Unica Content Integration
Guide du développeur 12.1

Contents

Chapter 1. Présentation

Unica Content Integration simplifie l'intégration aux systèmes de gestion de contenu et permet d'y rechercher du contenu.

Le contenu récupéré peut ainsi être utilisé par le client Unica Content Integration pour divers cas d'utilisation commerciale orientés sur le contenu. Un client Unica Content Integration est tout produit de la suite Unica qui s'intègre à lui pour consommer le contenu de systèmes cibles.

Plug-in

Pour l’intégration à différents CMS, Unica Content Integration utilise des API REST. Etant donné que chaque CMS dispose d’une interface de programmation unique, Unica Content Integration utilise des plug-ins ou des modules personnalisés écrits spécifiquement pour le CMS cible.

Vous pouvez implémenter des plug-ins à l'aide du langage de programmation Java. Unica Content Integration n'applique aucune dépendance de bibliothèque tierce pour le développement de tels plug-ins. Vous pouvez personnaliser les plug-ins afin d'utiliser n'importe quelle bibliothèque tierce pour son implémentation. Les plug-ins peuvent servir à combler les lacunes logiques liées au système cible.

Les plug-ins améliorent Unica Content Integration de manière non intrusive pour récupérer le contenu souhaité à partir du magasin de contenu externe.

Prise en charge de l'intégration et approche de développement de plug-in

Unica Content Integration fournit une prise en charge prête à l'emploi pour une intégration aisée avec les interfaces RESTful. Il facilite également une approche alternative du développement de plug-ins pour s'intégrer à des systèmes autres que RESTful, comme des bases de données, des systèmes de fichiers ou tout autre référentiel de contenu.

Un plug-in typique écrit pour l'intégration de l'API REST ne contient aucune logique permettant d'établir une connexion avec le système cible et de gérer les conditions de réussite et d'échec au niveau du protocole. Ces responsabilités sont gérées par Content Integration Framework. Les plug-ins ne fournissent que des informations spécifiques au système, telles que les suivantes :

- Emplacement absolu de l'API cible
- Méthode HTTP à utiliser
- En-têtes à fournir
- Corps de la requête à envoyer
- Type de réponse à attendre
- Transformateur pour la réponse reçue

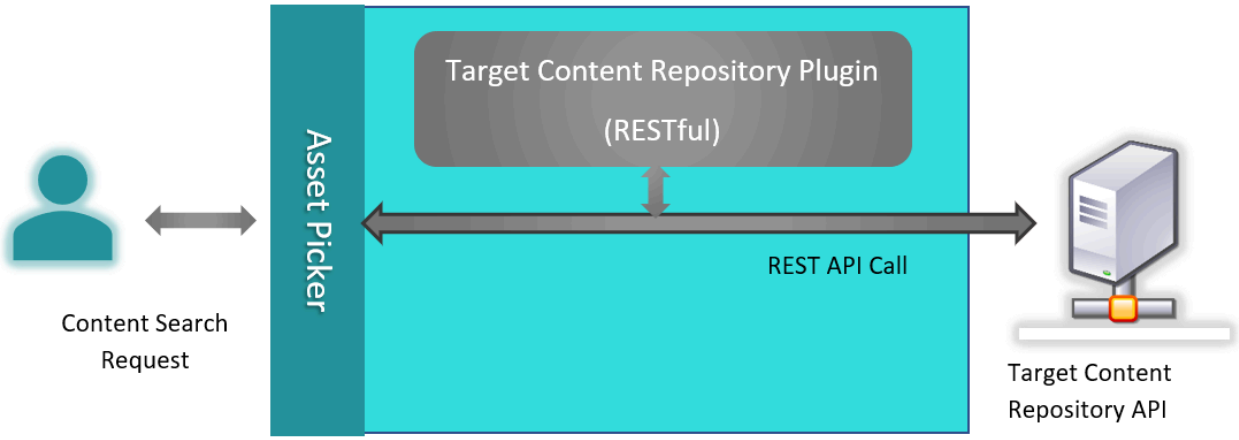
Une autre approche de développement de plug-in pour une intégration non RESTful implique une implémentation complète. Par exemple, un plug-in écrit pour récupérer le contenu de la base de données doit traiter tout ce qui contribue à l'établissement d'une connexion à la base de données, l'exécution de SQL, la fermeture de connexions, l'hydratation d'un ensemble de résultats, la gestion des échecs, etc.

Les plug-ins ne lancent pas la recherche de contenu. Content Integration Framework reçoit d'abord la requête de recherche, qui est déléguée au plug-in respectif. En cas d'intégrations RESTful, Content Integration Framework lance l'interaction HTTP et rassemble les informations nécessaires à partir du plug-in, si nécessaire.

Flux de recherche de contenu RESTful

La figure suivante montre le flux d'exécution de bout en bout pour la recherche de contenu RESTful :

Figure 1. Flux de recherche de contenu RESTful

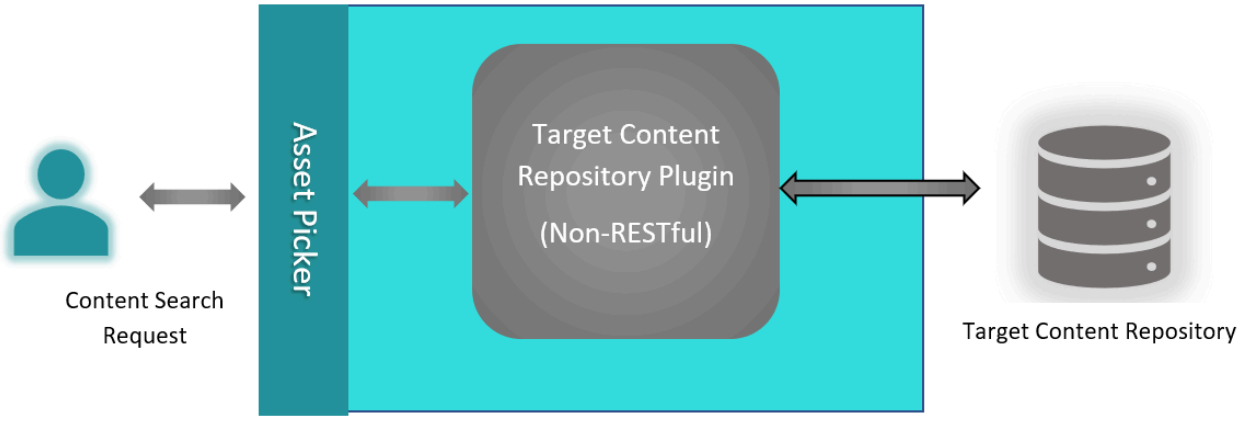


Lorsqu'Content Integration Framework reçoit une demande de recherche de contenu de l'utilisateur pour le système cible, il consulte le plug-in respectif pour recueillir des informations logiques spécifiques à la requête et adresse un appel d'API au système cible. Il consulte à nouveau le plug-in pour transformer la réponse de l'API dans un format attendu et répond à l'utilisateur.

