (STP_IMPORT_PROFILE_JSON.OK=Succès du processus d'import du profil d'archivage)
    - KO : une des règles ci-dessus n'a pas été respectée (STP_IMPORT_PROFILE_JSON.KO=Échec du processus d'import du profil d'archivage)
    - FATAL : une erreur fatale est survenue lors de la vérification de l'import du profil d'archivage (STP_IMPORT_PROFILE_JSON.FATAL=Erreur fatale lors du processus d'import du profil d'archivage)
    - WARNING : Avertissement lors du processus d'import du profil d'archivage (STP_IMPORT_PROFILE_JSON.WARNING=Avertissement lors du processus d'import du profil) d'archivage
    - IDENTIFIER DUPLICATION : L'identifiant est déjà utilisé (STP_IMPORT_PROFILE_JSON.IDENTIFIER_DUPLICATION.KO=Échec de l'import : l'identifiant est déjà utilisé)
    - EMPTY REQUIRED FIELD : Au moins un des champs obligatoires n'est pas renseigné (STP_IMPORT_PROFILE_JSON.EMPTY_REQUIRED_FIELD.KO=Échec de l'import : au moins un des champs obligatoires n'est pas renseigné)

5.8.2.2 Import du profil d'archivage STP_IMPORT_PROFILE_FILE (ProfileServiceImpl.java)

- Vérification de la concordance entre le fichier importé dans un profil et le format décrit dans la métadonnée « Format »
  - **Type** : bloquant

- **Règle** : le format du fichier doit être celui décrit dans le profil
- **Statuts** :
  - OK : le fichier importé est au même format que celui décrit dans le champ « Format » (STP_IMPORT_PROFILE_FILE.OK=Succès du processus d'import du profil d'archivage (fichier xsd ou rng)
  - KO : le fichier importé n'est pas au même format que celui décrit dans le champ « Format » (STP_IMPORT_PROFILE_FILE.KO=Échec du processus d'import du profil d'archivage (fichier xsd ou rng))
  - FATAL : une erreur fatale est survenue lors de la vérification de l'import du profil d'archivage (STP_IMPORT_PROFILE_FILE.FATAL=Erreur fatale lors du processus d'import du profil d'archivage (fichier xsd ou rng))

5.8.2.3 Mise à jour d'un profil d'archivage STP_UPDATE_PROFILE_JSON (ProfileServiceImpl.java)

La modification d'un profil d'archivage doit suivre les mêmes règles que celles décrites pour la création. L'association d'un fichier de profil avec les métadonnées d'un profil provoque également une opération de mise à jour du profil d'archivage. La modification doit suivre les mêmes règles que celles décrites pour la création. La clé de l'événement est « STP_UPDATE_PROFILE_JSON », entraînant des statuts STP_UPDATE_PROFILE_JSON.OK, STP_UPDATE_PROFILE_JSON.KO et STP_UPDATE_PROFILE_JSON.FATAL sur les mêmes contrôles que l'import.

5.8.2.4 Sauvegarde du JSON BACKUP_PROFILE (ProfileServiceImpl.java)

Cette tâche est appellée que ce soit en import initial ou lors de la modification des métadonnées de profils

- **Règle** : enregistrement d'une copie de la base de données des métadonnées de profils sur le stockage
- **Type** : bloquant
- **Statuts** :
  - OK : une copie de la base de donnée nouvellement importée est enregistrée (BACKUP_PROFILE.OK = Succès du processus de sauvegarde des profils)
  - KO : pas de cas KO
  - FATAL : une erreur fatale est survenue lors de la copie de la base de donnée nouvellement importée (BACKUP_PROFILE.FATAL = Erreur fatale lors du processus de sauvegarde des profils)

5.9 Workflow d'administration d'un référentiel des profils de sécurité

5.9.1 Introduction

Cette section décrit le processus (workflow) de création d'un profil de sécurité

5.9.2 Processus d'import et mise à jour d'un profil de sécurité

Le processus d'import d'un profil de sécurité permet à la fois de vérifier qu'il contient les informations minimales obligatoires, de vérifier la cohérence de l'ensemble des informations, et de lui affecter des élements peuplés automatiquement.

Tous les éléments réalisés au cours de ce processus sont exécutés dans une seule étape.

5.9.2.1 Import d'un profil de sécurité STP_IMPORT_SECURITY_PROFILE (SecurityProfileService.java)

- **Règle** : vérification et enregistrement du profil de sécurité. les données suivantes sont obligatoirement remplies :
  - Le champ « Name » doit être peuplé avec une chaîne de caractères unique
  - Le champ « Identifier » doit être unique
  - Le champ « FullAccess » doit être à « true » ou « false »
- **Type** : bloquant
- **Statuts** :
  - OK : les règles ci-dessus sont respectées (STP_IMPORT_SECURITY_PROFILE.OK=Succès du processus d'import du profil de sécurité)
  - KO : une des règles ci-dessus n'est pas respectée (STP_IMPORT_SECURITY_PROFILE.KO=Échec du processus d'import du profil de sécurité)
  - FATAL : une erreur fatale est survenue lors de l'import du profil de sécurité (STP_IMPORT_SECURITY_PROFILE.FATAL=Erreur fatale lors du processus d'import du profil de sécurité)

5.9.2.2 Mise à jour d'un profil de sécurité STP_UPDATE_SECURITY_PROFILE (SecurityProfileService.java)

La modification d'un profil de sécurité doit suivre les mêmes règles que celles décrites pour la création.

- OK : les règles ci-dessus sont respectées (STP_UPDATE_SECURITY_PROFILE.OK=Succès du processus de mise à jour du profil de sécurité)
- KO : une des règles ci-dessus n'est pas respectée (STP_UPDATE_SECURITY_PROFILE.KO=Échec du processus de mise à jour du profil de sécurité)
- FATAL : une erreur fatale est survenue lors de l'import du profil de sécurité (STP_UPDATE_SECURITY_PROFILE.FATAL=Erreur fatale lors du processus de mise à jour du profil de sécurité)

5.9.2.3 Sauvegarde du JSON STP_BACKUP_SECURITY_PROFILE (SecurityProfileService.java)

Cette tâche est appellée que ce soit en import initial ou en modification.

- **Règle** : enregistrement d'une copie de la base de données des profils de sécurité sur le stockage
- **Type** : bloquant
- **Statuts** :
  - OK : une copie de la base de donnée nouvellement importée est enregistrée (STP_BACKUP_SECURITY_PROFILE.OK = Succès du processus de sauvegarde des profils de sécurité)
  - KO : pas de cas KO
  - FATAL : une erreur fatale est survenue lors de la copie de la base de donnée nouvellement importée (STP_BACKUP_SECURITY_PROFILE.FATAL = Erreur fatale lors du processus de sauvegarde des profils de sécurité)

5.10 Workflow d'administration d'un référentiel des contextes

5.10.1 Introduction

Cette section décrit le processus (workflow) d'import des contextes dans le référentiel des contextes. Cette opération n'est réalisable que sur le tenant administration.

5.10.2 Processus d'import et mise à jour d'un référentiel des contextes applicatifs

Le processus d'import d'un référentiel des contextes applicatifs permet à la fois de vérifier qu'il contient les informations minimales obligatoires, de vérifier la cohérence de l'ensemble des informations, et de lui affecter des élements peuplés automatiquement.

Tous les éléments réalisés au cours de ce processus sont exécutés dans une seule étape.

5.10.2.1 Import d'un référentiel des contextes STP_IMPORT_CONTEXT (ContextServiceImpl.java)

- Vérification de la présence des informations minimales, de la cohérence des informations et affecter des données aux champs peuplés par la solution logicielle Vitam.
  - **Règle** : vérification et enregistrement du référentiel des contextes :
    - Le champ « Name » doit être peuplé avec une chaîne de caractères unique
    - Le champ « Status » doit être à « true » ou « false »
    - Le champ « EnableControl » doit être à « true » ou « false »
    - Le champ « Permissions » doit être peuplé avec un tableau contenant des fichiers JSON
    - Le champ « SecurityProfile » et « doit être peuplé avec une chaîne de caractères
    - Le champ « Identifier » doit être unique
  - **Type** : bloquant
  - **Statuts** :
    - OK : Les règles ci-dessus sont respectées (STP_IMPORT_CONTEXT.OK = Succès du processus d'import du contexte)
    - KO : une des règles ci-dessus n'est pas respectée (STP_IMPORT_CONTEXT.KO=Échec du processus d'import du contexte)
    - FATAL : une erreur fatale est survenue lors de l'import du contexte (STP_IMPORT_CONTEXT.FATAL=Erreur fatale lors du processus d'import du contexte)
    - IDENTIFIER DPLICATION : L'identifiant est déjà utilisé (STP_IMPORT_CONTEXT.IDENTIFIER_DUPLICATION.KO=Echec de l'import : l'identifiant est déjà utilisé)
    - EMPTY REQUIRED FIELD : Un des champs obligatoires n'est pas renseigné (STP_IMPORT_CONTEXT.EMPTY_REQUIRED_FIELD.KO=Echec de l'import : au moins un des champs obligatoires n'est pas renseigné)
    - SECURITY PROFILE NOT FOUND : Le profil de sécurité mentionné est inconnu du système (STP_IMPORT_CONTEXT.SECURITY_PROFILE_NOT_FOUND.KO=Echec de l'import : profil de sécurité non trouvé)
    - UNKNOWN VALUE : Au moins un objet déclare une valeur inconnue (STP_IMPORT_CONTEXT.UNKNOWN_VALUE.KO=Echec de l'import : au moins un objet déclare une valeur inconnue)

5.10.2.2 Mise à jour d'un contexte applicatif STP_UPDATE_CONTEXT (ContextServiceImpl.java)

- OK : Les règles ci-dessus sont respectées (STP_UPDATE_CONTEXT.OK=Succès du processus de mise à jour du contexte)
- KO : une des règles ci-dessus n'est pas respectée (STP_UPDATE_CONTEXT.KO=Échec du processus mise à jour du contexte)
- FATAL : une erreur fatale est survenue lors de l'import du contexte (STP_UPDATE_CONTEXT.FATAL=Erreur fatale lors du processus de mise à jour du contexte)

5.10.2.3 Sauvegarde du JSON STP_BACKUP_CONTEXT (ContextServiceImpl.java)

Cette tâche est appellée que ce soit en import initial ou en modification.

