Module de collecte

Date	Version
27/06/2022	1.0 (Version 5)

État du document

○ En projet ○ Vérifié ◉ Validé

Maîtrise du document

Responsabilité	Nom	Entité	Date
Rédaction	MVI	Équipe Vitam	29/03/2022
Vérification	MVI	Équipe Vitam	17/02/2022
Validation	AGR	Équipe Vitam	27/06/2022

Suivi des modifications

Version	Date	Auteur	Modifications
0.1	29/03/2022	MVI	Initialisation
0.2	16/05/2022	IJO	Relecture
0.3	19/05/2022	MVI	Corrections
0.4	31/05/2022	Equipe Vitam	Relecture
0.5	10/06/2022	MVI	Corrections
1.0	27/06/2022	AGR	Finalisation du document pour publication de la Version 5

Documents de référence

Document	Date de la version	Remarques
NF Z 44022 – MEDONA – Modélisation des données pour l'archivage	18/01/2014	
Standard d'échange de données pour l'archivage – SEDA – v. 2.1	06/2018	
Vitam – Structuration des *Submission Information Package* (SIP)	04/03/2022	
Vitam – Modèle de données	04/03/2022	
Vitam – Ontologie	04/03/2022	
Vitam – Profils d'archivage	04/03/2022	
Vitam – Profils d'unité archivistiques	04/03/2022	

Licence

Table des matières

1. Résumé

Jusqu'à présent, pour la gestion, la conservation, la préservation et la consultation des archives numériques, les acteurs du secteur public étatique ont utilisé des techniques d'archivage classiques, adaptées aux volumes limités dont la prise en charge leur était proposée. Cette situation évolue désormais rapidement et les acteurs du secteur public étatique doivent se mettre en capacité de traiter les volumes croissants d'archives numériques qui doivent être archivés, grâce à un saut technologique.

1.1. Présentation du programme Vitam

Les trois ministères (Europe et Affaires étrangères, Armées et Culture), combinant légalement mission d'archivage définitif et expertise archivistique associée, ont décidé d'unir leurs efforts, sous le pilotage de la Direction interministérielle du numérique (DINum), pour faire face à ces enjeux. Ils ont décidé de lancer un programme nommé Vitam (Valeurs Immatérielles Transmises aux Archives Pour Mémoire) qui couvre plus précisément les opérations suivantes :

- la conception, la réalisation et la maintenance mutualisées d'une solution logicielle d'archivage électronique de type back-office, permettant la prise en charge, le traitement, la conservation et l'accès aux volumes croissants d'archives (projet de solution logicielle Vitam) ;
- l'intégration par chacun des trois ministères porteurs du Programme de la solution logicielle dans sa plate-forme d'archivage. Ceci implique l'adaptation ou le remplacement des applications métiers existantes des services d'archives pour unifier la gestion et l'accès aux archives, la reprise des données archivées depuis le début des années 1980, la réalisation d'interfaces entre les applications productrices d'archives et la plate-forme d'archivage (projets SAPHIR au MEAE, ADAMANT au MC et ArchiPél au MinArm) ;
- le développement, par un maximum d'acteurs de la sphère publique, de politiques et de plates-formes d'archivage utilisant la solution logicielle.

La solution logicielle Vitam est développée en logiciel libre et recourt aux technologies innovantes du Big Data, seules à même de relever le défi de l'archivage du nombre d'objets numériques qui seront produits ces prochaines années par les administrations de l'État. Afin de s'assurer de la qualité du logiciel livré et de limiter les dérives calendaires de réalisation, le projet est mené selon une conduite de projet Agile. Cette méthode dite « itérative », « incrémentale » et « adaptative » opère par successions de cycles réguliers et fréquents de développements-tests-corrections-intégration. Elle associe les utilisateurs tout au long des développements en leur faisant tester les éléments logiciels produits et surtout en leur demandant un avis sur la qualité des résultats obtenus. Ces contrôles réguliers permettent d'éviter de mauvaises surprises lors de la livraison finale de la solution logicielle en corrigeant au fur et à mesure d'éventuels dysfonctionnements.

Le programme Vitam a bénéficié du soutien du Commissariat général à l'investissement dans le cadre de l'action : « Transition numérique de l'État et modernisation de l'action publique » du

Programme d'investissement d'avenir (PIA). Il a été lancé officiellement le 9 mars 2015, suite à la signature de deux conventions, la première entre les ministères porteurs et les services du Premier ministre, pilote du programme au travers de la DINum, et la seconde entre les services du Premier ministre et la Caisse des dépôts et consignations, relative à la gestion des crédits attribués au titre du Programme d'investissements d'avenir.

La phase projet du Programme Vitam s'est achevée début 2020 avec la publication de la V3 de la solution logicielle et le lancement de la phase produit, définie par une convention de maintenance et amélioration continue entre les ministères porteurs et les services du Premier ministre. Cette nouvelle phase maintient le pilotage stratégique interministériel et confie le pilotage opérationnel au ministère de la Culture. La place des utilisateurs est renforcée par la création du Club utilisateurs, dont un représentant participe aux instances de gouvernance et qui a vocation à permettre les échanges, les retours d'expériences, l'entraide, la définition d'évolution, les contributions, etc.
Le rythme d'une publication majeure par an est maintenu avec la publication début 2021 de la V4 de la solution logicielle Vitam, enrichie par un front-office développé par des utilisateurs et nommé Vitam UI. Une version « release candidate » est également publiée à l'automne.

1.2. Présentation du document

Le document présente les fonctionnalités associées à l'utilisation du module de collecte dans la solution logicielle Vitam (version *béta*).

Il s'articule autour des axes suivants :

- une présentation du module de collecte ;
- une présentation des mécanismes mis en œuvre dans la solution logicielle Vitam pour prendre en compte les opérations de collecte, en application du SEDA ;
- des conseils de mise en œuvre.

Le présent document décrit les fonctionnalités qui sont offertes par la solution logicielle Vitam au terme de la version 5 (mars 2022). Il a vocation à être amendé, complété et enrichi au fur et à mesure de la réalisation de la solution logicielle Vitam et des retours et commentaires formulés par les ministères porteurs et les partenaires du programme.

2. Présentation du module de collecte

2.1. Qu'est-ce que la collecte d'archives ?

La **collecte** d'archives désigne l'ensemble des opérations qui vont conduire au **transfert** d'un ensemble d'archives :

- d'un service producteur à un service d'archives,
- d'un service d'archives intermédiaires à un service d'archives définitives.

Un **versement** correspond à un ensemble d'archives faisant l'objet du transfert.
Le transfert matérialise l'opération visant à transférer la responsabilité de la conservation des archives d'un service à un autre.

La collecte d'archives peut intervenir à plusieurs occasions :

- Un service producteur soumet un versement à un service d'archives ;
- La collecte peut être à l'initiative d'un service d'archives, dans le cadre d'une campagne de collecte d'archives ;
- Des règles de gestion associées à des archives arrivent à échéance et impliquent un transfert de responsabilité en termes de conservation de ces dernières et, de fait, un transfert vers un autre service d'archivage.

La manière de procéder au transfert d'archives est définie dans la norme NF Z 44-022 et dans sa déclinaison pour les acteurs du service public, le Standard d'échanges de données pour l'archivage (SEDA).

La transaction s'effectue en deux temps :

- la soumission d'un ensemble d'archives à verser en vue de :
  - vérifier son contenu et, le cas échéant, procéder à des traitements (ex. suppression de documents, renommage, etc.) ;
  - documenter et contextualiser le versement, par l'ajout d'informations sur le service producteur, son contenu, ses règles de gestion ;
- après validation de l'ensemble d'archives candidates à l'archivage par les parties concernées, transfert des archives en vue d'en assurer leur conservation et transfert de la responsabilité liée à leur conservation.