Flux de recherche de contenu non RESTful

La figure suivante montre le flux d'exécution de bout en bout pour la recherche de contenu non RESTful :

Figure 2. Flux de recherche de contenu non RESTful



Les plug-ins non RESTful interagissent avec le référentiel de contenu et communiquent les résultats de la recherche à Content Integration Framework. Contrairement aux référentiels

RESTful, Content Integration Framework ne connaîtra pas le type, l'architecture, le protocole et le mécanisme d'authentification utilisés pour communiquer avec le référentiel cible.

Chapter 2. Présentation du développement de plug-ins

Unica Content Integration simplifie l'intégration aux nouveaux référentiels de contenu sans devoir modifier la structure principale d'Content Integration.

Unica Content Integration s'intègre facilement à des plug-ins indépendants et spécifiques au système. Une fois le plug-in développé et inclus dans le chemin d'accès `<ASSET_PICKER_HOME>/plugins/custom` du serveur d'applications hébergeant Content Integration, le référentiel de contenu correspondant peut être intégré dans la suite de produits Unica en mettant à jour quelques configurations dans Unica Platform. Pour plus d'informations, voir Unica Content Integration - Guide d'administration.

Note: *<ASSET_PICKER_HOME>* désigne le répertoire d'installation de base de Unica Content Integration placé dans le répertoire de base de Platform. Par conséquent, toute utilisation ultérieure de *<ASSET_PICKER_HOME>* dans ce guide doit être considérée comme un chemin d'accès au répertoire Content Integration dans le répertoire de base de Platform.

Unica Content Integration est fourni avec un kit de développement contenant les dépendances, les projets de référence et un projet de démarrage permettant de démarrer rapidement le développement de plug-ins. Le kit de développement est placé dans le répertoire `<ASSET_PICKER_HOME>/dev-kits`. Quatre projets de référence, nommés `aem-integration`, `wcm-integration`, `dx-integration` et `commerce-integration` sont respectivement disponibles pour Adobe Experience Manager (AEM), IBM Web Content Manager (WCM), HCL Digital Experience et HCL Commerce.

Composants de plug-in

Un plug-in standard contient les composants suivants :

- Déclarations de service *(on page 6)*
- Implémentations de service *(on page 19)*

Le terme Service représente une classe Java, qui aide indirectement à consommer un service REST externe ou qui interagit directement avec des services Web ou des systèmes Web externes dans un but précis. Le système externe ne doit pas être un système de gestion de contenu standard et les services externes ne doivent appartenir à aucun CMS standard. Il peut s'agir d'un système ou d'une API.

Tout service implémenté par le plug-in doit être déclaré dans un fichier de déclaration de service géré de façon centralisée. Un fichier de déclaration de service est un fichier de configuration YML contenant la liste des services implémentés par tous les plug-ins disponibles. Le fichier de déclaration de service doit être nommé `custom-plugin-services.yml`. Il doit être disponible dans le répertoire `<ASSET_PICKER_HOME>/conf`. La structure du fichier `custom-plugin-services.yml` doit être similaire au fichier `plugin-services.yml`, qui existe dans le même répertoire. Le fichier `plugin-services.yml` contient des déclarations de service pour des intégrations système prêtes à l'emploi. Un service peut être standard ou personnalisé.

Les services standard comportent une sémantique et un objectif spéciaux dans Asset Picker.Unica Content Integration La mise en œuvre de certains services standard est obligatoire pour qu'Asset Picker fonctionne avec le référentiel de contenu.Content Integration Framework

Déclarations de service

Les déclarations de service de référence se trouvent dans le projet `asset-integration-starter` dans le répertoire `dev-kits\asset-integration-starter\src\main\resources\META-INF`.

Voici des exemples de déclarations de service du projet `asset-integration-starter` :

```
services:
  -
    systemId: Foo
    serviceName: simple-search
    factoryClass: com.example.service.rest.SimpleSearchService
    params:
```

```
    supportedContentTypes: # Standard parameter, applicable only for
simple-search service
      Images: Images
    customParam1: p1Value # String parameter
    customParam2: 1234.56 # Numeric parameter
    customParam3: # Key-value/Dictionary/Map parameter
      p3Key1: p3Value1
      p3Key2: p3Value2
      p3Key3: p3Value3
    customParam4: # Array parameter
      - p4Value1
      - p4Value2
      - p4Value3

 -
   systemId: Foo
   serviceName: resource-loader
   factoryClass: com.example.service.rest.ResourceLoaderService
   params:
     customParam1: p1Value # String parameter
     customParam2: 1234.56 # Numeric parameter
     customParam3: # Key-value/Dictionary/Map parameter
       p3Key1: p3Value1
       p3Key2: p3Value2
       p3Key3: p3Value3
     customParam4: # Array parameter
       - p4Value1
       - p4Value2
       - p4Value3
 -
   systemId: Foo
   serviceName: asset-selection-callback
```

```
    factoryClass: com.example.service.rest.ContentSelectionCallbackService
    params:
      customParam1: p1Value # String parameter
      customParam2: 1234.56 # Numeric parameter
      customParam3: # Key-value/Dictionary/Map parameter
        p3Key1: p3Value1
        p3Key2: p3Value2
        p3Key3: p3Value3
      customParam4: # Array parameter
        - p4Value1
        - p4Value2
        - p4Value3
  -
    systemId: Foo
    serviceName: custom-service
    factoryClass: com.example.service.rest.CustomService
    params:
      customParam1: p1Value # String parameter
      customParam2: 1234.56 # Numeric parameter
      customParam3: # Key-value/Dictionary/Map parameter
        p3Key1: p3Value1
        p3Key2: p3Value2
        p3Key3: p3Value3
      customParam4: # Array parameter
        - p4Value1
        - p4Value2
        - p4Value3
```