- **Règle** : enregistrement d'une copie de la base de données des contextes applicatifs sur le stockage
- **Type** : bloquant
- **Statuts** :
  - OK : une copie de la base de donnée nouvellement importée est enregistrée (STP_BACKUP_CONTEXT.OK = Succès du processus de sauvegarde des contextes)
  - KO : pas de cas KO
  - FATAL : une erreur fatale est survenue lors de la copie de la base de donnée nouvellement importée (STP_BACKUP_CONTEXT.FATAL = Erreur fatale lors du processus de sauvegarde des contextes)

5.11 Workflow d'import d'un référentiel des documents types

5.11.1 Introduction

Cette section décrit le processus (workflow) permettant d'importer un documents type (Profil d'unité archivistique).

5.11.2 Processus d'import et mise à jour d'un document type (Profil d'unité archivistique) (vision métier)

Le processus d'import d'un document type (profil d'unité archivistique) permet à la fois de vérifier qu'il contient les informations minimales obligatoires, de vérifier la cohérence de l'ensemble des informations, et de lui affecter des élements peuplés automatiquement.

Tous les éléments réalisés au cours de ce processus sont exécutés dans une seule étape.

5.11.2.1 Import des métadonnées d'un document type (profil d'unité archivistique) IMPORT_ARCHIVEUNITPROFILE (ArchiveUnitProfileServiceImpl.java)

- Vérification de la présence des informations minimales, de la cohérence des informations et affectation des données aux champs peuplés par la solution logicielle Vitam.
  - **Règle** : le document type (Profil d'unité archivistique) répond aux exigences suivantes :
    - Les données suivantes sont obligatoirement remplies :
      - Le champ « Name » est peuplé d'une chaîne de caractères
      - Le champs « Identifier » est peuplé d'une chaîne de caractère si le référentiel des documents type est configuré en mode esclave sur le tenant séléctionné

    - Les données suivantes optionnelles si elles sont remplies le sont en respectant les règles énoncées pour chacune :
        - Le champ « Description » est peuplé avec une chaîne de caractères
        - Le champ « Status » est peuplé avec la valeur ACTIVE ou la valeur INACTIVE
        - Le champ « CreationDate » est peuplé avec une valeur correspondant à une date au format : JJ/MM/AA
        - Le champ « ActivationDate » est peuplé avec une valeur correspondant à une date au format : JJ/MM/AA
        - Le champ « DeactivationDate » est peuplé avec une valeur correspondant à une date au format : JJ/MM/AA
- **Type** : bloquant
- **Statuts** :
    - OK : les règles ci-dessus sont respectées (IMPORT_ARCHIVEUNITPROFILE.OK=Succès du processus d'import du document type)
    - KO : une des règles ci-dessus n'a pas été respectée (IMPORT_ARCHIVEUNITPROFILE.KO=Échec du processus d'import du document type)
    - FATAL : une erreur fatale est survenue lors de la vérification de l'import du document type (IMPORT_ARCHIVEUNITPROFILE.FATAL=Erreur fatale lors du processus d'import du du document type)
    - STARTED : Début du processus d'import du document type (IMPORT_ARCHIVEUNITPROFILE.STARTED=Début du processus d'import du document type)
    - WARNING : Avertissement lord du processus d'import du document type (IMPORT_ARCHIVEUNITPROFILE.WARNING=Avertissement lors du processus d'import du document type)
    - IDENTIFIER DUPLICATION : L'identifiant est déjà utilisé (IMPORT_ARCHIVEUNITPROFILE.IDENTIFIER_DUPLICATION.KO=Echec de l'import : l'identifiant est déjà utilisé)
    - EMPTY REQUIRED FIELD : Au moins un des champs obligatoires n'est pas renseigné (IMPORT_ARCHIVEUNITPROFILE.EMPTY_REQUIRED_FIELD.KO=Echec de l'import : au moins un des champs obligatoires n'est pas renseigné)
    - INVALID JSON SCHEMA : Schéma JSON invalide (IMPORT_ARCHIVEUNITPROFILE.INVALID_JSON_SCHEMA.KO=Echec de l'import : schéma JSON non valide)

5.11.2.2 Mise à jour d'un document type (Profil d'unité archivistique) UPDATE_ARCHIVEUNITPROFILE (ArchiveUnitProfileManager.java)

- OK : les règles ci-dessus sont respectées (UPDATE_ARCHIVEUNITPROFILE.OK=Succès du processus de mise à jour du document type)
- KO : une des règles ci-dessus n'a pas été respectée (UPDATE_ARCHIVEUNITPROFILE.KO=Échec du processus d'import du document type)
- FATAL : une erreur fatale est survenue lors de la vérification de l'import du document type (UPDATE_ARCHIVEUNITPROFILE.FATAL=Erreur fatale lors du processus de mise à jour du document type)

5.11.2.3 Sauvegarde du JSON BACKUP_ARCHIVEUNITPROFILE (ArchiveUnitProfileManager.java)

Cette tâche est appellée que ce soit en import initial ou lors de la modification des métadonnées de document type.

- **Règle** : enregistrement d'une copie de la base de données des métadonnées sur le stockage
- **Type** : bloquant
- **Statuts** :
  - OK : une copie de la base de donnée nouvellement importée est enregistrée (BACKUP_ARCHIVEUNITPROFILE.OK = Succès du processus de sauvegarde des document types)
  - KO : Echech du processus de sauvegarde du document type (BACKUP_ARCHIVEUNITPROFILE.KO = Echec du processus de sauvegarde des document types)

5.12 Workflow d'import d'un référentiel des vocabulaires de l'ontologie

5.12.1 Introduction

Cette section décrit le processus permettant d'importer des vocabulaires de l'ontologie. Cette opération n'est réalisable que sur le tenant d'administration.

5.12.2 Processus d'import et mise à jour des vocabulaires de l'ontologie (vision métier)

Le processus d'import d'une ontologie permet d'ajouter des vocabulaires qui seront utilisés dans les documents types (Profil d'unité archivistique).

Tous les éléments réalisés au cours de ce processus sont exécutés dans une seule étape.

5.12.2.1 Import des métadonnées d'une ontologie IMPORT_ONTOLOGY (OntologyServiceImpl.java)

- Vérification de la présence des informations minimales, de la cohérence des informations et affectation des données aux champs peuplés par la solution logicielle Vitam.
  - **Règle** : l'ontologie répond aux exigences suivantes :
    - Le fichier est au format Json.
    - Les données suivantes sont obligatoires :
      - Le champ « Identifier » est peuplé d'une chaîne de caractères
      - **Le champ « Type » est peuplé par une valeur comprise dans la liste :**
        - Text
        - Keyword
        - Date
        - Long
        - Double
        - Boolean
        - Geo-point
        - Enumération de valeur

- Le champ « Origin » est peuplé par la valeur « EXTERNAL » ou « INTERNAL ». (L'INTERNAL correspond à l'ontologie interne de VITAM embarquée dans la solution)
- Les données suivantes sont facultatives :
  - Le champ « SedaField » est peuplé d'une chaîne de caractères
  - Le champ « ApiField » est peuplé d'une chaîne de caractères
  - Le champ « Description » est peuplé d'une chaîne de caractères
  - Le champ « ShortName » correspond au champ traduction, il est peuplé par une chaîne de valeur
  - Le champ « Collections » indique la collection dans laquelle le vocabulaire est rataché. ex : « Unit »

  Exemple ontologie :

```
[ {
"Identifier" : "AcquiredDate",
"SedaField" : "AcquiredDate",
"ApiField" : "AcquiredDate",
"Description" : "unit-es-mapping.json",
"Type" : "DATE",
"Origin" : "INTERNAL",
"ShortName" : "AcquiredDate",
"Collections" : [ "Unit" ]
}, {
"Identifier" : "BirthDate",
"SedaField" : "BirthDate",
"ApiField" : "BirthDate",
"Description" : "unit-es-mapping.json",
"Type" : "DATE",
"Origin" : "INTERNAL",
"ShortName" : "BirthDate",
"Collections" : [ "Unit" ]
}]
```

- **Statuts** :
  - OK : les règles ci-dessus sont respectées (IMPORT_ONTOLOGY.OK=Succés du processus d'import de l'ontologie)
  - KO : une des règles ci-dessus n'a pas été respectée (IMPORT_ONTOLOGY.KO=Echec du processus d'import de l'ontologie)
  - FATAL : une erreur fatale est survenue lors de la vérification de l'import de l'ontologie (IMPORT_ONTOLOGY.FATAL=Erreur fatale lors du processus d'import de l'ontologie)
  - WARNING : Avertissement lors du processus d'import de l'ontologie (IMPORT_ONTOLOGY.WARNING=Avertissement lors du processus d'import de l'ontologie)

5.12.2.2 Mise à jour d'une ontologie

La modification d'une ontologie s'effectue par ré-import du fichier Json. Le nouvel import annule et remplace l'ontologie précédente. Ce ré-import observe les règles décrites dans le processus d'import, décrit plus haut.