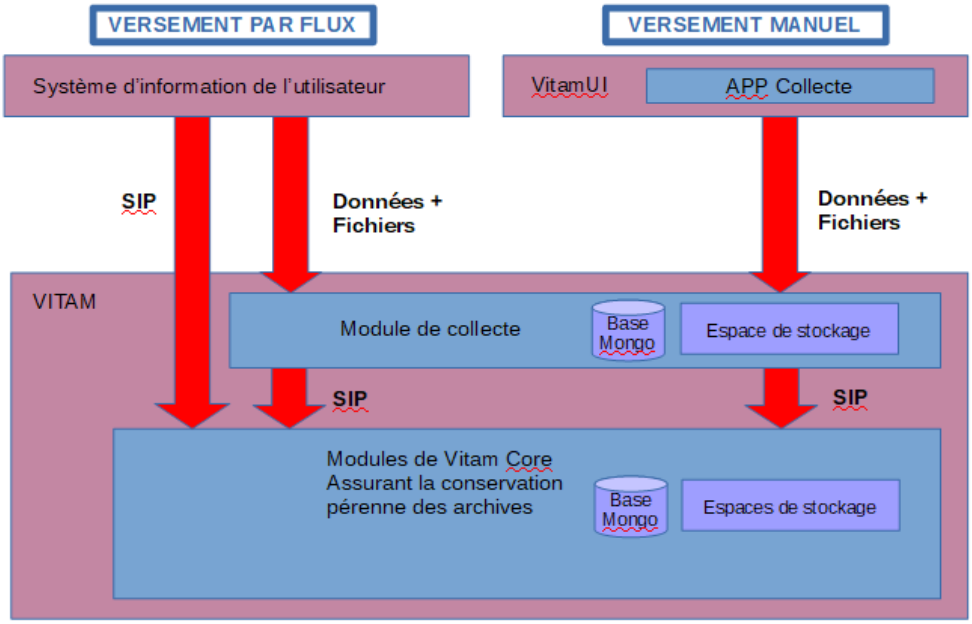
2.2. Qu'est-ce que le module de collecte ?

La solution logicielle Vitam met à disposition un **module de collecte** en vue de recevoir des ensembles hétérogènes d'archives, de procéder à des traitements sur ces archives et de les transférer pour conservation dans le système d'archivage électronique. Ce module :

- constitue un service du back-office de la solution logicielle Vitam ;
- est utilisé par l'APP Collecte du front-office VitamUI *(APP non implémentée en V5)*.

Ce module intervient en amont du transfert (ou versement) d'archives dans la solution logicielle Vitam. Son installation, ainsi que son utilisation, sont optionnelles.

Positionnement du module de collecte en amont du versement



Il permet de :

(Pré-)Verser les archives	collecter depuis l'extérieur un ensemble d'archives, caractérisées par des métadonnées et des fichiers numériques, et constituer : • des versements automatisés émanant d'un système d'information externe ; • des versements manuels et unitaires ◦ constitués par exemple d'arborescences bureautiques ou de messageries, ◦ réalisés depuis des interfaces, notamment celles de l'APP Collecte du front-office VitamUI (*APP non implémentée en V5*)
Traiter les archives	procéder à des traitements archivistiques tels que : • définition de métadonnées contextuelles , • identification de format, • calcul d'empreintes, • réorganisation d'arborescence *(service non implémenté)*, • mise à jour de métadonnées descriptives et de gestion *(service non implémenté)*, • tri, dédoublonnage et suppression de fichiers numériques *(services non implémentés)*,

	• etc.
Transférer les archives	générer un SIP conforme au Standard d'échanges de données pour l'archivage (SEDA) et le transférer dans le système d'archivage électronique pour conservation.

2.3. Pourquoi, quand et comment utiliser le module de collecte ?

Le module de collecte permet de :

- faciliter, voire automatiser, les versements d'archives, depuis un service externe vers la solution logicielle Vitam ;
- le cas échéant, effectuer un contrôle supplémentaire sur les projets de versement et leur contenu, en vue de vérifier la qualité des données, en amont de leur transfert dans un système d'archivage électronique ;
- enrichir les métadonnées (notamment par l'identification de formats et le calcul d'empreintes) ;
- proposer un format pivot et générique pour les services externes souhaitant verser des archives au sein de la solution logicielle Vitam ;
- améliorer l'interopérabilité entre les systèmes d'information.

L'utilisation du module de collecte peut être envisagée dans les cas suivants :

- versements de flux applicatifs, afin de les automatiser ;
- versements de dossiers ou de documents dits « sériels », obéissant strictement à des règles de nommage et de description uniformes (par exemple, des images numérisées par un service d'archives) ;
- versements réguliers et récurrents d'un même type d'archives par différents services producteurs et donc répondant à la volonté de disposer d'une description homogène, y compris pour faciliter les recherches sur celles-ci ;
- versements ponctuels d'archives hétérogènes, quel que soit leur type (bureautiques, audiovisuelles, etc.).

En outre, il peut être utilisé de manière différente en permettant de :

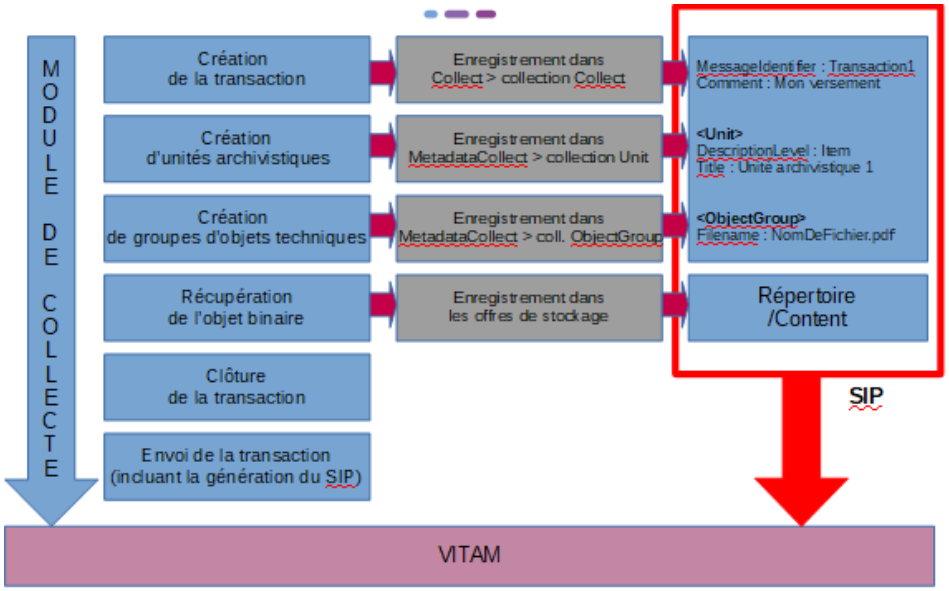
- en faisant appel uniquement à ses services back-office et ses API, automatiser des flux de versements émanant de systèmes d'information ;
- construire des interfaces, appelant ses API permettant de générer un SIP, en vue de faciliter la saisie d'un bordereau de transfert et la préparation d'un versement ;
- utiliser les interfaces de l'APP Collecte du front-office VitamUI (*APP non implémentée en V5*).