Fichier de déclaration de service

Le fichier de déclaration de service contient l'élément `services`, qui est un tableau de déclarations de service individuelles. Une déclaration de service est un dictionnaire

contenant trois éléments obligatoires nommés `systemId`, `serviceName` et `factoryClass`, et un élément facultatif nommé `params`. Les détails des éléments sont les suivants :

- `systemId`

  Cette valeur de chaîne identifie de manière unique un référentiel de contenu cible. Cet identificateur doit contenir de préférence uniquement des caractères alphanumériques anglais. Utilisez des points, des tirets et des traits de soulignement pour plus de lisibilité. Evitez tout autre caractère spécial ou unicode. L'identificateur choisi à une reprise pour le système cible doit rester cohérent dans toutes les déclarations de service pour le même système. Cet identificateur est également utilisé dans la configuration d'Unica Platform pour l'intégration du système respectif.

  Voici quelques exemples d'identificateurs de système valides :

  ```
  WCM
  AEM
  Example
  WCM_1.0
  AEM_1_1
  DX-CORE
  DX
  ```

  Vous pouvez créer différents plug-ins pour différentes versions du même système. Dans ce cas, il est nécessaire d'utiliser différents identificateurs pour identifier distinctement chaque version. Par ailleurs, le même plug-in peut contenir différentes versions d'implémentations de service spécifiques aux différentes versions du système correspondant. Dans ce cas, différents identificateurs système (systemIds) doivent être soigneusement attribués aux déclarations de service respectives. Par exemple, deux versions différentes de WCM, à savoir les versions 1.0 et 2.0, peuvent contenir des API différentes pour le service de recherche de contenu, pour ainsi donner lieu aux entrées de service suivantes pour les versions respectives :

  ```
  -
    systemId: WCM_1.0
  ```

```
    serviceName: simple-search
    factoryClass: com.hcl.wcm.service_1_0.WcmSimpleSearchService


  -
    systemId: WCM_2.0
    serviceName: simple-search
    factoryClass: com.hcl.wcm.service_2_0.WcmSimpleSearchService
```

Les deux entrées peuvent appartenir au même plug-in ou peuvent être placées dans deux plug-ins différents pour des raisons de clarté d'implémentation. Content Integration Framework n'impose aucune restriction.

- `serviceName`

  Cette valeur de chaîne identifie de manière unique le service donné pour le système correspondant. Il peut s'agir d'un nom de service standard ou d'un nom choisi de manière appropriée pour le service personnalisé. Voici la liste des noms de service standard :

  - `simple-search`
  - `resource-loader`

- `factoryClass`

  Il s'agit d'un chemin d'accès complet à la classe Java qui fournit une implémentation de service.

- `params`

  Offre une manière de fournir des paramètres statiques au service pour contrôler ou modifier le comportement de service en fonction des valeurs de paramètre. En bref, `params` peut être utilisé pour conserver la configuration clé-valeur statique pour les implémentations de service. Cela peut inclure certains paramètres de service standard, ainsi que les paramètres personnalisés qu’un service peut vouloir utiliser. Les valeurs de paramètre sont converties en objets ayant les classes d’encapsuleurs primitifs les plus proches, telles que Integer, Long, Double, String, etc. Une valeur de paramètre peut également être une mappe, un tableau ou une liste d’autres valeurs (les plug-ins doivent vérifier le type d’exécution de ces valeurs avant de les utiliser).

Le fichier de déclaration de service contient également certaines propriétés relatives au référentiel de contenu cible. Ces propriétés sont couvertes dans l'élément racine des systèmes. Voici un exemple d'une telle entrée contenant toutes les propriétés prise en charge :

```
systems:
YOUR_SYSTEM_ID:
    params:
        param1: value1
        param2:
            k1: v1
            k2: v2
        param3: 100
    additionalFeatures:
        securityPolicy: false
        content:
            paginatedSearch: true
            paginatedList: true
            anonymousContent: true
```

Cet exemple d'entrée montre les valeurs par défaut prises en compte pour chaque propriété mentionnée ici, dans le cas où aucune entrée de ce type n'est présente pour le référentiel cible donné. Par conséquent, cette entrée est facultative, sauf si une ou plusieurs de ces considérations par défaut ne sont pas vraies pour le référentiel de contenu cible. La section ci-dessous présente la signification de chaque propriété :

`params` - Offre une manière de fournir des paramètres statiques au plug-in respectif pour contrôler ou modifier le comportement du plug-in en fonction des valeurs de paramètre. En bref, params peut être utilisé pour conserver la configuration clé-valeur statique pour les implémentations de plug-in. Cela peut inclure des paramètres système standard prédéfinis, ainsi que tout paramètre personnalisé qu'un plug-in respectif peut vouloir utiliser. Les valeurs de paramètre sont converties en objets ayant les classes d’encapsuleurs primitifs les plus proches, telles que Integer, Long, Double, String, etc. Une valeur de paramètre peut

également être une mappe, un tableau ou une liste d’autres valeurs (les plug-ins doivent vérifier le type d’exécution de ces valeurs avant de les utiliser).

`additionalFeatures | securityPolicy` - Ce paramètre doit être défini sur true lorsque le contenu est protégé au sein du système respectif à l'aide des stratégies de sécurité d'Unica.

`additionalFeatures | content | paginatedSearch` - Cet indicateur de fonctionnalité est utilisé pour communiquer si le référentiel de contenu prend en charge ou non les résultats de recherche de contenu paginés. L'expérience utilisateur est modifiée en conséquence pour afficher les résultats de la recherche de contenu.