Note : la mise à jour des vocabulaires de l'ontologie doit respecter certaines règles de compatibilité concernant la valeur du « Type » :

- Le champ Type Text peut être modifié en Keyword, Text
- Le champ Type Keyword peut être modifié en Keyword, Text

- Le champ Type Date peut être modifié en Keyword, Text
- Le champ Type Long peut être modifié en Keyword, Text, Double
- Le champ Type Double peut être modifié en Keyword, Text
- Le champ Type Boolean peut être modifié en Keyword, Text
- Le champ Type Geo-point peut être modifié en Keyword, Text
- Le champ Type Enumération de valeur peut être modifié en Keyword, Text
- **Statuts** :
  - OK : les règles ci-dessus sont respectées (IMPORT_ONTOLOGY.OK=Succés du processus d'import de l'ontologie)
  - KO : une des règles ci-dessus n'a pas été respectée (IMPORT_ONTOLOGY.KO=Echec du processus d'import de l'ontologie)
  - FATAL : une erreur fatale est survenue lors de la vérification de l'import de l'ontologie (IMPORT_ONTOLOGY.FATAL=Erreur fatale lors du processus d'import de l'ontologie)
  - WARNING : Avertissement lors du processus d'import de l'ontologie (IMPORT_ONTOLOGY.WARNING=Avertissement lors du processus d'import de l'ontologie)

5.12.2.3 Sauvegarde du JSON (BACKUP_ONTOLOGY)

Cette tâche est appellée en import de l'ontologie.

- **Règle** : enregistrement d'une copie de la base de données des métadonnées sur le stockage
- **Type** : bloquant
- **Statuts** :
  - OK : une copie de la base de donnée nouvellement importée est enregistrée (BACKUP_ONTOLOGY.OK=Succés du processus de sauvegarde des ontologies)
  - KO : Echec du processus de sauvegarde de l'ontologie (BACKUP_ONTOLOGY.KO=Echec du processus de sauvegarde des ontologies)

CHAPITRE 6

TRACEABILITY

6.1 Workflow de création d'un journal sécurisé des groupes d'objets et des unités archivistiques

6.1.1 Introduction

Cette section décrit le processus (workflow) de sécurisation des journaux mis en place dans la solution logicielle Vitam pour les groupes d'objets et les unités archivistiques.

Celui-ci est défini dans le fichier « LogbookAdministration.java » (situé ici : sources/logbook/logbook-administration/src/main/java/fr/gouv/vitam/logbook/administration/core/)

6.1.2 Processus de sécurisation des journaux (vision métier)

Le processus de sécurisation des journaux consiste en la création d'un fichier .zip contenant l'ensemble des journaux à sécuriser, ainsi que le tampon d'horodatage calculé à partir de l'arbre de Merkle de la liste de ces mêmes journaux. Les journaux concernés par cette sécurisation sont le journal des opérations et le journal des écritures.

Ce fichier zip est ensuite enregistré sur les offres de stockage, en fonction de la stratégie de stockage.

6.1.3 Sécurisation du journal des opérations (STP_OP_SECURISATION)

La fin du processus peut prendre plusieurs statuts :

- **Statuts** :
  - OK : le journal des opérations a été sécurisé (STP_OP_SECURISATION.OK = Succès du processus de sécurisation du journal des opérations)
  - KO : pas de cas KO
  - FATAL : une erreur fatale est survenue lors de la sécurisation du journal des opérations (STP_OP_SECURISATION.FATAL = Erreur fatale lors du processus de sécurisation du journal des opérations)

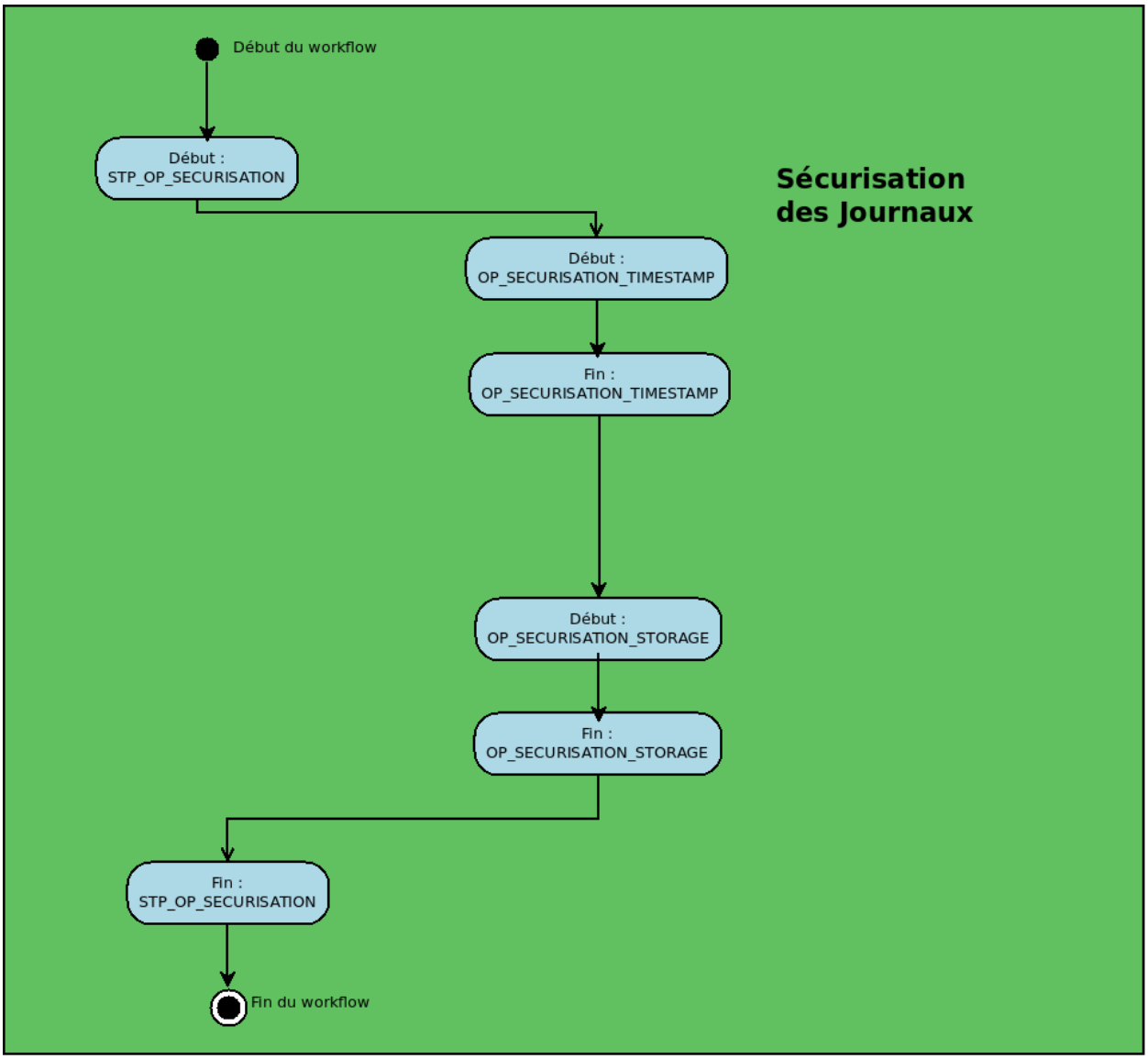
6.1.3.1 OP_SECURISATION_TIMESTAMP (LogbookAdministration.java)

- **Règle** : calcul du tampon d'horodatage à partir de la racine de l'arbre de Merkle consitué de la liste des journaux qui sont en train d'être sécurisés.
- **Type** : bloquant
- **Status :**
    - OK : le tampon d'horodatage est calculé (OP_SECURISATION_TIMESTAMP.OK=Succès de la création du tampon d'horodatage de l'ensemble des journaux)
    - FATAL : une erreur fatale est survenue lors de l'horodatage (OP_SECURISATION_TIMESTAMP.FATAL=Erreur fatale lors de la création du tampon d'horodatage de l'ensemble des journaux)

6.1.3.2 OP_SECURISATION_STORAGE (LogbookAdministration.java)

- **Règle** : écriture des journaux sécurisés sur les offres de stockage, en fonction de la stratégie de stockage.
- **Type** : bloquant
- **Status :**
    - OK : le journal sécurisé est écrit sur les offres de stockage (OP_SECURISATION_STORAGE.OK=Succès de l'enregistrement des journaux sur les offres de stockage)
    - FATAL : une erreur fatale est survenue lors de l'écriture du journal sécurisé (OP_SECURISATION_STORAGE.FATAL=Erreur fatale lors de l'enregistrement des journaux sur les offres de stockage)

D'une façon synthétique, le workflow est décrit de cette façon.