3. Mécanismes mis en œuvre dans la solution logicielle Vitam

La solution logicielle Vitam offre à un service d'archives plusieurs fonctionnalités lui permettant de **collecter des archives** au moyen du module de collecte :

- leur **(pré-)versement** (ou collecte) dans le module ;
- leur **transfert** vers les espaces de conservation pérennes de la solution logicielle Vitam.

Fonctionnement du module de collecte



3.1. Versement

3.1.1. Définitions

Dans la solution logicielle Vitam, il est possible de préparer le versement d'un SIP conforme au Standard d'échanges des données pour l'archivage (SEDA) au moyen du module de collecte.

Ce module est propre à chaque tenant de la solution logicielle Vitam.

Il permet de recevoir dans un espace de stockage et des bases de données dédiés :

- sous forme d'enregistrements JSON les informations relatives à :
  - des métadonnées correspondant à l'en-tête du message ArchiveTransfer (Base Collect > Collection Transaction),
  - des métadonnées descriptives et de gestion associées à des unités archivistiques (Base MetadataCollect > Collection Unit),
  - le cas échéant, des métadonnées techniques (Base MetadataCollect > Collection

ObjectGroup),
- les objets associés.

À l'étape d'ajout des objets, la solution logicielle Vitam réalise une mise à jour des métadonnées techniques, qui inclut :
- un calcul d'empreinte de l'objet,
- une identification de formats.

Les fonctionnalités ont été conçues et réalisées de manière à prendre en compte :
- 1 à n unités archivistiques pour une transaction, correspondant à un ensemble de métadonnées d'en-tête,
- 0 à n parents pour les unités archivistiques,
- 1 groupe d'objets techniques par unité archivistique,
- 1 objet binaire pour une version d'usage de l'objet pour un groupe d'objets techniques donné.

Point d'attention : Le versement des données et des objets dans le module de collecte ne sont pas journalisées dans le journal des opérations.

Après réception, le module formate ces différentes informations sous la forme de SIP conformes au standard SEDA, puis les transfère vers le back-office de la solution logicielle Vitam (opération de type « INGEST »). Cette dernière opération est journalisée dans le journal des opérations (« INGEST »).

3.1.2. Utilisation des API

L'envoi et l'enregistrement des données dans le module de collecte s'effectue au moyen de quatre commandes distinctes :
- création d'une transaction correspondant aux métadonnées d'en-tête,
- création unitaire des unités archivistiques,
- création unitaire des groupes d'objets techniques,
- ajout des objets.

Point d'attention : l'ordre d'utilisation des API doit être respecté en vue d'enregistrer avec succès l'ensemble des données et objets constituant un versement.

3.1.2.1. Envoi d'une transaction

Dans la transaction sont envoyées les métadonnées correspondant aux informations dites d'en-tête du bordereau telles que l'identifiant du message ou le contrat d'entrée.

La transaction peut comporter les éléments suivants[1] :

1 Pour plus d'informations, consulter le document *Modèle de données*, chapitre 6.1, « Collection Collect ». Un exemple de contexte de collecte se trouve dans l'annexe 1 du présent document.

Champ	Description
ArchivalAgreement	identifiant du contrat d'entrée utilisé pour réaliser l'entrée, destiné à alimenter le champ ArchivalAgreement du message ArchiveTransfer (champ obligatoire).
MessageIdentifier	identifiant du lot d'objets, utilisé pour identifier les versements, destiné à alimenter le champ MessageIdentifier du message ArchiveTransfer (champ obligatoire).
Comment	précisions sur la demande de transfert destiné à alimenter le champ Comment du message ArchiveTransfer (champ facultatif).
OriginatingAgencyIdentifier	identifiant du service producteur, destiné à alimenter le champ OriginatingAgencyIdentifier du message ArchiveTransfer (champ obligatoire).
SubmissionAgencyIdentifier	identifiant du service versant, destiné à alimenter le champ SubmissionAgencyIdentifier du message ArchiveTransfer (champ obligatoire). S'il est absent ou vide à l'initialisation de la transaction, alors la valeur contenue dans le champ OriginatingAgencyIdentifier est reportée dans ce champ.
ArchivalAgencyIdentifier	identifiant du service d'archivage, destiné à alimenter le champ ArchivalAgencyIdentifier du message ArchiveTransfer (champ obligatoire).
TransferringAgencyIdentifier	identifiant du service de transfert, destiné à alimenter le champ TransferringAgencyIdentifier du message ArchiveTransfer (champ obligatoire).
ArchivalProfile	identifiant du profil d'archivage utilisé pour réaliser l'entrée, destiné à alimenter le champ ArchivalProfile du message ArchiveTransfer (champ facultatif).
AcquisitionInformation	Modalité d'entrée, destiné à alimenter le champ AcquisitionInformation du message ArchiveTransfer (champ facultatif).

Point d'attention : Au terme de la version 5 :

- on ne peut pas inclure de règles de gestion dans la partie transactionnelle ;
- aucun contrôle n'est effectué avec les référentiels présents dans la solution logicielle Vitam. En outre, il n'est pas possible de modifier les informations envoyées dans le module de collecte. De fait, il est fortement recommandé de :
  - veiller à inclure des références (ex. service producteur, contrat d'entrée, etc.) existant dans la solution, de manière à éviter de possibles échecs lors des étapes de contrôles des données référentielles du processus d'entrée (opération « INGEST ») ;
  - ne pas envoyer dans le module de collecte des éléments sans valeurs, en particulier ceux

qui sont obligatoires et doivent nécessairement être renseignés.

Exemple : requête de transaction

```
POST  {{url}}/collect/v1/transactions
Accept: application/json
Content-Type: application/json
X-Tenant-Id: {{tenant}}
X-Access-Contract-Id: ContratTNR

{
  "ArchivalAgencyIdentifier": "Vitam",
  "TransferingAgencyIdentifier": "AN",
  "OriginatingAgencyIdentifier": "MICHEL_MERCIER",
  "SubmissionAgencyIdentifier": "MICHEL_MERCIER",
  "MessageIdentifier": "20220302-000005",
  "ArchivalAgreement":"IC-000001",
  "Comment": "Versement du service producteur : Cabinet de Michel Mercier"
}
```

Cette action provoque la création d'un enregistrement dans la base de données MongoDB, dans la collection « Collect »[2] (base *Collect*)[3].
À cet enregistrement, est associé un statut dont la valeur est égale à « OPEN ».

Exemple : enregistrement de la transaction dans la collection « Collect »


```
{
   "_id": "aeeaaaaaaghdvam4abgoyambfxnhd6aaaaaq",
   "Status": "OPEN",
   "context": {
      "ArchivalAgreement": "IC-000001",
      "MessageIdentifier": "20220302-000005",
      "ArchivalAgencyIdentifier": "Vitam",
      "TransferingAgencyIdentifier": "AN",
      "OriginatingAgencyIdentifier": "MICHEL_MERCIER",
      "SubmissionAgencyIdentifier": "MICHEL_MERCIER",
      "Comment": "Versement du service producteur : Cabinet de Michel Mercier"
   },
   "_tenant": 1
}
```


Lors de cette action, l'opération peut aboutir aux résultats suivants :

2 À noter que cette collection sera renommée « Transaction » dans la version 6 RC de la solution logicielle Vitam.

3 Pour plus d'informations sur l'enregistrement de la transaction dans la base de données, il est recommandé de consulter le document *Modèle de données*.

Statut	Motifs
Succès	Action réalisée sans rencontrer de problèmes particuliers.
Échec	Présence d'un champ inconnu dans la requête.