`additionalFeatures | content | paginatedList` - Cet indicateur de fonctionnalité est utilisé pour communiquer si le référentiel de contenu prend en charge ou non les listes de contenu paginées. L'expérience utilisateur est modifiée en conséquence pour afficher la liste de contenu.

`additionalFeatures | content | anonymousContent` - Cet indicateur de fonctionnalité est utilisé pour communiquer si le contenu accessible publiquement doit être attendu ou non de la part du référentiel de contenu. S'il est défini sur true, le plug-in doit renvoyer l'URL accessible publiquement pour chaque contenu. Si le contenu ne peut pas être rendu accessible publiquement à l'aide de l'URL HTTP(S), le développeur de plug-ins doit définir cet indicateur sur false. Dans ce cas, les utilisateurs ne pourront pas voir ni télécharger le contenu extrait depuis le référentiel. Si le système cible ne fournit pas d'URL accessible de manière anonyme pour le contenu, vous devez exécuter le service `resource-loader` pour autoriser le téléchargement de contenu protégé.

Services standard

Le tableau ci-dessous présente les services standard d'Unica Content Integration. Par conséquent, aucun des noms de service répertoriés ici ne doit être utilisé pour une implémentation de service personnalisée. Le SDK de Content Integration fournit des interfaces et des types standard pour implémenter ces services standard. Ces interfaces et types sont abordés plus en détail dans les sections suivantes.

Table 1. Services standard et leur description

Nom du service standard	Description
`simple-search`	Le service Recherche simple répond aux demandes de recherche de contenu reçues par Content Integration Framework. Ce service accepte la chaîne de requête de recherche avec les détails de pagination des résultats requis. En fonction du succès de l'opération de recherche, il renvoie le résultat de la recherche pour une requête de recherche donnée et en fonction de la pagination requise. Il s'agit d'un service obligatoire pour le plug-in.
`list-folders`	Ce service est facultatif. Folder est un terme générique utilisé pour représenter un objet de conteneur utilisé dans le système cible pour organiser hiérarchiquement le contenu. Ce service est appelé pour rendre la liste des dossiers et des sous-dossiers afin de faciliter la navigation dans ces contenus organisés hiérarchiquement. **Note:** `list-folders` et `list-contents` sont des services corrélés. L'implémentation des deux services doit exister pour que la navigation de contenu fonctionne correctement.
`list-contents`	Ce service est facultatif. Ce service est appelé pour répertorier le contenu appartenant à un dossier particulier. **Note:** `list-folders` et `list-contents` sont des services corrélés. L'implémentation

Nom du service standard	Description
	des deux services doit exister pour que la navigation de contenu fonctionne correctement.
`get-content-details`	L'implémentation de ce service est utile pour récupérer les détails d'un contenu individuel. Le contenu obtenu à l'aide de `simple-search` et `list-contents services` est référencé plus loin dans d'autres produits Unica. Ultérieurement, il est possible que des utilisateurs souhaitent voir les détails d'un contenu déjà référencé. Par conséquent, nous vous invitons à implémenter ce service pour permettre aux utilisateurs de voir les détails de contenus à la demande.
`get-object-schema`	Ce service est facultatif. L'implémentation de ce service est utile pour permettre aux utilisateurs de Centralized Offer Management de mapper des attributs de contenu à des attributs d'offre, puis d'extraire les valeurs des attributs d'offre mappés à partir des attributs de contenu correspondants en sélectionnant le contenu souhaité dans Content Picker. Par conséquent, s'il est implémenté, ce service facilite l'utilisation d'autres attributs de contenu en plus des URL de contenu pour la création d'offre.
`resource-loader`	Ce service est utile lorsque le téléchargement direct du contenu à partir du système cible n'est pas possible. Ce service n'est pas obligatoire et ne doit être mis en œuvre

Nom du service standard	Description
	que lorsque les complications suivantes se présentent : • **S'il n'existe aucun lien Web direct pour télécharger le contenu** Le contenu renvoyé par les services `simple-search` et `list-contents` doit inclure une URL menant au contenu respectif afin que le client Content Integration puisse le télécharger directement via Internet. Si un tel lien direct vers le contenu n'est pas présent, il est nécessaire d'implémenter le service `resource-loader` en remplaçant l'implémentation par défaut fournie par Content Integration Framework. Par exemple, si le contenu est conservé dans une table de base de données, les services `simple-search` et `list-contents` récupèreront les enregistrements depuis la base de données. Etant donné que les éléments sont chargés à partir de la base de données, il peut ne pas y avoir d'URL pointant directement vers chaque enregistrement. Dans ce cas, le service `resource-loader` peut utiliser l'identificateur de contenu pour localiser et fournir les données appropriées chaque fois que le téléchargement de contenu est

Nom du service standard	Description
	demandé. Toutes les demandes de téléchargement de contenu passeront par Content Integration Framework, qui déléguera la tâche de téléchargement au service `resource-loader` en lui fournissant l'URL du contenu et son identificateur. • **Si les liens Web vers le contenu sont protégés** Il est possible que certains systèmes ne fournissent pas un accès anonyme au contenu malgré la disponibilité de liens Web directs. Dans ces cas-là, l'accès est généralement fourni uniquement après avoir fourni les détails d'authentification requis. Par défaut, Content Integration Framework enregistre une implémentation de service prête à l'emploi du service `resource-loader` pour chaque plug-in. Cette implémentation par défaut utilise la véritable URL du contenu pour télécharger le contenu à partir du système distant en fournissant les détails d'authentification appropriés sujets aux configurations à Unica Platform. (Pour plus d'informations sur les configurations d'intégration du système, voir Unica Content Integration - Guide d'administration).