6.2 Workflow de création de journal des cycles de vie sécurisé des groupes d'objets

6.2.1 Introduction

Cette section décrit le processus (workflow) permettant la sécurisation des journaux du cycle de vie mis en place dans la solution logicielle Vitam des groupes d'objets. Le workflow mis en place dans la solution logicielle Vitam est défini dans le fichier « DefaultObjectGroupLifecycleTraceability.json ». Ce fichier est disponible dans : sources/processing/processing-management/src/main/resources/workflows.

Note : Le traitement permettant la sécurisation des journaux du cycle de vie procède par des tranches de lots de 100K. La solution Vitam à la fin de ce premier lot enclenche un autre traitement de 100K et ce jusqu'à avoir traités l'ensemble des groupes d'objets.

6.2.2 Processus de sécurisation des journaux des cycles de vie (vision métier)

Le processus de sécurisation des journaux des cycles de vie consiste en la création d'un fichier .zip contenant l'ensemble des journaux du cycle de vie à sécuriser, ainsi que le tampon d'horodatage.

Ce fichier zip est ensuite enregistré sur les offres de stockage, en fonction de la stratégie de stockage.

6.2.3 Sécurisation des journaux du cycle de vie LOGBOOK_OBJECTGROUP_LFC_TRACEABILITY (LogbookLFCAdministration.java)

La fin du processus peut prendre plusieurs statuts :

- **Statuts** :
  - OK : les journaux du cycle de vie ont été sécurisés (LOGBOOK_OBJECTGROUP_LFC_TRACEABILITY.OK = Succès de la sécurisation des journaux du cycle de vie des groupes d'objets)
  - WARNING : il n'y pas de nouveaux journaux à sécuriser depuis la dernière sécurisation (LOGBOOK_OBJECTGROUP_LFC_TRACEABILITY.WARNING = Avertissement lors de la sécurisation des journaux du cycle de vie des groupes d'objets)
  - KO : pas de cas KO
  - FATAL : une erreur fatale est survenue lors de la sécurisation du journal des opérations (LOGBOOK_OBJECTGROUP_LFC_TRACEABILITY.FATAL = Erreur fatale lors de la sécurisation des journaux du cycle de vie des groupes d'objets)

6.2.3.1 Préparation des listes des cycles de vie

6.2.3.1.1 Étape 1 - STP_PREPARE_OG_LFC_TRACEABILITY - distribution sur REF

- Liste cycles de vie à sécuriser - PREPARE_OG_LFC_TRACEABILITY - fichier out : GUID/Operations/lastOperation.json & Operations/traceabilityInformation.json
  - **Règle** : récupération des journaux des cycles de vie à sécuriser et récupération des informations concernant les dernières opérations de sécurisation.
  - **Type** : bloquant
  - **Statuts** :
    - OK : les fichiers des journaux des cycles de vie ont été exportés (dans ObjectGroup) ainsi que les informations concernant les dernières opérations de sécurisation (PREPARE_OG_LFC_TRACEABILITY.OK=Succès du listage des journaux du cycle de vie des groupes d'objets)
    - KO : les informations sur la dernière opération de sécurisation n'ont pas pu être obtenues / exportées, ou un problème a été rencontré avec un journal de cycle de vie (PREPARE_OG_LFC_TRACEABILITY.KO=Échec du listage des journaux du cycle de vie des groupes d'objets)
    - FATAL : une erreur fatale est survenue (PREPARE_LC_TRACEABILITY.FATAL=Erreur fatale lors du listage des journaux du cycle de vie des groupes d'objets)

6.2.3.1.2 Étape 2 - STP_OG_CREATE_SECURED_FILE - distribution sur LIST - fichiers présents dans ObjectGroup

- Traitement des journaux du cycle de vie des groupes d'objets - OG_CREATE_SECURED_FILE
  - **Règle** : application de l'algorithme pour créer les fichiers sécurisés des journaux du cycle de vie des groupes d'objets, journal par journal, et génèration du fichier sécurisé.
  - **Type** : bloquant
  - **Statuts** :
    - OK : le fichier sécurisé pour le journal du cycle de vie en cours a été généré (STP_OG_CREATE_SECURED_FILE.OK=Succès du processus de sécurisation des groupes d'objets)
    - WARNING : il n'y a pas de nouveaux journaux à sécuriser (STP_OG_CREATE_SECURED_FILE.WARNING=Avertissement lors du du processus de sécurisation des groupes d'objets)
    - KO : le fichier pour le groupe d'objet n'a pas pu être trouvé (STP_OG_CREATE_SECURED_FILE.KO=Échec du processus de sécurisation des groupes d'objets)
    - FATAL : une erreur fatale est survenue lors de la génération des fichiers sécurisés (STP_OG_CREATE_SECURED_FILE.FATAL=Erreur fatale lors du processus de sécurisation des groupes d'objets)
- Vérification de la liste des éléments à traiter (OBJECTS_LIST_EMPTY)
  - **Règle** : vérification de la présence ou non d'objets à traiter. Cette action ne s'inscrit dans le journal des opérations uniquement dans les cas fatal et warning.
  - **Type** : non applicable
  - **Statuts** :
    - WARNING : il n'y a pas de nouveaux journaux à sécuriser (OBJECTS_LIST_EMPTY.WARNING = Avertissement lors de l'établissement de la liste des objets : il n'y a pas d'objet pour cette étape)
    - FATAL : une erreur fatale est survenue lors de la génération des fichiers sécurisés (OBJECTS_LIST_EMPTY.FATAL = Erreur fatale lors de l'établissement de la liste des objets)

6.2.3.1.3 Étape 3 - STP_OG_TRACEABILITY_FINALIZATION - distribution sur REF

- Finalisation de la sécurisation - FINALIZE_OG_LFC_TRACEABILITY - fichier présent dans : GUID/Operations/lastOperation.json & Operations/traceabilityInformation.json
  - **Règle** : récupération des différents fichiers générés aux étapes 1 et 2 puis calcul du tampon d'horodatage

Cette section décrit le processus (workflow) permettant la sécurisation des journaux du cycle de vie mis en place dans la solution logicielle Vitam des unités archivistiques. Le workflow mis en place dans la solution logicielle Vitam est défini dans le fichier « DefaultUnitLifecycleTraceability.json ». Ce fichier est disponible dans : sources/processing/processing-management/src/main/resources/workflows.

Note : Le traitement permettant la sécurisation des journaux du cycle de vie procède par des tranches de lots de 100K. La solution Vitam à la fin de ce premier lot enclenche un autre traitement de 100K et ce jusqu'à avoir traités l'ensemble des unités archivistiques.

6.2.4 Processus de sécurisation des journaux des cycles de vie des unités archivistiques (vision métier)

Le processus de sécurisation des journaux des cycles de vie consiste en la création d'un fichier .zip contenant l'ensemble des journaux du cycle de vie à sécuriser, ainsi que le tampon d'horodatage.

Ce fichier zip est ensuite enregistré sur les offres de stockage, en fonction de la stratégie de stockage.