3.1.2.2. Envoi d'unités archivistiques

À une transaction peuvent être associées 1 à n unité(s) archivistique(s).
Chaque unité archivistique :

- doit définir un titre (Title) et un niveau de description (DescriptionLevel), qui sont des éléments rendus obligatoires en entrée de la solution logicielle Vitam ;
- peut inclure :
  - des règles de gestion ;
  - des métadonnées descriptives supplémentaires ;
  - un lien vers une unité archivistique de niveau supérieur.

Point d'attention :

- En prérequis à la création d'une unité archivistique, il faut avoir au préalable créé une transaction et la signaler dans l'API. En outre, la transaction associée doit avoir un statut « OPEN »[4] ;
- Au terme de la version 5, aucun contrôle n'est effectué avec les référentiels présents dans la solution logicielle Vitam. En outre, à ce stade, il n'est pas possible de modifier les informations envoyées dans le module de collecte. De fait, il est fortement recommandé de veiller à :
  - inclure les métadonnées obligatoires attendues par la solution logicielle Vitam, à savoir un titre (Title) et un niveau de description (DescriptionLevel) ;
  - inclure des règles de gestion ou des identifiants de profil d'unité archivistique existant dans la solution, de manière à éviter de possibles échecs lors des étapes de contrôles des données référentielles du processus d'entrée (opération « INGEST ») ;
  - veiller au bon nommage des éléments (ou vocabulaires internes), tout en respectant leurs caractéristiques (s'il s'agit d'une chaîne de caractères, d'une date, etc.), et à leur insertion dans l'ontologie, s'il s'agit de vocabulaires externes ;
  - ne pas envoyer dans le module de collecte des éléments sans valeurs, en particulier ceux qui sont obligatoires et doivent nécessairement être renseignés.

Exemple : requête de création d'une unité archivistique pour la transaction préalablement créée dont l'identifiant est aeeaaaaaaghiyso4ablmyal74slqwtqaaaaq.

@transaction-id = **aeeaaaaaaghiyso4ablmyal74slqwtqaaaaq**

4 Si la transaction est clôturée (son statut est égal à « CLOSE »), il n'est plus possible de lui adjoindre des unités archivistiques, ainsi que des métadonnées techniques et des objets.

```
@tenant = 1

POST  {{url}}/collect/v1/transactions/{{transaction-id}}/units
Accept: application/json
Content-Type: application/json
X-Tenant-Id: {{tenant}}
X-Access-Contract-Id: ContratTNR
{
   "DescriptionLevel": "RecordGrp",
   "Title": "Discours du ministre",
   "#management": {
    "AccessRule": {
        "Rules": [
           {
              "Rule": "ACC-00001",
              "StartDate": "2000-01-01"
           }
        ]
     }
    }
}
```

Points d'attention :

- L'élément #management doit **toujours** être présent dans la requête, même si aucune règle de gestion n'est définie pour une unité archivistique donnée ;

Exemple : requête de création d'une unité archivistique qui ne définit pas de règle de gestion.

```
POST  {{url}}/collect/v1/transactions/{{transaction-id}}/units
Accept: application/json
Content-Type: application/json
X-Tenant-Id: {{tenant}}
X-Access-Contract-Id: ContratTNR

{
   "DescriptionLevel": "RecordGrp",
   "Title": "Discours de Michel Mercier",
   "#management": {
   }
}
```

- Pour associer une unité archivistique à une autre unité archivistique, de niveau supérieur, il est recommandé de :
  - créer en premier lieu l'unité archivistique de niveau supérieur et de récupérer son

identifiant technique,

- ajouter parmi les métadonnées de l'unité archivistique de niveau inférieur le paramètre #unitups avec pour valeur l'identifiant de(s) unité(s) archivistique(s) de niveau supérieur.

Exemple : requête de création d'une unité archivistique incluant un rattachement à une unité archivistique de niveau supérieur.

```
POST  {{url}}/collect/v1/transactions/{{transaction-id}}/units
Accept: application/json
Content-Type: application/json
X-Tenant-Id: {{tenant}}
X-Access-Contract-Id: ContratTNR

{
   "DescriptionLevel": "Item",
   "Title": "Discours prononcé à l'occasion de l'audition devant la commission des lois relative à la proposition de loi constitutionnelle relative à la fonction de représentation par le Sénat des collectivités locales",
   "TransactedDate":"2010-12-08",
   "#unitups": ["aeeaaaaaaghiyso4ablmyal74smvqcqaaaaq"],
   "#management": {
   }
}
```

Cette action provoque la création d'un enregistrement dans la base de données MongoDB, dans la collection « Unit » (base *MetadataCollect*)[5].
À cet enregistrement, est associé l'identifiant de la transaction (_opi).

Exemple : enregistrement de l'unité archivistique dans la collection « Unit » de la base « MetadataCollect ».


```
{
   "_id": "aeeaaaaaaghdvam4abgoyambfxvoaoiaaaaq",
   "DescriptionLevel": "RecordGrp",
   "Title": "Discours du ministre",
   "_mgt": {
      "AccessRule": {
         "Rules": [
            {
               "Rule": "ACC-00001",
               "StartDate": "2000-01-01"
            }
         ]
      }
   },
   "_opi": "aeeaaaaaaghdvam4abgoyambfxnhd6aaaaaq",
   "_up": [],
```


5 Pour plus d'informations sur l'enregistrement de l'unité archivistique dans la base de données, il est recommandé de consulter le document *Modèle de données*.

```
  "_us": [],
  "_graph": [],
  "_sps": [],
  "_uds": {},
  "_min": 1,
  "_max": 1,
  "_glpd": "2022-06-04T08:52:56.689",
  "_acd": "2022-06-04T08:52:56.689",
  "_aud": "2022-06-04T08:52:56.689",
  "_v": 0,
  "_av": 0,
  "_tenant": 1
}
```


Lors de cette action, l'opération peut aboutir aux résultats suivants :

Statut	Motifs
Succès	Action réalisée sans rencontrer de problèmes particuliers.
Échec	- L'unité archivistique n'a pas été associée à une transaction ; - La transaction associée à l'unité archivistique n'existe pas ; - La transaction associée à l'unité archivistique a été clôturée.

3.1.2.3. Envoi des métadonnées techniques

À une unité archivistique donnée peut être associé un groupe d'objets techniques, dans lequel on peut ajouter unitairement une version d'objet d'un usage particulier.
Il est possible d'associer plusieurs usages à un groupe d'objets techniques :

- original numérique (BinaryMaster),
- original physique (PhysicalMaster),
- copie numérique (Dissemination),
- vignette (Thumbnail),
- texte brut (TextContent).

Point d'attention :

- En prérequis à la création de métadonnées techniques, il faut avoir au préalable créé :
  - une transaction et la signaler dans l'API. En outre, la transaction associée doit avoir un statut « OPEN »[6] ;
  - une unité archivistique et la signaler dans l'API.
- À cette étape, il n'est pas nécessaire d'envoyer les métadonnées correspondant à l'empreinte d'un fichier numérique, à l'identification de son format ou à son poids, dans la mesure où ce type de métadonnées est calculé lors de l'envoi du fichier numérique ;

6 Si la transaction est clôturée (son statut est égal à « CLOSE »), il n'est plus possible de lui adjoindre des unités archivistiques, ainsi que des métadonnées techniques et des objets.