Nom du service standard	Description
	Sinon, les plug-ins peuvent aussi remplacer l'implémentation resource-loader par défaut pour modifier le comportement de téléchargement de contenu (à l'aide de l'URL ou de l'identificateur de contenu). Si le service `resource-loader` est pris en charge à l'aide de l'approche RESTful, Content Integration Framework continuera à fournir des détails d'authentification basés sur la configuration de Platform. **Note:** Le contenu doit être rendu accessible de manière anonyme s'il doit être consultable/accessible par un public extérieur. Dans ce cas, l'utilisation du service resource-loader n'est pas encouragée dans les systèmes de production. L'utilisation du service `resource-loader` peut être désactivée à tout moment en définissant la propriété Contenu anonyme sur `Oui` dans la configuration de Platform. De même, il peut être activé en définissant la même propriété sur `Non`.
`list-content-categories`	Le contenu peut être catégorisé de manière logique à l'aide de sa classification naturelle. Par exemple, le contenu numérique peut être catégorisé en Images, Documents, Multimédia (audios et vidéos), Archives,

Nom du service standard	Description
	etc. De même, les produits de commerce électronique peuvent être catégorisés en plusieurs catégories générales, telles que Electronique, Soins de santé, Livres, Meubles, etc. Content Integration Framework offre les façons suivantes pour transmettre ces catégorisations de contenu de manière à faciliter la recherche de contenus au sein d'une catégorie spécifique. • **`supportedContentTypes` paramètre de service** Un paramètre de niveau de service standard, `supportedContentTypes`, peut être utilisé pour fournir de manière statique un dictionnaire de types de contenu pris en charge sous la déclaration de service `simple-search`. • **`getSupportedContentTypes()` méthode dans l'implémentation du service de recherche** La méthode `getSupportedContentTypes()` peut être remplacée pour générer dynamiquement une mappe de types de contenu pris en charge, où la clé sert d'identificateur de catégorie et la valeur sert de libellé affiché dans l'interface utilisateur. Cette méthode est exécutée lors du démarrage de l'application. Par conséquent, aucun appel d'API distant ne peut

Nom du service standard	Description
	être effectué à l'aide des capacités de Content Integration Framework, car il est possible que l'application ne soit pas totalement initialisée lorsque cette méthode est appelée. • **`list-content-categories` service** Le service list-content-categories peut éventuellement être implémenté pour répondre à la limitation de la méthode `getSupportedContentTypes()`. Il permet d'effectuer des appels d'API distants pour extraire les catégories de contenu de manière encore plus dynamique. S'il est implémenté, ce service remplace les approches mentionnées précédemment. Content Integration Framework appelle ce service chaque fois que la fenêtre contextuelle de recherche de contenu est rendue.
`get-cognitive-analysis`	Ce service est facultatif. S'il est implémenté, il est utilisé pour extraire les détails cognitifs associés à l'image donnée, sous réserve de la configuration du « Fournisseur de services cognitifs préféré » dans Unica Platform.

Implémentations de service

Pour chaque service déclaré dans le fichier de déclaration de service, une implémentation doit être présente à l'intérieur de la classe `factoryClass` respective.

Content Integration Framework fournit un SDK pour rationaliser l'implémentation du service et facilite le développement rapide de plug-ins. Le SDK Content Integration autorise deux approches différentes pour les implémentations de service : RESTful et fonctionnelle.

Cette section va donner une brève présentation de ces approches. Pour obtenir des informations complémentaires, reportez-vous au projet `asset-integration-starter`.

Cette rubrique présente également certains types et interfaces, leurs paramètres de type générique et énumérations à partir du SDK Content Integration. Pour des détails supplémentaires, voir Développement de plug-ins SDK (on page 27).

Approche RESTful

La classe `com.example.service.rest.CustomService` vous aide à comprendre l'implémentation de service basé sur REST.

Cette classe est une implémentation de l'interface `RestService` et représente donc un service basé sur REST. Etant donné que REST est entièrement basé sur des normes HTTP, l'interface `RestService` dans le SDK Content Integration est étendue à partir de l'interface `HttpService` et est définie comme une interface de marqueur. L'interface `RestService` ne déclare aucune autre méthode supplémentaire. Voici les méthodes déclarées dans l'interface `HttpService`, que l'implémentation de service basée sur REST doit implémenter. Toutes les méthodes ne sont pas obligatoires. Toutes les méthodes acceptent l'objet `ExecutionContext`, qui contient toutes les informations contextuelles nécessaires à chaque méthode pour effectuer sa tâche désignée. Le paramètre de type générique de la classe `ExecutionContext` représente le type d'entrée requis pour le service respectif lors de son appel.

- **HttpRequest buildRequest(ExecutionContext<RQ> executionContext)**
  Il s'agit d'une méthode obligatoire. Elle renvoie un objet de type `com.hcl.unica.cms.model.request.HttpRequest`. La classe `HttpRequest` fournit une API de génération pour construire l'objet avec les détails applicables. Cet objet comprend tous les détails requis pour la réalisation d'une requête HTTP, comme l'URL de nœud final, la méthode HTTP, les en-têtes HTTP et le corps de la requête HTTP. L'API de générateur `HttpRequest` accepte les arguments suivants :
  - **String endpointUrl**

URL absolue vers l'API cible.

- **HttpMethod httpMethod**

  Méthode HTTP à utiliser pour effectuer l'appel d'API.
  Doit correspondre à l'une des valeurs de l'énumération `com.hcl.unica.system.integration.service.HttpMethod`.

- **En-têtes <Mappe<Chaîne, Objet>> optionnels**

  Une mappe facultative d'en-têtes HTTP. Elle peut inclure des en-têtes HTTP standard et personnalisés. Les noms d'en-tête doivent être spécifiés en termes de clés de mappe et les valeurs d'en-tête doivent être fournies en tant que valeurs correspondantes dans la mappe. En l'absence de cette valeur optionnelle, aucun en-tête personnalisé ne sera envoyé avec la requête HTTP sortante.

  Note: Bien que la mappe de l'en-tête accepte les valeurs de type Objet (ou ses sous-types), seuls les objets Chaîne sont pris en charge à compter de l'implémentation actuelle d'Content Integration Framework. Tout autre type de valeur sera ignoré et l'avertissement suivant sera consigné :

  ```
  Header '{HEADER_NAME}' with value '{TO_STRING_REPRESENTATION}'
   will not be set since it is not a String and no Converter is
   available.
  ```

- **Charge<?> facultative**

  Si le service cible attend un corps de requête, cette méthode peut être remplacée pour créer le corps de requête HTTP souhaité. Il peut s'agir de n'importe quel objet valide tant que l'en-tête `Content-Type` approprié est fourni dans la mappe d'en-têtes. En l'absence de cette argument, un corps de requête vide sera envoyé avec la requête HTTP sortante.