6.2.5 Sécurisation des journaux du cycle de vie des unités archivistiques LOGBOOK_UNIT_LFC_TRACEABILITY (LogbookLFCAdministration.java)

La fin du processus peut prendre plusieurs statuts :

- **Statuts** :
  - OK : les journaux du cycle de vie ont été sécurisés (LOGBOOK_UNIT_LFC_TRACEABILITY.OK = Succès de la sécurisation des journaux du cycle de vie des unités archivistiques)
  - WARNING : il n'y pas de nouveaux journaux à sécuriser depuis la dernière sécurisation (LOGBOOK_UNIT_LFC_TRACEABILITY.WARNING = Avertissement lors de la sécurisation des journaux du cycle de vie des unités archivistiques)
  - KO : pas de cas KO
  - FATAL : une erreur fatale est survenue lors de la sécurisation du journal des opérations (LOGBOOK_UNIT_LFC_TRACEABILITY.FATAL = Erreur fatale lors de la sécurisation des journaux du cycle de vie des unités archivistiques)

6.2.5.1 Préparation des listes des cycles de vie

6.2.5.1.1 Étape 1 - STP_PREPARE_UNIT_LFC_TRACEABILITY - distribution sur REF

- Liste cycles de vie à sécuriser - PREPARE_UNIT_LFC_TRACEABILITY - fichier out : GUID/Operations/lastOperation.json & Operations/traceabilityInformation.json
  - **Règle** : récupération des journaux des cycles de vie à sécuriser et récupération des informations concernant les dernières opérations de sécurisation.
  - **Type** : bloquant
  - **Statuts** :
    - OK : les fichiers des cycles de vie ont été exportés (dans UnitsWithoutLevel et ObjectGroup) ainsi que les informations concernant les dernières opérations de sécurisation (PREPARE_UNIT_LFC_TRACEABILITY.OK=Succès du listage des journaux du cycle de vie)
    - KO : les informations sur la dernière opération de sécurisation n'ont pas pu être obtenues / exportées, ou un problème a été rencontré avec un cycle de vie (PREPARE_UNIT_LFC_TRACEABILITY.KO=Échec du listage des journaux du cycle de vie)
    - FATAL : une erreur fatale est survenue (PREPARE_UNIT_LFC_TRACEABILITY.FATAL=Erreur fatale lors du listage des journaux du cycle de vie)

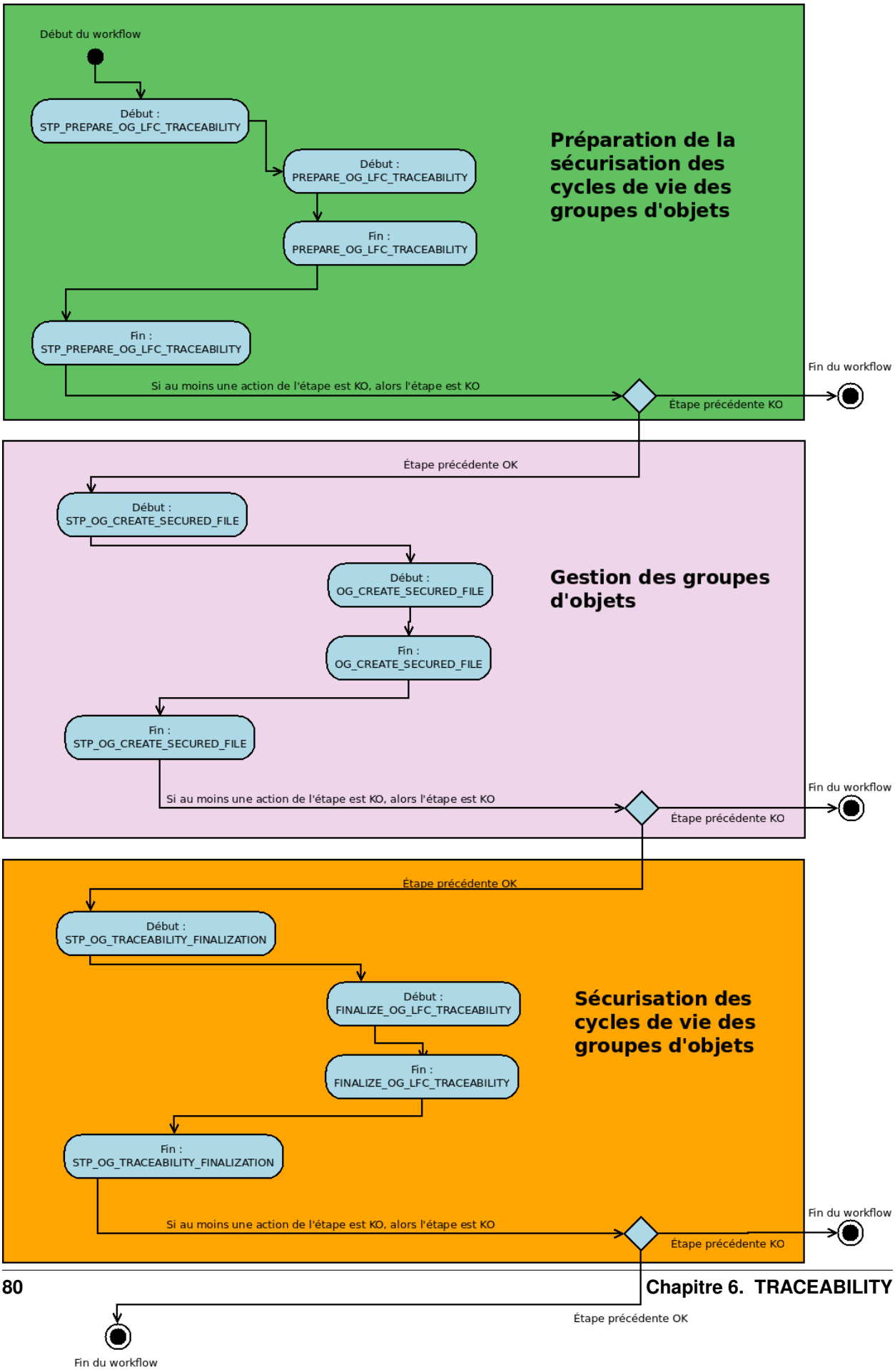
6.2.5.1.2 Étape 3 - STP_UNITS_CREATE_SECURED_FILE - distribution sur LIST - fichiers présents dans GUID

- Traitement des journaux du cycle de vie pour les unités archivistiques - UNITS_CREATE_SECURED_FILE
  - **Type** : bloquant

- **Règle** : application de l'algorithme pour créer les fichiers sécurisés des cycles de vie des unités archivistiques, journal par journal, et génèration du fichier sécurisé.
- **Statuts** :
  - OK : le fichier zip final a été créé et sauvegarder sur les offres de stockage (FINALIZE_OG_LFC_TRACEABILITY.OK=Succès de la sécurisation des journaux du cycle de vie des groupes d'objets)
  - KO : le fichier zip n'a pas pu être généré ou sauvegardé sur les offres (FINALIZE_OG_LFC_TRACEABILITY.KO=Échec de la sécurisation des journaux du cycle de vie des groupes d'objets)
  - FATAL : une erreur fatale est survenue lors de la création du fichier zip final et de la sauvegarde sur les offres de stockage (FINALIZE_OG_LFC_TRACEABILITY.FATAL=Erreur fatale lors de la sécurisation des journaux du cycle de vie des groupes d'objets)

D'une façon synthétique, le workflow est décrit de cette façon :

Diagramme d'activité du workflow de sécurisation des cycles de vie des groupes d'objets


Début du workflow
Début : STP_PREPARE_OG_LFC_TRACEABILITY
Début : PREPARE_OG_LFC_TRACEABILITY
Fin : PREPARE_OG_LFC_TRACEABILITY
Fin : STP_PREPARE_OG_LFC_TRACEABILITY
Préparation de la sécurisation des cycles de vie des groupes d'objets
Si au moins une action de l'étape est KO, alors l'étape est KO
Étape précédente KO
Fin du workflow
Étape précédente OK
Début : STP_OG_CREATE_SECURED_FILE
Début : OG_CREATE_SECURED_FILE
Fin : OG_CREATE_SECURED_FILE
Fin : STP_OG_CREATE_SECURED_FILE
Gestion des groupes d'objets
Si au moins une action de l'étape est KO, alors l'étape est KO
Étape précédente KO
Fin du workflow
Étape précédente OK
Début : STP_OG_TRACEABILITY_FINALIZATION
Début : FINALIZE_OG_LFC_TRACEABILITY
Fin : FINALIZE_OG_LFC_TRACEABILITY
Fin : STP_OG_TRACEABILITY_FINALIZATION
Sécurisation des cycles de vie des groupes d'objets
Si au moins une action de l'étape est KO, alors l'étape est KO
Étape précédente KO
Fin du workflow
Étape précédente OK
Fin du workflow

6.3 Workflow de création de journal des cycles de vie sécurisé des unités archivistiques

6.3.1 Introduction

Cette section décrit le processus (workflow) permettant la sécurisation des journaux du cycle de vie mis en place dans la solution logicielle Vitam des unités archivistiques. Le workflow mis en place dans la solution logicielle Vitam est défini dans le fichier « DefaultUnitLifecycleTraceability.json ». Ce fichier est disponible dans : sources/processing/processing-management/src/main/resources/workflows.

Prudence : Le traitement permettant la sécurisation des journaux du cycle de vie procède par des tranches de lots de 100K. La solution Vitam à la fin de ce premier lot enclenche un autre traitement de 100K et ce jusqu'à avoir traités l'ensemble des unités archivistiques.

6.3.2 Processus de sécurisation des journaux des cycles de vie des unités archivistiques (vision métier)

Le processus de sécurisation des journaux des cycles de vie consiste en la création d'un fichier .zip contenant l'ensemble des journaux du cycle de vie à sécuriser, ainsi que le tampon d'horodatage.

Ce fichier zip est ensuite enregistré sur les offres de stockage, en fonction de la stratégie de stockage.