- Au terme de la version 5, il n'est pas possible de modifier les informations envoyées dans le module de collecte. De fait, il est fortement recommandé de veiller au bon nommage des éléments (ou vocabulaires internes), tout en respectant leurs caractéristiques (s'il s'agit d'une chaîne de caractères, d'une date, etc.), et à leur insertion dans l'ontologie, s'il s'agit de vocabulaires externes.
- Par ailleurs, certains contrôles, disponibles au moment du transfert dans la solution logicielle Vitam n'ont pas été implémentés. Il est fortement recommandé de ne créer dans le module de collecte que des versions initiales d'objet numérique pour un usage donné.

Exemple : requête d'association d'un groupe d'objets techniques, et plus exactement une version initiale d'un original numérique, à une unité archivistique dont l'identifiant est aeeaaaaaaghiyso4ablmyal74sndwiqaaaaq.

```
@transaction-id = aeeaaaaaaghiyso4ablmyal74slqwtqaaaaq
@unit-id = aeeaaaaaaghiyso4ablmyal74sndwiqaaaaq
@tenant = 1

POST  {{url}}/collect/v1/units/{{unit-id}}/objects/BinaryMaster/1
Accept: application/json
Content-Type: application/json
X-Tenant-Id: {{tenant}}
X-Access-Contract-Id: ContratTNR

{
  "fileInfo": {
    "filename": "Test01.jpeg"
  }
}
```

Cette action provoque la création d'un enregistrement dans la base de données MongoDB, dans la collection « ObjectGroup » (base *MetadataCollect*[7]).
À cet enregistrement, est associé l'identifiant de la transaction (_opi).

Exemple : enregistrement des métadonnées techniques dans la collection « ObjectGroup » de la base « MetadataCollect ».


```
{
    "_id": "aeeaaaaaaghdvam4abgoyambfxzdbviaaaaq",
    "_tenant": 1,
    "FileInfo": {
        "Filename": "Test01.jpeg"
    },
    "_nbc": 2,
    "_opi": "aeeaaaaaaghdvam4abgoyambfxnhd6aaaaaq",
```


7 Pour plus d'informations sur l'enregistrement de l'unité archivistique dans la base de données, il est recommandé de consulter le document *Modèle de données*.

```
  "_qualifiers": [
    {
      "qualifier": "BinaryMaster",
      "_nbc": 1,
      "versions": [
        {
          "_id": "aeeaaaaaaghdvam4abgoyambfxzdbvyaaaaq",
          "DataObjectVersion": "BinaryMaster_1",
          "FileInfo": {
            "Filename": "Test01.jpeg"
          },
          "Size": 0
        }
      ]
    }
  ],
  "_v": 1,
  "_av": 1
}
```


Lors de cette action, l'opération peut aboutir aux résultats suivants :

Statut	Motifs
Succès	Action réalisée sans rencontrer de problèmes particuliers.
Échec	- L' unité archivistique n'existe pas ou est erronée ; - L'objet technique n'a pas été associé à une unité archivistique ; - La requête renvoie une deuxième fois une demande de création d'un usage d'objet ayant déjà été versé dans le module de collecte ; - La transaction associée n'existe pas ; - La transaction associée a été clôturée.

3.1.2.4. Envoi des objets

À une unité archivistique associée un groupe d'objets techniques et pour un usage et une version d'objet donné, on peut ajouter unitairement un fichier numérique.

Point d'attention :

- En prérequis à l'envoi d'un objet, il faut avoir au préalable créé :
  - une transaction et la signaler dans l'API. En outre, la transaction associée doit avoir un statut « OPEN »[8] ;
  - une unité archivistique et la signaler dans l'API.

Exemple : requête d'association d'un fichier numérique, et plus exactement une version initiale d'un original

8 Si la transaction est clôturée (son statut est égal à « CLOSE »), il n'est plus possible de lui adjoindre des unités archivistiques, ainsi que des métadonnées techniques et des objets.

numérique, à une unité archivistique dont l'identifiant est aeeaaaaaaghiyso4ablmyal74sndwiqaaaaq.

@transaction-id = aeeaaaaaaghiyso4ablmyal74slqwtqaaaaq
@unit-id = **aeeaaaaaaghiyso4ablmyal74sndwiqaaaaq**
@tenant = 1

POST {{url}}/collect/v1/units/{{unit-id}}/objects/**BinaryMaster/1/binary**
Accept: application/json
Content-Type: application/octet-stream
X-Tenant-Id: {{tenant}}
X-Access-Contract-Id: ContratTNR

< MichelMercier/Discours/ Test01.jpeg

Cette action provoque :

- l'enregistrement de l'objet binaire sur les offres de stockage.
- la mise à jour des métadonnées techniques de l'objet avec ajout de l'empreinte d'un fichier numérique ou de l'identification de son format, mise à jour de son poids exprimé en octets, calculés lors de l'envoi du fichier numérique.

Exemple : ajout de l'empreinte du fichier et de son identification de format et mise à jour du poids dans la collection « ObjectGroup » de la base « MetadataCollect ».


```
{
   "_id": "aeeaaaaaaghdvam4abgoyambfxzdbviaaaaq",
   "_tenant": 1,
   "FileInfo": {
      "Filename": "Test01.jpeg"
   },
   "_nbc": 2,
   "_opi": "aeeaaaaaaghdvam4abgoyambfxnhd6aaaaaq",
   "_qualifiers": [
      {
         "qualifier": "BinaryMaster",
         "_nbc": 1,
         "versions": [
            {
               "_id": "aeeaaaaaaghdvam4abgoyambfxzdbvyaaaaq",
               "DataObjectVersion": "BinaryMaster_1",
               "FormatIdentification": {
                  "FormatLitteral": "JPEG File Interchange Format",
                  "MimeType": "image/jpeg",
                  "FormatId": "fmt/43"
               },
               "FileInfo": {
                  "Filename": "Test01.pdf"
               },
               "Size": 7702,
```

```
                "Uri": "Content/Test01jpeg",
                "MessageDigest":
"0e0cec05a1d72ee5610eaa5afbc904c012d190037cbc827d08272102cdecf0226efcad122b86e7699f767c661c9f3702
379b8c2cb01c4f492f69deb200661bb9",
                "Algorithm": "SHA-512"
            }
        ]
    }
  ],
  "_v": 1,
  "_av": 1
}
```


Lors de cette action, l'opération peut aboutir aux résultats suivants :

Statut	Motifs
Succès	Action réalisée sans rencontrer de problèmes particuliers.
Échec	- L' unité archivistique n'existe pas ou est erronée ; - La transaction associée n'existe pas ; - La transaction associée a été clôturée.

Point d'attention :

- La transaction associée doit avoir un statut « OPEN » ;
- En prérequis à la création d'un , il faut avoir au préalable créé :
  - une transaction et la signaler dans l'API. En outre, la transaction associée doit avoir un statut « OPEN »[9] ;
  - une unité archivistique et la signaler dans l'API.
- Certains contrôles, disponibles au moment du transfert dans la solution logicielle Vitam n'ont pas été implémentés. Il est fortement recommandé de ne pas envoyer dans le module de collecte plusieurs objets numériques pour un usage et une version donnée.

3.2. Transfert

3.2.1. Définitions

La solution logicielle Vitam permet de clôturer un versement, puis de le transférer depuis le module de collecte vers le back-office sous la forme d'un SIP conforme au SEDA 2.1.

L'opération de clôture, effectuée dans le module de collecte, n'est pas journalisée dans le journal des opérations, tandis que l'opération de transfert des archives sous la forme d'un SIP, envoyée par le module de collecte, est tracée dans le journal des opérations (opération de type « INGEST »).

Point d'attention : Au terme de la version 5, aucun mécanisme de purge des versements envoyés

9 Si la transaction est clôturée (son statut est égal à « CLOSE »), il n'est plus possible de lui adjoindre des unités archivistiques, ainsi que des métadonnées techniques et des objets.

en succès vers le back-office n'a été implémenté dans le module de collecte.