  Note: **Prise en charge de Jackson et de JAXB** : la sérialisation d'objets à l'aide de Jackson et de JAXB est entièrement prise en charge par Content Integration Framework. Ainsi, un objet correctement décoré avec des annotations Jackson ou JAXB peut être défini comme la charge de la requête. Dans ce cas, l'en-tête `Content-Type` approprié doit être spécifié dans la mappe d'en-têtes. La

sérialisation de l'objet fourni dans le corps de la requête est gérée par Content Integration Framework. Aucune sérialisation explicite n'est donc requise.

- **Object transformResponse(HttpResponse<RS> response, ExecutionContext<RQ> executionContext)**

Cette méthode facultative transforme la réponse HTTP au format souhaité. Le premier argument, `com.hcl.unica.system.model.response.HttpResponse` , de cette méthode, représente la réponse reçue du système cible. Le paramètre de type générique de la classe `HttpResponse` représente le type de corps de réponse ou la charge de réponse attendue de l'API distante. La charge de réponse peut être de n'importe quel type, comme une chaîne contenant le texte entier tel que reçu du service, un tableau d'octets comprenant le corps de la réponse ou un POJO désérialisé représentant le JSON/XML de la réponse. En plus de la charge de réponse, l'objet `HttpResponse` peut être utilisé pour obtenir des en-têtes de réponse, un code d'état et des cookies.

Note: Prise en charge de Jackson et de JAXB : La désérialisation d'objets à l'aide de Jackson et de JAXB est entièrement prise en charge par Content Integration Framework. Ainsi, un objet correctement décoré avec des annotations Jackson ou JAXB peut être accepté en tant qu'argument adressé à cette méthode. La désérialisation du corps de la réponse en type spécifié est gérée par Content Integration Framework. Par conséquent aucune désérialisation explicite n'est requise pendant la transformation de la réponse à l'intérieur de cette méthode.

En l'absence de cette implémentation, aucune transformation implicite n'est effectuée par Content Integration Framework.

Hormis ces méthodes, il existe une autre méthode dont le `getServiceInterface` a hérité de `com.hcl.unica.system.integration.service.AbstractService interface`, qui doit être implémentée par le service. Cependant, son implémentation est plus pertinente pour l'appel de service que pour l'implémentation de service.

Content Integration Framework se charge de l'interaction HTTP réelle avec le système cible et consulte simplement l'objet de service pour obtenir les détails mentionnés précédemment.

Gestion des erreurs : Content Integration Framework gère les erreurs ou les exceptions reçues lors d'un appel HTTP. Les méthodes répertoriées ci-dessus ne doivent déclencher aucune exception vérifiée. Au besoin, il est possible de déclencher des exceptions non vérifiées.

Approche fonctionnelle

Reportez-vous à la classe `com.example.service.functional.CustomService` pour comprendre l'implémentation du service fonctionnel.

La classe d'objet Java liée est une implémentation d'une interface. Contrairement au service basé sur REST, il n'existe pas de méthode de rappel spécifique HTTP dans ce type d'implémentation de service. En réalité, le service fonctionnel n'est pas nécessairement lié à une invocation HTTP. Ce type de service peut inclure toute opération qui ne dispose pas d'une prise en charge prête à l'emploi depuis Content Integration Framework. Il peut communiquer avec la base de données, invoquer un service Web tiers, gérer le fonctionnement du système de fichiers, etc.

Implémentez la méthode suivante pour un service fonctionnel. Cette méthode accepte également un argument de type `ExecutionContext`, contenant les informations contextuelles requises pour réaliser la tâche souhaitée. Le paramètre de type générique de la classe `ExecutionContext` représente le type d’entrée requis pour le service respectif lors de son appel.

- **RS execute(ExecutionContext<RQ> executionContext)**

  Cette méthode effectue sa tâche désignée à l'aide des informations contextuelles qui lui sont transmises. En retour, elle donne la valeur souhaitée après avoir terminé son opération. La valeur de retour indiquée dans cette signature est un type générique et se base sur le type utilisé lors de l'implémentation de l'interface `FunctionalService`.

Gestion des erreurs

La méthode ci-dessus ne doit déclencher aucune exception vérifiée. Au besoin, il est possible de déclencher des exceptions non vérifiées.

Common methods

Voici les méthodes communes applicables à RESTful et aux services fonctionnels. Ces méthodes sont héritées de l'interface `com.hcl.unica.system.integration.service.AbstractService`.

- **Class<? extends ServiceGateway<RQ, ?>> getServiceInterface()**

  L'implémentation de cette méthode est plus pertinente pour l'appel de service que pour l'implémentation de service. Pour plus d'informations, voir Développement de plug-ins SDK (on page 27).

- **void init(SystemConfig systemConfig, ServiceConfig serviceConfig)**

  Remplacez cette méthode facultative pour effectuer une initialisation unique (après la construction de l'objet de service) avant de traiter une demande. Utilisez l'objet SystemConfig et l'objet ServiceConfig, transmis à cette méthode, afin d'obtenir les détails spécifiques au système et au service pour effectuer les initialisations nécessaires, telles que l'obtention d'une connexion de base de données, l'ouverture d'un descripteur de fichier, etc. Un objet distinct de votre classe de service est créé pour chaque configuration système individuelle dans Unica Platform. Par conséquent, si le même système cible est configuré pour deux partitions différentes dans Unica Centralized Offer Management, deux objets différents de votre classe de service seront créés pour chaque partition. De même, si le même système cible est configuré pour tout autre produit Unica, un objet distinct pour cette configuration existera. L'objet `com.hcl.unica.system.integration.config.SystemConfig` encapsule toutes les configurations système effectuées dans la section de configuration d'Unica Platform, tandis que l'objet `com.hcl.unica.system.integration.config.ServiceConfig` conserve toutes les configurations effectuées pour le service correspondant dans les fichiers `<ASSET_PICKER_HOME>/conf/plugin-services.yml` et `<ASSET_PICKER_HOME>/conf/custom-plugin-services.yml`. Ces objets sont également accessibles à l'aide de `ExecutionContext` dans toutes les méthodes décrites plus haut.