6.3.3 Sécurisation des journaux du cycle de vie des unités archivistiques LOGBOOK_UNIT_LFC_TRACEABILITY (LogbookLFCAdministration.java)

La fin du processus peut prendre plusieurs statuts :

- **Statuts** :
  - OK : les journaux du cycle de vie ont été sécurisés (LOGBOOK_UNIT_LFC_TRACEABILITY.OK = Succès de la sécurisation des journaux du cycle de vie des unités archivistiques)
  - WARNING : il n'y pas de nouveaux journaux à sécuriser depuis la dernière sécurisation (LOGBOOK_UNIT_LFC_TRACEABILITY.WARNING = Avertissement lors de la sécurisation des journaux du cycle de vie des unités archivistiques)
  - KO : pas de cas KO
  - FATAL : une erreur fatale est survenue lors de la sécurisation du journal des opérations (LOGBOOK_UNIT_LFC_TRACEABILITY.FATAL = Erreur fatale lors de la sécurisation des journaux du cycle de vie des unités archivistiques)

6.3.3.1 Préparation des listes des cycles de vie

6.3.3.1.1 Étape 1 - STP_PREPARE_UNIT_LFC_TRACEABILITY - distribution sur REF

- Liste cycles de vie à sécuriser - PREPARE_UNIT_LFC_TRACEABILITY - fichier out : GUID/Operations/lastOperation.json & Operations/traceabilityInformation.json

- **Règle** : récupération des journaux des cycles de vie à sécuriser et récupération des informations concernant les dernières opérations de sécurisation.
- **Type** : bloquant
- **Statuts** :
  - OK : les fichiers des cycles de vie ont été exportés (dans UnitsWithoutLevel et ObjectGroup) ainsi que les informations concernant les dernières opérations de sécurisation (PREPARE_UNIT_LFC_TRACEABILITY.OK=Succès du listage des journaux du cycle de vie)
  - KO : les informations sur la dernière opération de sécurisation n'ont pas pu être obtenues / exportées, ou un problème a été rencontré avec un cycle de vie (PREPARE_UNIT_LFC_TRACEABILITY.KO=Échec du listage des journaux du cycle de vie)
  - FATAL : une erreur fatale est survenue (PREPARE_UNIT_LFC_TRACEABILITY.FATAL=Erreur fatale lors du listage des journaux du cycle de vie)

6.3.3.1.2 Étape 3 - STP_UNITS_CREATE_SECURED_FILE - distribution sur LIST - fichiers présents dans GUID

- Traitement des journaux du cycle de vie pour les unités archivistiques - UNITS_CREATE_SECURED_FILE
  - **Type** : bloquant
  - **Règle** : application de l'algorithme pour créer les fichiers sécurisés des journaux du cycle de vie des unités archivistiques, journal par journal, et génèration du fichier sécurisé.
  - **Statuts** :
    - OK : le fichier sécurisé pour les journaux du cycle de vie en cours a été généré (UNITS_CREATE_SECURED_FILE.OK=Succès du processus de sécurisation des journaux du cycle de vie des unités archivistiques)
    - WARNING : il n'y a pas de nouveaux journaux à sécuriser (STP_UNITS_CREATE_SECURED_FILE.WARNING = Avertissement lors du processus de sécurisation des unités archivistiques)
    - KO : le fichier pour le groupe d'objet n'a pas pu être trouvé (UNITS_CREATE_SECURED_FILE.KO=Échec du processus de sécurisation des journaux du cycle de vie des unités archivistiques)
    - FATAL : une erreur fatale est survenue lors de la génération des fichiers sécurisés (UNITS_CREATE_SECURED_FILE.FATAL=Erreur fatale lors du processus de sécurisation des journaux du cycle de vie des unités archivistiques)
- Vérification de la liste des éléments à traiter (OBJECTS_LIST_EMPTY)

Il s'agit du même contrôle que l'étape 2

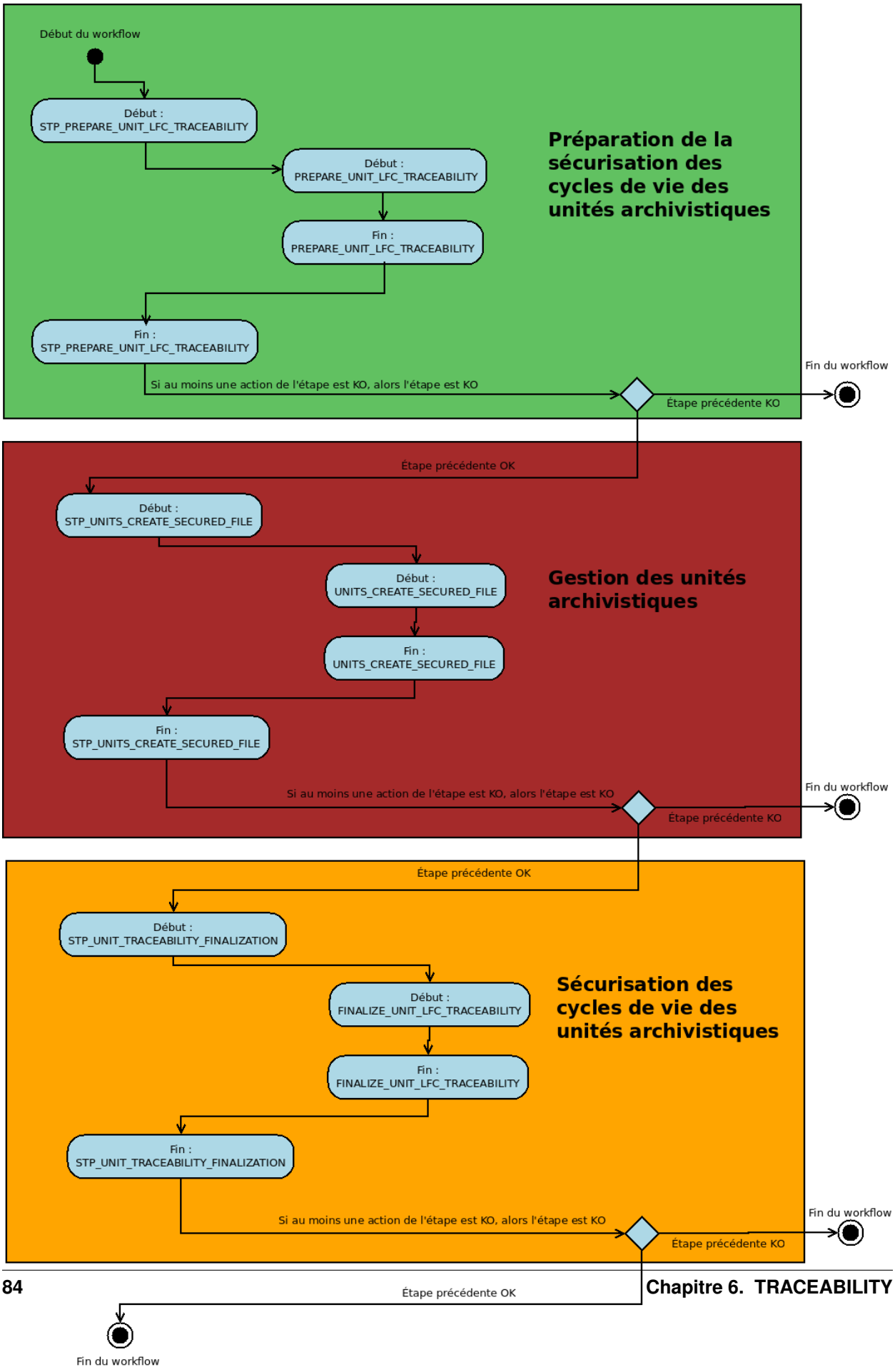
6.3.3.1.3 Étape 4 - STP_UNIT_TRACEABILITY_FINALIZATION - distribution sur REF

- Finalisation de la sécurisation - FINALIZE_LC_TRACEABILITY - fichier présent dans : GUID/Operations/lastOperation.json & Operations/traceabilityInformation.json
  - **Type** : bloquant
  - **Règle** : récupération des différents fichiers générés aux étapes 2 et 3 puis calcul du tampon d'horodatage
  - **Statuts** :
    - OK : le fichier zip final a été créé et sauvegarder sur les offres de stockage (FINALIZE_LC_TRACEABILITY.OK=Succès de la sécurisation des journaux du cycle de vie)
    - KO : le fichier zip n'a pas pu être généré ou sauvegardé sur les offres (FINALIZE_LC_TRACEABILITY.KO=Échec de la sécurisation des journaux du cycle de vie)

- FATAL : une erreur fatale est survenue lors de la création du fichier zip final et de la sauvegarde sur les offres de stockage (FINALIZE_LC_TRACEABILITY.FATAL=Erreur fatale lors de la sécurisation des journaux du cycle de vie)

D'une façon synthétique, le workflow est décrit de cette façon :

Diagramme d'activité du workflow de sécurisation des cycles de vie des unités archivistiques