3.2.2. Utilisation des API

Une fois le versement des données dans le module de collecte terminé, il est possible de :

- clôturer la transaction,
- générer un SIP pour transfert dans la solution logicielle Vitam.

Point d'attention : l'ordre d'utilisation des API doit être respecté en vue de transférer avec succès l'ensemble des données et objets constituant un versement vers la solution logicielle Vitam pour conservation.

3.2.2.1. Clôture d'une transaction

La solution logicielle permet de clôturer une transaction, une fois l'ensemble des données liées à cette transaction envoyées dans le module de collecte.

Point d'attention :

- Au terme de la version 5, il n'est pas possible de modifier le statut de la transaction, notamment pour la rouvrir.

Exemple : requête de clôture

@transaction-id = **aeeaaaaaaghiyso4ablmyal74slqwtqaaaaq**
@unit-id = aeeaaaaaaghiyso4ablmyal74sndwiqaaaaq
@tenant = 1

POST {{url}}/collect/v1/transactions/**{{transaction-id}}**/close
Accept: application/json
Content-Type: application/json
X-Tenant-Id: {{tenant}}
X-Access-Contract-Id: ContratTNR

{

}

Cette action provoque la modification de l'enregistrement dans la base de données MongoDB, dans la collection « Collect » (base *Collect*) : la valeur du champ « Status » est désormais « CLOSE ».

Exemple : enregistrement de la transaction dans la collection « Collect »

{

```
  "_id": "aeeaaaaaaghdvam4abgoyambfxnhd6aaaaaq",
  "Status": "CLOSE",
  "context": {
     "ArchivalAgreement": "IC-000001",
     "MessageIdentifier": "20220302-000005",
     "ArchivalAgencyIdentifier": "Vitam",
     "TransferingAgencyIdentifier": "AN",
     "OriginatingAgencyIdentifier": "MICHEL_MERCIER",
     "SubmissionAgencyIdentifier": "MICHEL_MERCIER",
     "Comment": "Versement du service producteur : Cabinet de Michel Mercier"
  },
  "_tenant": 1
}
```


Dès lors, il n'est plus possible de lui associer des unités archivistiques, ainsi que des groupes d'objets techniques et des fichiers numériques.

Lors de cette action, l'opération peut aboutir aux résultats suivants :

Statut	Motifs
Succès	Action réalisée sans rencontrer de problèmes particuliers.
Échec	- La transaction associée n'existe pas ; - La transaction associée a déjà été clôturée.

3.2.2.2. Envoi du SIP

Pour une transaction donnée, une fois celle-ci clôturée, le module de collecte permet de générer un SIP et de le transférer au moyen d'une opération de type « INGEST ».

Point d'attention :

- En prérequis à l'envoi du SIP, il faut avoir au préalable clôturé la transaction (son statut doit être égal à « CLOSE ») et signaler cette dernière dans l'API.

Exemple : requête d'envoi du SIP vers la solution logicielle Vitam pour conservation

```
@transaction-id = aeeaaaaaaghiyso4ablmyal74slqwtqaaaaq
@unit-id = aeeaaaaaaghiyso4ablmyal74sndwiqaaaaq
@tenant = 1

POST  {{url}}/collect/v1/transactions/{{transaction-id}}/send
Accept: application/json
Content-Type: application/json
X-Tenant-Id: {{tenant}}
X-Access-Contract-Id: ContratTNR
```

```
{

}
```

Cette action provoque :

- la modification de l'enregistrement dans la base de données MongoDB, dans la collection « Collect » (base *Collect*) ;
- l'initialisation d'une opération de type « INGEST ».

Si l'opération de type « INGEST » est :

- en succès, la valeur du champ « Status » est désormais « ACK_OK » (non implémenté) ;
- en cours, la valeur du champ « Status » est « WAITING_ACK » (non implémenté) ;
- en erreur, la valeur du champ « Status » est « ACK_KO » (non implémenté).

Point d'attention : Au terme de la version 5, la valeur du champ « Status » reste inchangée : « SEND ».

Lors de cette action, l'opération peut aboutir aux résultats suivants :

Statut	Motifs
Succès	Action réalisée sans rencontrer de problèmes particuliers.
Échec	- La transaction associée n'existe pas ; - La transaction associée n'a pas été clôturée.

Point d'attention : Au terme de la release 5, les données transférées avec succès pour archivage ne sont pas purgées du module de collecte.

4. Conseils de mise en œuvre

À l'issue de cette phase de réalisation de fonctionnalités concernant le module de collecte, l'équipe du programme Vitam est en mesure de fournir plusieurs recommandations de mise en œuvre.

Comment formaliser les données dans les API du module de collecte ?

Il est possible de verser séparément dans le module de collecte selon un formalisme spécifique au format JSON :

- les métadonnées correspondant à l'en-tête du message ArchiveTransfer,
- les métadonnées décrivant les unités archivistiques,
- les métadonnées techniques.

Chaque appel API[10] permettant de verser ces métadonnées inclut une liste d'éléments, clés ou vocabulaires que l'on souhaite intégrer dans le module de collecte.

Si ces éléments doivent respecter le formalisme attendu par le SEDA et l'ontologie de la solution logicielle Vitam (s'il s'agit d'une chaîne de caractères, d'une date, d'un élément répétable, etc.), il n'est pas nécessaire de les ordonnancer comme le demande le SEDA.

Si certains éléments, issus du SEDA et/ou du système externe, ne doivent pas être envoyés, il n'est pas nécessaire de les référencer dans les différents appels API[11].

Certains contrôles, disponibles au moment du transfert dans la solution logicielle Vitam n'ont pas été implémentés. Il est fortement recommandé de ne pas envoyer dans le module de collecte des éléments sans valeurs, en particulier ceux qui sont obligatoires et doivent nécessairement être renseignés.

Généralités

Dans un enregistrement JSON, un élément (ou vocabulaire) est désigné par son **nom**.

Exemple : l'élément suivant se nomme Description.

```
"Description": "Ceci est une description"
```

Point d'attention :

- Si un élément (ou vocabulaire), présent dans un enregistrement, n'existe pas dans l'ontologie, Il est fortement recommandé de l'avoir préalablement créé dans le référentiel[12].
- Les vocabulaires issus du SEDA sont nommés de la même manière qu'ils le sont dans le standard (ex. l'élément « Tag » défini dans le SEDA est référencé sous ce nom dans l'ontologie et doit être indiqué comme tel).

10 Ces appels API sont décrits dans le chapitre 3 du présent document, dans la sous-section 3.1.2 « Utilisation des API ».

11 La partie ci-dessous explicite la grammaire attendue par le format JSON dans le cadre d'un appel API. Des précisions sur le formalisme attendu est également disponible dans le document *Modèle de données*.

12 Pour plus d'informations sur les vocabulaires, consulter le document *Ontologie*.

- Il existe néanmoins quelques exceptions à cette règle :
  - le bloc Management doit être déclaré sous le nom #management. La modélisation des catégories de règle de gestion attend :
    - un sous-bloc « Rules » pouvant contenir 0 à n règles de gestion, non défini par le SEDA,
    - un sous-bloc « Inheritance » pouvant bloquer des règles de gestion, non défini par le SEDA. Ce sous-bloc peut également contenir l'élément « PreventRulesId » correspondant au champ « RefNonRuleId » du SEDA.
  - si le bloc Event porte le nom « Event », ses sous-blocs doivent être renommés pour se conformer à l'ontologie :
    - « EventIdentifier » en « evId »,
    - « EventTypeCode » en « evTypeProc »,
    - « EventType » en « evType »,
    - « EventDateTime » en « evDateTime »,
    - « EventDetail » en « evTypeDetail »,
    - « Outcome » en « outcome »,
    - « OutcomeDetail » en « outDetail »,
    - « OutcomeDetailMessage » en « outMessg »,
    - « EventDetailData » en « evDetData ».
  - les éléments Title et Description, s'ils contiennent des attributs, seront intitulés « Title_ » et « Description_ » et devront définir des propriétés[13].