  Note: Content Integration Framework ne fournit pas de méthode spéciale de fin de cycle de vie pour que les services nettoient les éléments initialisés dans la méthode

init. Nous vous recommandons d'utiliser l'approche Java standard en implémentant la méthode Finalize, si nécessaire.

Choix de la meilleure approche

Bien qu'il soit possible d'implémenter un service en utilisant l'une ou l'autre des approches, chacune d'elles présente certains avantages et limites en termes de capacités.

1. **Approche RESTful**
   a. Avantages
      - Moins verbeuse et lecture plus proche de l'interaction HTTP traditionnelle
      - Gestion d'erreur au niveau du transport prête à l'emploi
      - Support prêt à l'emploi pour un nouvel essai en cas de pannes temporaires
      - Support prêt à l'emploi pour la connectivité par proxy
      - Support prêt à l'emploi pour les améliorations futures dans Content Integration Framework

   b. Limitations
      - Ne peut pas être utilisée dans le cas d'intégrations non RESTful ou non HTTP, comme les interactions de base de données ou de système de fichiers
2. **Approche fonctionnelle**
   a. Avantages
      - Peut être utilisée dans le cas d'intégrations non RESTful ou non HTTP, comme les interactions de base de données ou de système de fichiers

   b. Limitations
      - Aucun support prêt à l'emploi disponible pour la gestion des erreurs de niveau de transport, les nouvelles tentatives, la connectivité par proxy et toutes les améliorations futures à partir d'Asset PickerContent Integration Framework
      - Si besoin est, l'implémentation explicite des fonctions prêtes à l'emploi manquantes peut rendre les implémentations de service très détaillées.

Vous pouvez constater que l'approche fonctionnelle convient aux intégrations non basées sur RESTful ou HTTP. Tout service implémenté à l'aide de l'approche RESTful peut également l'être à l'aide de l'approche fonctionnelle en prenant en charge toutes les fonctionnalités prêtes à l'emploi fournies par Content Integration Framework. Bien que

l'approche fonctionnelle donne de la flexibilité en termes de conception de l'implémentation, elle prive de quelques fonctionnalités utiles.

Chapter 3. Développement de plug-ins SDK

Cette rubrique fournit des informations sur les différentes classes, interfaces et énumérations du SDK Content Integration, à l'aide d'unités logiques correspondantes dans des projets de référence `asset-integration-starter`, `aem-integration` et `wcm-integration` qui sont incluses dans le cadre du kit de développement avec la fonctionnalité Content Integration.

Le SDK Content Integration pour le développement de plug-ins se trouve dans le répertoire `<ASSET_PICKER_HOME>/dev-kits/sdk/` de votre serveur d'applications. Les fichiers jar suivants se trouvent dans le répertoire `sdk` :

- `integration-api.jar`
- `entity-mapper-api.jar`
- `standard-integrations.jar`

Ces fichiers jar contiennent toutes les classes, interfaces et énumérations SDK présentées dans cette section. Consultez les classes appropriées à partir de ces fichiers jar chaque fois que vous êtes dans la rubrique correspondante de ce guide.

Paramètres de type générique

Les paramètres de type générique servent à mettre en œuvre des interfaces de service. Pour plus d'informations sur l'interface de service, voir Implémentations de service (on page 19).

Un service qui réside dans un plug-in n'est qu'une unité de programmation, qui prend une certaine entrée et renvoie la sortie attendue. De même, l'API REST, encapsulée par notre service, demande l'entrée requise (corps de demande, en-têtes, cookies et paramètres de requête) et produit la réponse souhaitée (corps de réponse, en-têtes et cookies). Il nécessite certaines notations génériques pour les entrées et sorties échangées pendant le flux logique de bout en bout.

Content Integration Framework utilise le paramètre de type RQ pour indiquer le type d'entrée fourni au service lors de son appel. Ici, le paramètre de type RS est utilisé pour indiquer le type d'objet renvoyé par le service fonctionnel ou le type de corps de réponse renvoyé par l'API REST distante appelée à l'aide de l'approche RESTful. L'objectif de RS peut changer en fonction de l'endroit où il est utilisé, mais il indique toujours la valeur de retour d'un élément.

RestService<RQ, RS>

Reportez la classe `com.example.service.rest.CustomService` du projet `asset-integration-starter` afin de comprendre les paramètres de types utilisés dans l'interface `RestService`. `RestService` n'est qu'une interface de marqueur étendue à partir de `HttpService`. La définition de ces paramètres de types est également similaire pour le `HttpService`.

- **RQ**

  Un service a besoin d'une entrée pour effectuer son opération. RQ correspond au type d'entrée ou de demande que le service demande lorsqu'il est appelé. TLe `com.example.service.rest.CustomService` prend une entrée de type `ServiceInput`. Le même paramètre de type est utilisé dans l'objet `ExecutionContext` communiqué à toutes les méthodes dans l'interface `RestService` ou `HttpService`. L'entrée ou la requête, l'objet communiqué service, au moment où il est invoqué, est obtenu en appelant la méthode `getRequest` dans l'objet `ExecutionContext`.

  ```
  @Override
   public HttpRequest buildRequest(ExecutionContext<ServiceInput>
   executionContext) {
    ServiceInput input = executionContext.getRequest();
    // Remaining implementation omitted for brevity
   }
  ```

- **RS**

  Ce type de paramètre correspond au type de réponse (post-désérialisation) reçue de l'API REST distante. L'implémentation de service choisit ce paramètre en fonction du type d'objet avec lequel elle souhaite travailler dans la méthode `transformResponse`. Si vous examinez la signature de la méthode `transformResponse` de la classe

`com.example.service.rest.CustomService`, vous verrez que `ApiResponse` est communiqué en tant que type d'argument à la classe `HttpResponse`, qui correspond au paramètre de type RS de l'interface `RestService`.