Début du workflow
Début : STP_PREPARE_UNIT_LFC_TRACEABILITY
Début : PREPARE_UNIT_LFC_TRACEABILITY
Fin : PREPARE_UNIT_LFC_TRACEABILITY
Fin : STP_PREPARE_UNIT_LFC_TRACEABILITY
Préparation de la sécurisation des cycles de vie des unités archivistiques
Si au moins une action de l'étape est KO, alors l'étape est KO
Étape précédente KO
Fin du workflow
Étape précédente OK
Début : STP_UNITS_CREATE_SECURED_FILE
Début : UNITS_CREATE_SECURED_FILE
Fin : UNITS_CREATE_SECURED_FILE
Fin : STP_UNITS_CREATE_SECURED_FILE
Gestion des unités archivistiques
Si au moins une action de l'étape est KO, alors l'étape est KO
Étape précédente KO
Fin du workflow
Étape précédente OK
Début : STP_UNIT_TRACEABILITY_FINALIZATION
Début : FINALIZE_UNIT_LFC_TRACEABILITY
Fin : FINALIZE_UNIT_LFC_TRACEABILITY
Fin : STP_UNIT_TRACEABILITY_FINALIZATION
Sécurisation des cycles de vie des unités archivistiques
Si au moins une action de l'étape est KO, alors l'étape est KO
Étape précédente KO
Fin du workflow
Étape précédente OK
Fin du workflow

6.4 Création de journal sécurisé des journaux des écritures sécurisés

6.4.1 Introduction

Cette section décrit la sécurisation des journaux des écriture mis en place dans la solution logicielle Vitam. Contrairement aux autres sécurisations de journaux de cycles de vie ou du journal des opérations, celle-ci n'est pas utilisée au sein d'un workflow.

6.4.2 Sécurisation des journaux des écritures (vision métier)

Le processus de sécurisation des journaux des écritures consiste en la création d'un fichier .zip contenant :

- Des logs des journaux sécurisés (logFile.log). Ces logs comportent un certain nombre d'informations comme la date des écritures, l'empreinte des fichiers concernés, le tenant, l'adresse des offres...

Ces logs sont un extrait des logs du moteur de stockage, sélectionnés entre deux intervalles de dates.

- Un fichier d'information décrivant le périmètre du fichier des logs des journaux sécurisés associé : date de début et date de fin définissant l'intervalle de sélection des logs à sécuriser, ainsi que l'empreinte du fichier logFile et la date de création du .zip

Au niveau du journal des opérations, cette action est entièrement réalisée dans une seule étape (STP_STORAGE_SECURISATION)

- **Status** :
  - OK : le tampon d'horodatage est calculé (STP_STORAGE_SECURISATION.OK = Succès du processus de sécurisation du journal des écritures)
  - WARNING : il n'y a pas de nouveaux journaux à sécuriser (STP_STORAGE_SECURISATION.WARNING = avertissement lors du processus de sécurisation du journal des écritures)
  - KO : pas de cas KO
  - FATAL : une erreur fatale est survenue lors de l'horodatage (STP_STORAGE_SECURISATION.FATAL = Erreur fatale lors du processus de sécurisation du journal des écritures)

Cette étape déclenche également l'action :

- Vérification de la liste des éléments à traiter (OBJECTS_LIST_EMPTY), décrite dans le paragraphe concernant la création de journal des cycles de vie sécurisé

CHAPITRE 7

UPDATE

7.1 Workflow de mise à jour des unités archivistiques

7.1.1 Introduction

Cette section décrit le processus permettant la mise à jour des unités archivistiques, hors mise à jour des règles de gestion déclenchée par une modification du référentiel des règles de gestion. Ce cas est détaillé dans un autre paragraphe de cette section. Le processus de mise à jour n'est pas configurable.

7.1.2 Processus de mise à jour des unités archivistiques (vision métier)

Le processus de mise à jour des unités archivistiques est lancé lors d'une mise à jour de n'importe quelle métadonnée d'une unité archivistique. Un certain nombre d'étapes et actions sont journalisées dans le journal des opérations. Les étapes et actions associées ci-dessous décrivent ce processus de mise à jour (clé et description de la clé associée dans le journal des opérations).

7.1.3 Mise à jour des unités archivistiques STP_UPDATE_UNIT (AccessInternalModuleImpl.java)

La fin du processus peut prendre plusieurs statuts :

- **Statuts** :
  - OK : la mise à jour de l'unité archivistique a bien été effectuée (STP_UPDATE_UNIT.OK = Succès du processus de mise à jour des métadonnées de l'"unité archivistique)
  - KO : la mise à jour de l'unité archivistique n'a pas été effectuée en raison d'une erreur (STP_UPDATE_UNIT.KO = Échec du processus de mise à jour des métadonnées de l'"unité archivistique)
  - FATAL : une erreur fatale est survenue lors de la mise à jour de l'unité archivistique (STP_UPDATE_UNIT.FATAL = Erreur fatale lors du processus de mise à jour des métadonnées de l'"unité archivistique)

7.1.3.1 Vérification des règles de gestion UNIT_METADATA_UPDATE_CHECK_RULES (AccessInternalModuleImpl.java)

- **Règle** : vérification des règles de gestion
- **Type** : bloquant
- **Statuts** :
  - OK : le rapport est généré (UNIT_METADATA_UPDATE_CHECK_RULES.OK = Succès de la génération du rapport d'analyse du rérentiel des règles de gestion)
  - KO : pas de cas KO
  - FATAL : une erreur fatale est survenue lors de la création du rapport (UNIT_METADATA_UPDATE_CHECK_RULES.FATAL = Erreur fatale lors de la génération du rapport d'analyse du référentiel des règles de gestion)

7.1.3.2 Indexation des métadonnées UNIT_METADATA_UPDATE (ArchiveUnitUpdateUtils.java)

- **Règle** : Indexation des métadonnées des unités archivistiques dans les bases internes de la solution logicielle Vitam, c'est à dire le titre des unités, leurs descriptions, leurs dates extrêmes, etc. C'est également dans cette tâche que le journal du cycle de vie est enregistré dans la base de données.
- **Type** : bloquant
- **Statuts** :
  - OK : les métadonnées des unités archivistiques ont été indexées avec succès (UNIT_METADATA_UPDATE.OK = Succès de la mise à jour des métadonnées des unités archivistiques)
  - KO : les métadonnées des unités archivistiques n'ont pas été indexées (UNIT_METADATA_UPDATE.KO = Échec de la mise à jour des métadonnées des unités archivistiques)
  - FATAL : une erreur fatale est survenue lors de l'indexation des métadonnées des unités archivistiques (UNIT_METADATA_UPDATE.FATAL = Erreur fatale lors de la mise à jour des métadonnées des unités archivistiques)

7.1.3.2.1 Enregistrement du journal du cycle de vie des unités archivistiques

Sécurisation en base des journaux du cycle de vie des unités archivistiques (avant cette étape, les journaux du cycle de vie des unités archivistiques sont dans une collection temporaire afin de garder une cohérence entre les métadonnées indexées et les journaux lors d'une entrée en succès ou en échec).

Cette action n'est pas journalisée.

7.1.3.3 Écriture des métadonnées de l'unité archivistique sur l'offre de stockage UNIT_METADATA_STORAGE (AccessInternalModuleImpl.java)

- **Règle** : Sauvegarde des métadonnées des unités archivistiques sur les offres de stockage en fonction de la stratégie de stockage.(Pas d'évènements stockés dans le journal de cycle de vie)
- **Type** : bloquant
- **Statuts** :
  - OK : la sécurisation des journaux du cycle de vie s'est correctement déroulée (UNIT_METADATA_UPDATE.OK = Succès de l'enregistrement des journaux du cycle de vie des groupes d'objets)
  - FATAL : une erreur fatale est survenue lors de la sécurisation du journal du cycle de vie (UNIT_METADATA_UPDATE.FATAL = Erreur fatale lors de l'enregistrement des journaux du cycle de vie des groupes d'objets)

7.2 Workflow de mise à jour des règles de gestion des unités archivistiques

7.2.1 Introduction

Cette section décrit le processus (workflow) permettant la mise à jour des règles de gestion des unités archivistiques.

Le workflow mis en place dans la solution logicielle Vitam est défini dans le fichier « DefaultRulesUpdateWorkflow.json ». Ce fichier est disponible dans : sources/processing/processing-management/src/main/resources/workflows.

7.2.2 Processus de mise à jour des règles de gestion des unités archivistiques (vision métier)

Le processus de mise à jour des règles de gestion des unités archivistiques est lancé à la suite d'une mise à jour des règles de gestion lorsque la solution logicielle Vitam détecte qu'une règle de gestion a été modifiée et est utilisée par une ou plusieurs unités archivistiques. Toutes les étapes et actions sont journalisées dans le journal des opérations.

Les étapes et actions associées ci-dessous décrivent le processus de mise à jour (clé et description de la clé associée dans le journal des opérations).