Types

L'élément (ou vocabulaire) est associé à un **type** particulier[14]. On distingue plusieurs types JSON possibles :

- « string » : texte ;
- « number » : nombre, entier ou décimal ;
- « integer » : nombre entier ;
- « boolean » : booléen dont la valeur est true ou false ;
- « object » : objet ;
- « array » : liste ou tableau de valeurs textuelles.

Il doit alors être cohérent avec celui du vocabulaire tel qu'il est déclaré dans l'ontologie[15] :

Type d'indexation dans l'ontologie	Type correspondant dans un profil d'unité archivistique	Commentaires

13 Pour plus d'informations sur ces modélisations particulières, il est recommandé de consulter le document Modèle de données, chapitre 5.1, « Collection Unit ».

14 Les types des éléments propres au SEDA sont listés dans l'annexe 2 « Types JSON ». Il est conseillé de se reporter à cette annexe, afin de typer correctement les éléments.

15 La présence de crochets dans le tableau de correspondances indique que le vocabulaire employé dans l'enregistrement peut ou doit se présenter sous la forme d'un tableau ou « array », pouvant contenir un type de valeur en particulier. Le vocabulaire représenté entre crochets est également répétable.

	Vocabulaire interne	Vocabulaire externe	
TEXT	string ou [string]	[string]	
KEYWORD	string ou [string]	[string]	
DATE	string ou [string] + pattern date	[string] + pattern date	
LONG	number ou integer [number] ou [integer]	[number] ou [integer]	
DOUBLE	number ou [number]	[number]	
BOOLEAN	boolean ou [boolean]	[boolean]	
GEO_POINT	string	[string]	L'équipe Vitam n'a pas investigué sur les usages de ces deux types d'indexation.
ENUM	[string] + pattern énumératif	[string] + pattern énumératif	

Par analogie au SEDA et au langage XML, il convient de prêter attention aux éléments suivants :

- sera qualifié en objet (« object ») un élément contenant des sous-éléments, par exemple : Management, Writer, Keyword ;
- sera qualifié en tableau (ou « array ») :
  - un élément répétable, tel que Tag ou OriginatingAgencyArchiveUnitIdentifier,
  - un vocabulaire externe ;
- certains éléments du SEDA (PreventInheritance, NeedAuthorization, NeedReassessingAuthorization) doivent contenir un booléen.

On trouvera essentiellement les types suivants :

- pour les **informations d'en-tête** : les éléments sont uniquement de type « string » ou « array » ;
- pour une **unité archivistique** :
  - pour les *vocabulaires internes*, propres au SEDA, les principaux types rencontrés sont : « string », « array » et « object », auxquels s'ajoute un unique « boolean » et un unique « integer »[16].
  - quant aux *vocabulaires externes*, ajoutés pour répondre à des besoins et transferts spécifiques, il faut les identifier systématiquement comme des « array » (ou tableaux), c'est-à-dire des éléments répétables. Ces tableaux peuvent inclure ensuite un type particulier de chaînes : texte, entier, décimal ou booléen.
- Pour un **groupe d'objets techniques** :

16 Les types des éléments propres au SEDA sont listés dans l'annexe 2 « Types JSON ». du présent document.

- pour les *vocabulaires internes*, propres au SEDA, les principaux types rencontrés sont : « string », « array » et « object », auxquels s'ajoutent quelques « integer » ou « number ».

Les éléments de type « array » et « object » ont une structuration plus complexe qu'un type simple :

- un type simple se définit par un nom, auquel est associé une seule valeur. Ce type est associé à un élément dont la cardinalité est 0-1 ou 1-1.

Exemple : les vocabulaires Description, GpsAltitude et NeedAuthorization sont des éléments simples. Attendant un entier, Size est caractérisé par un item de type « integer », tandis que NeedAuthorization est de type « boolean ».

"Description": "Ceci est une description"
"GpsAltitude": 390
"NeedAuthorization": true

- un élément de type « array » peut contenir une liste ou un tableau de valeurs ou d'objets. Ce type est associé à un élément dont la cardinalité est 0-n ou 1-n.

Exemple : les vocabulaires externes NomDuCapitaine, AgeDuCapitaine et le vocabulaire interne Tag sont dotés d'un type « array » dans l'enregistrement d'une unité archivistique. Attendant un entier, AgeDuCapitaine est caractérisé par un item de type « integer ».

"NomDuCapitaine" : ["Capitaine Fracasse"],
"AgeDuCapitaine" : [1],
"Tag" : ["Marine", "Capitaine Fracasse"]

- un élément de type « object » définit des sous-éléments. Ce type peut être associé à un tableau de sous-éléments (ex. « RegisteredAgent »).

Exemple :

- *l'élément RegisteredAgent est un tableau d'objets qui contient les sous-propriétés ou sous-éléments FirstName, FullName, BirthName.*

"RegisteredAgent": [{
"FirstName": "description",
"FullName": "description",
"BirthName": "description"
}]

- *l'élément BirthPlace contient les sous-propriétés ou sous-éléments Geogname, Address, PostalCode, City, Region, Country.*

"BirthPlace": {
"Geogname": "Toronto [Ontario]",
"Address": "123 my Street",
"PostalCode": "CD255",

```
            "City": "Toronto",
            "Region": "Ontario",
            "Country": "Canada"
        }
```

Enfin, en fonction du type d'indexation, les valeurs doivent être enregistrées

- soit entre guillemets (string)
- soit sans guillemets (integer, number, boolean).

Exemple : les vocabulaires Title, Size et NeedReassessingAuthorization sont des éléments simples : le premier est un string, le second un integer et le dernier un boolean.

"Title": "Ceci est une description"
"Size": 390
"NeedReassessingAuthorization": false

Cas particulier

Les éléments **« Title » et « Description »**, s'ils définissent un attribut ou sont répétables, doivent prendre la forme d'objet.

Exemple : sont autorisés les vocabulaires Title et Description avec attributs, avec une cardinalité 1 – n.

```
"Title_": {
    "fr": "Ceci est un titre en français",
    "en": "This is a title in English"
 }
"Description_": {
    "fr": "Ceci est une description en français",
    "en": "This is a content in English"
 }
```

Le bloc définissant les **règles de gestion** contient également des particularités. Il doit être déclaré sous le nom #management. La modélisation des catégories de règle de gestion attend :

- un sous-bloc « Rules » pouvant contenir 0 à n règles de gestion,
- un sous-bloc « Inheritance » pouvant bloquer des règles de gestion[17].

Exemple : Le bloc AppraisalRule déclare deux règles différentes dans le sous-bloc Rules, « APP-00001 » et « APP-00002 », et définit un blocage sur la catégorie de règle avec le sous-bloc PreventInheritance.

```
"#management": {
    "AppraisalRule": {
      "Rules": [ {
          "Rule": "APP-00001",
          "StartDate": "2015-01-01",
```

17 Pour plus d'informations sur ces modélisations particulières, il est recommandé de consulter le document *Modèle de données*, chapitre 5.1, « Collection Unit ».

```
          "EndDate": "2095-01-01"
          },
          {
          "Rule": "APP-00002"
          } ],
       "Inheritance": {
          "PreventInheritance": true,
          "PreventRulesId": []
          },
      "FinalAction": "Keep"
}
```

Annexe 1 : Exemples de données entrantes

Nota bene : les cas présentés ci-dessous sont des exemples fictifs.