7.2.2.1 Préparation des listes d'unités archivistiques à mettre à jour

- **Étape 1** - STP_PREPARE_LISTS

Distribution sur REF -> GUID/PROCESSING/updatedRules.json

- Liste des entrées en cours d'exécution - LIST_RUNNING_INGESTS - fichier de sortie : GUID/PROCESSING/runningIngests.json
  - **Règle** : vérification des entrées en cours d'exécution. Un fichier runningIngests.json est rempli avec les identifiants des entrées en cours. Le fichier est vide si aucune entrée n'est en cours.
  - **Statuts** :
    - OK : le fichier listant les entrées (qu'il soit vide ou non) a bien été créé (LIST_RUNNING_INGESTS.OK=Succès de l'établissement de la liste des processus d'entrée en cours).
    - KO : la liste des entrées en cours n'a pas pu être récupéré, ou alors la liste des entrées n'a pas pu être enregistrée sur le workspace (LIST_RUNNING_INGESTS.KO=Échec de l'établissement de la liste des processus d'entrée en cours)
    - FATAL : une erreur fatale est survenue lors du listage des entrées (LIST_RUNNING_INGESTS.FATAL=Erreur fatale lors de l'établissement de la liste des processus d'entrée en cours)
- Liste des unités archivistiques à mettre à jour - LIST_ARCHIVE_UNITS - fichier de sortie : GUID/PROCESSING/auToBeUpdated.json
  - **Règle** : Récupération de la liste des unités archivistiques à mettre à jour. Pour chaque unité archivisitique concernée, un fichier est créé et déposé sur le workspace pour pouvoir être traité plus tard dans le workflow.
  - **Statuts** :
    - OK : la liste des unités archivistiques à traiter a pu être créée. Les fichiers associés ont bien été créés (LIST_ARCHIVE_UNITS.OK=Succès lors de l'établissement de la liste des unités archivistiques à mettre à jour)

        - FATAL : une erreur fatale est survenue lors du listage des unités archivistiques (LIST_ARCHIVE_UNITS.FATAL=Erreur fatale lors de l'établissement de la liste des unités archivistiques à mettre à jour)

- **Étape 2** - STP_UNIT_UPDATE

Distribution sur LIST GUID/UnitsWithoutLevel. Etape distribuée.

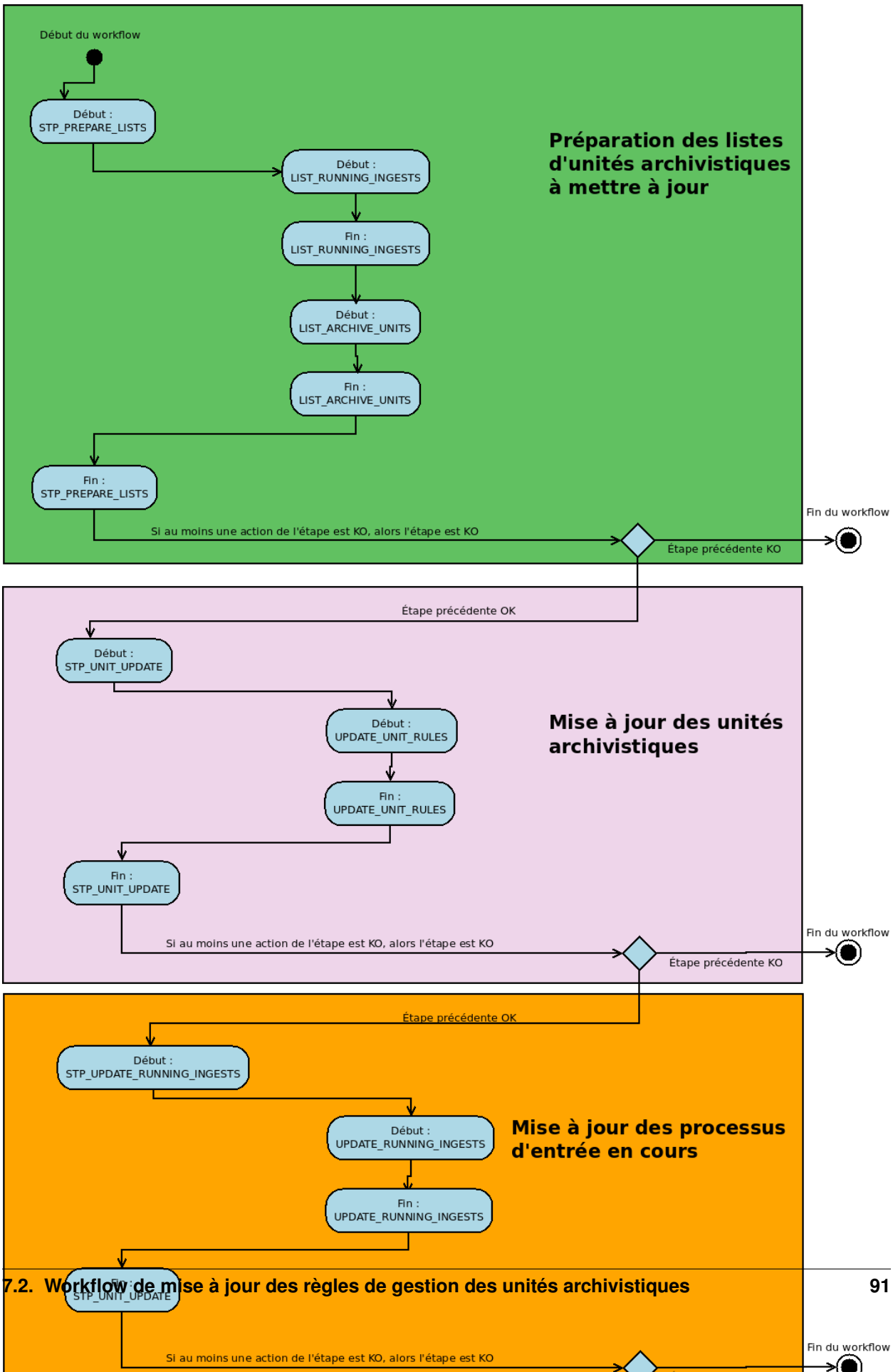
- Mise à jour des règles de gestion d'une unité archivistique - UPDATE_UNIT_RULES
    - **Règle** : pour une unité archivistique, vérification des règles de gestion impactées et recalcul / mise à jour des dates de fin.
    - **Statuts** :
        - OK : l'unité archivistique a bien été mise à jour (UPDATE_UNIT_RULES.OK=Succès de la mise à jour des règles de gestion des unités archivistiques).
        - KO : l'unité archivistique n'a pas été trouvée, ou n'a pas pu être mise à jour (UPDATE_UNIT_RULES.KO=Échec de la mise à jour des règles de gestion des unités archivistiques)
        - FATAL : une erreur fatale est survenue lors de la mise à jour de l'unité archivistique (UPDATE_UNIT_RULES.FATAL=Erreur fatale lors de la mise à jour des règles de gestion des unités archivistiques)

- **Étape 3** - STP_UPDATE_RUNNING_INGESTS

Distribution sur REF GUID/PROCESSING/updatedRules.json.

- Mise à jour des entrées en cours - UPDATE_RUNNING_INGESTS
    - **Règle** : pour une liste d'entrées en cours, vérification de la finalisation de chaque entrées puis vérification des règles de gestion impactées, et recalcul / mise à jour des dates de fin. Fichier d'entrée : GUID/PROCESSING/runningIngests.json
    - **Statuts** :
        - OK : les entrées en cours ont été finalisées, et les unités archivistiques ont bien été mises à jour (STP_UPDATE_RUNNING_INGESTS.OK=Succès du processus de mise à jour des entrées en cours).
        - KO : un problème a été rencontré avec le fichier des règles de gestion mises à jour (STP_UPDATE_RUNNING_INGESTS.KO=Échec du processus de mise à jour des entrées en cours)
        - FATAL : une erreur fatale est survenue lors de la mise à jour des processus d'entrées (STP_UPDATE_RUNNING_INGESTS.FATAL=Erreur fatale lors du processus de mise à jour des entrées en cours)

D'une façon synthétique, le workflow est décrit de cette façon :


Début du workflow
Début : STP_PREPARE_LISTS
Début : LIST_RUNNING_INGESTS
Fin : LIST_RUNNING_INGESTS
Début : LIST_ARCHIVE_UNITS
Fin : LIST_ARCHIVE_UNITS
Fin : STP_PREPARE_LISTS
Préparation des listes d'unités archivistiques à mettre à jour
Si au moins une action de l'étape est KO, alors l'étape est KO
Étape précédente KO
Fin du workflow
Étape précédente OK
Début : STP_UNIT_UPDATE
Début : UPDATE_UNIT_RULES
Fin : UPDATE_UNIT_RULES
Fin : STP_UNIT_UPDATE
Mise à jour des unités archivistiques
Si au moins une action de l'étape est KO, alors l'étape est KO
Étape précédente KO
Fin du workflow
Étape précédente OK
Début : STP_UPDATE_RUNNING_INGESTS
Début : UPDATE_RUNNING_INGESTS
Fin : UPDATE_RUNNING_INGESTS
Mise à jour des processus d'entrée en cours
Si au moins une action de l'étape est KO, alors l'étape est KO
Fin du workflow