Métadonnées d'en-tête


```
{
 "ArchivalAgencyIdentifier": "Vitam",
 "TransferingAgencyIdentifier": "AN",
 "OriginatingAgencyIdentifier": "MICHEL_MERCIER",
 "SubmissionAgencyIdentifier": "MICHEL_MERCIER",
 "MessageIdentifier": "20220302-000005",
 "ArchivalAgreement":"IC-000001",
 "Comment": "Versement du service producteur : Cabinet de Michel Mercier"
}
```


Unités archivistiques


```
{
   "DescriptionLevel": "RecordGrp",
   "Title": "Discours de Michel Mercier",
   "#management": {
     "AccessRule": {
         "Rules": [
            {
               "Rule": "ACC-00001",
               "StartDate": "2000-01-01"
            }
         ]
      }
     }
}
```


Objets


```
{
 "fileInfo": {
   "filename": "MonFichier.txt"
 }
}
```

Annexe 2 : Types JSON

Pour les éléments propres au SEDA, le tableau suivant précise les types de certains d'entre eux :

- Métadonnées d'en-tête :

	obligatoire	**string**	**number**	**boolean**	**object**	**array**
ArchivalAgreement	x	x				
MessageIdentifier	x	x				
Comment		x[18]				
OriginatingAgencyIdentifier	x	x				
SubmissionAgencyIdentifier		x				
ArchivalAgencyIdentifier	x	x				
TransferringAgencyIdentifier	x	x				
ArchivalProfile		x				
AcquisitionInformation		x				

- Métadonnées descriptives et de gestion d'une unité archivistique :

	obligatoire	**string**	**number**	**boolean**	**object**	**array**
ArchiveUnitProfile		x				
#management[19]	x				x	
AccessRule					x	
AppraisalRule					x	
StorageRule					x	
ReuseRule					x	
ClassificationRule					x	
HoldRule					x	
Rules					x	x
Rule		x				
StartDate		x[20]				

18 En l'état actuel des développements, le champ Comment est non répétable, alors que, dans le SEDA, il l'est.

19 #management est à employer en lieu et place de la balise Management pour se conformer aux attendus de la solution logicielle Vitam.

20 Avec un pattern date.

EndDate		x[21]				
FinalAction[22]		x				
Inheritance					x	
PreventInheritance				x		
PreventRulesId						x[23]
HoldEndDate		x[24]				
HoldOwner		x				
HoldReason		x				
HoldReassessingDate		x[25]				
PreventRearrangement				x		
ClassificationOwner		x				
ClassificationReassessingDate		x[26]				
NeedReassessingAuthorization				x		
ClassificationLevel		x				
ClassificationAudience		x				
Logbook						x
NeedAuthorization				x		
DescriptionLevel	x	x				
Title	x	x				x
Title_					x	
FilePlanPosition		x				x
SystemId		x				x
OriginatingSystemId		x				x
ArchivalAgencyArchiveUnitIdentifier		x				x
OriginatingAgencyArchiveUnitIdentifier		x				x
TransferringAgencyArchiveUnitIdentifier		x				x
Description		x				x

21 Avec un pattern date.

22 Simple énumération.

23 Tableau contenant des éléments de type string.

24 Avec un pattern date.

25 Avec un pattern date.

26 Avec un pattern date.

Description_					x	
CustodialHistory					x	
CustodialHistoryItem		x				x
Type		x				
DocumentType		x				
Language		x[27]				x
DescriptionLanguage		x[28]				
Status		x				
Version		x				
Tag		x				x
Keyword					x	x
KeywordContent		x				
KeywordReference		x				
KeywordType		x				
Coverage					x	
Spatial		x				x
Temporal		x				x
Juridictional		x				x
OriginatingAgency					x	
SubmissionAgency					x	
Identifier		x				
AuthorizedAgent					x	x
Writer					x	x
Addressee					x	x
Recipient					x	x
Transmitter					x	x
Sender					x	x
FirstName		x				
BirthName		x				
FullName		x				
GivenName		x				
Gender		x				

27 Le SEDA attend plus précisément un pattern langue.

28 Le SEDA attend plus précisément un pattern langue.

BirthDate		x[29]				
DeathDate		x[30]				
BirthPlace					x	
DeathPlace					x	
Geogname		x				
Address		x				
PostalCode		x				
City		x				
Region		x				
Country		x				
Nationality		x				
Corpname		x				
Identifier		x				
Function		x				
Activity		x				
Position		x				
Role		x				
Mandate		x				
RelatedObjectReference					x	
IsVersionOf					x	
Replaces					x	
Requires					x	
IsPartOf					x	
References					x	
ArchiveUnitRefId		x				
DataObjectReference					x	
DataObjectReferenceId		x				
DataObjectGroupReferenceId		x				
RepositoryArchiveUnitPID		x				
RepositoryObjectPID		x				
CreatedDate		x				
TransactedDate		x				

29 Avec un pattern date.
30 Avec un pattern date.

AcquiredDate		x				
SentDate		x				
ReceivedDate		x				
RegisteredDate		x				
StartDate		x				
EndDate		x				
Event					x	
evId[31]		x				
evTypeProc[32]		x				
evType[33]		x				
EvDateTime[34]		x				
evTypeDetail[35]		x				
outcome[36]		x				
outDetail[37]		x				
outMessg[38]		x				
evDetData[39]		x				
Signature					x	
Signer					x	
Validator					x	
ValidationTime		x				
MasterData		x				
ReferencedObject					x	
SignedObjectId		x				

31 evId est à employer en lieu et place de la balise EventIdentifier dans un schéma de contrôle pour se conformer aux attendus de la solution logicielle Vitam.

32 evTypeProc est à employer en lieu et place de la balise EventTypeCode dans un schéma de contrôle pour se conformer aux attendus de la solution logicielle Vitam.

33 evType est à employer en lieu et place de la balise EventType dans un schéma de contrôle pour se conformer aux attendus de la solution logicielle Vitam.

34 evDateTime est à employer en lieu et place de la balise EventDateTime dans un schéma de contrôle pour se conformer aux attendus de la solution logicielle Vitam.

35 evTypeDetail est à employer en lieu et place de la balise EventDetail dans un schéma de contrôle pour se conformer aux attendus de la solution logicielle Vitam.

36 outcome est à employer en lieu et place de la balise Outcome dans un schéma de contrôle pour se conformer aux attendus de la solution logicielle Vitam.

37 outDetail est à employer en lieu et place de la balise OutcomeDetail dans un schéma de contrôle pour se conformer aux attendus de la solution logicielle Vitam.

38 outMessg est à employer en lieu et place de la balise OutcomeDetailMessage dans un schéma de contrôle pour se conformer aux attendus de la solution logicielle Vitam.

39 evDetData est à employer en lieu et place de la balise EventDetailData dans un schéma de contrôle pour se conformer aux attendus de la solution logicielle Vitam.

SignedObjectDigest		x				
Gps					x	
GpsVersionId		x				
GpsAltitude			x			
GpsAltitudeRef		x				
GpsLatitude		x				
GpsLongitude		x				
GpsLongitudeRef		x				
GpsDateStamp		x				