Note: Une désérialisation se produit selon l'en-tête `Content-Type` présent dans la réponse HTTP envoyée par l'API REST. Le type utilisé comme deuxième argument générique de `RestService`, ou `HttpService`, doit être annoté de manière appropriée si une désérialisation Jackson ou JAXB est attendue.

FunctionalService<RQ, RS>

L'interface `FunctionalService` est analogue à l'interface `java.util.function.Function` de la bibliothèque Java standard. Les paramètres de type de `FunctionalService` présentent une sémantique similaire à ceux de l'interface `java.util.function.Function`.

- **RQ**

  Représente le type d'entrée donné au service lors de l'appel.
- **RS**

  Représente le type de valeur renvoyé par le service lors de son arrêt.

ServiceGateway<RQ, RS>

Cette interface sert à implémenter la méthode `getServiceInterface` depuis l'interface `AbstractService<RQ, RS>`. `AbstractService` est une interface importante de `RestService`, ou de `HttpService`, ou de `FunctionalService`. La sémantique pour RQ et RS pour AbstractService est la même que pour `RestService` ou `HttpService`. Elle déclare la méthode `getServiceInterface`, qui doit être mise en œuvre par un service. La méthode `getServiceInterface` doit renvoyer l'objet classe de l'élément dérivé (interface enfant) de `ServiceGateway`. La définition de `com.hcl.unica.system.integration.service.gateway.ServiceGateway` est la suivante :

```
public interface ServiceGateway<RQ, RS> {
  public RS execute(RQ request) throws ServiceExecutionException;
 }
```

La sémantique du paramètre de type RQ est la même que celle mentionnée précédemment. L'autre paramètre de type, RS, représente la sortie du service qui réside dans le plug-in. Il ne représente pas la réponse reçue de l'API REST distante ou de tout autre système cible. Pour la classe `com.example.service.rest.CustomService`, le `CustomServiceGateway` est défini comme l'interface enfant de `ServiceGateway` à l'aide des arguments de type `ServiceInput` et `ServiceOutput`, étant donné que le service reçoit une entrée de type `ServiceInput` et renvoie la valeur de type `ServiceOutput` lors de l'arrêt.

Note:

- La méthode `getServiceInterface` dans la classe `com.example.service.rest.CustomService` renvoie l'objet de classe de `CustomServiceGateway`. L'interface `ServiceGateway` (ou son interface enfant) fournit des informations à propos de l'entrée et de la sortie de la mise en œuvre du service. L'interface `ServiceGateway` sert aussi à contenir la référence de l'instance de service et à appeler son exécution.
- En obtenant une référence à l'instance `ServiceGateway` de tout service ainsi implémenté, la méthode execute(RQ request) peut être appelée pour exécuter le service. Notez que la méthode execute peut engendrer l'exception `ServiceExcecutionException` si quelque chose se passe mal au cours de l'exécution du service. Les détails relatifs à l'appel de service et à la gestion des exceptions seront fournis dans les rubriques qui suivent.

Invocation de service

Le projet `asset-integration-starter` contient une classe `com.example.service.client.CustomServiceClient` pour illustrer l'appel de service.

La classe `CustomServiceClient` obtient une référence à l'objet `SystemGateway` pour le système représenté par un identificateur *Foo* en appelant la méthode `SystemGatewayFactory.getSystemGateway` avec `Foo` comme argument. La méthode `SystemGatewayFactory.getSystemGateway` fournit donc un descripteur à n'importe quel système cible en spécifiant son *systemId*. Une fois que le descripteur est obtenu en tant

qu'objet `SystemGateway`, il peut être utilisé pour appeler un service sur le système cible respectif. Voici le fragment de code correspondant de la classe `CustomServiceClient` :

```
private SystemGateway systemGateway =
 SystemGatewayFactory.getSystemGateway("Foo");
```

SystemGateway

`com.hcl.unica.system.integration.service.gateway.SystemGateway` fournit une méthode surchargée `executeService`, pour l'exécution d'un service sur le système cible. Une version de cette méthode permet d'exécuter tout service déclaré dans les fichiers de déclaration de service (`<ASSET_PICKER_HOME>/conf/custom-plugin-services.yml` et `<ASSET_PICKER_HOME>/conf/plugin-services.yml` pour le système respectif. L'autre version permet d'exécuter un appel HTTP ad hoc sur le système cible sans déclarer de service explicite dans le fichier de déclaration de service. Les deux versions de la méthode `executeService` avec leurs signatures sont les suivantes :

- **<RQ, RS> RS executeService(String serviceName, RQ serviceInput, Class<? extends ServiceGateway<RQ, RS>> gatewayClass) throws ServiceExecutionException**
  Il s'agit d'une méthode générique qui fonctionne avec les paramètres de type RQ et RS. La signification de RQ et RS est la même que celle mentionnée précédemment. Cette méthode permet d'exécuter un service déjà déclaré. La méthode `invocationDemo` dans la classe `CustomServiceClient` illustre l'utilisation de cette méthode. Elle accepte les arguments suivants :
  - **String serviceName**

    Il doit s'agir du nom du service à exécuter. Le nom du service doit correspondre exactement à la déclaration correspondante dans le fichier de déclaration de service.
  - **RQ serviceInput**

    Il s'agit d'une entrée pour le service en cours d'exécution. Le paramètre de type RQ représente le type d'entrée requis pour le service appelé.
  - **Class<? extends ServiceGateway<RQ, RS>> gatewayClass**

Il doit être identique à la valeur de retour de la méthode `getServiceInterface` dans l'implémentation de service correspondante. Il permet à Content Integration Framework d'identifier l'entrée correcte pour le service en cours d'exécution et renvoie la sortie du type souhaité. Les paramètres de type RQ et RS utilisés pour l'argument `gatewayClass` représentent le type d'entrée fourni lors de l'appel de service et le type de réponse renvoyé par le service à la fin, respectivement.

En cas de réussite, cette méthode renvoie l'objet de type représenté par le paramètre de type RS. Par conséquent, le troisième argument de la méthode `executeService`, `gatewayClass` , régit le type d'entrée qui entre dans le service et le type de valeur renvoyé par le service.

- **<T> HttpResponse<T> executeService(HttpRequest request, Class<T> expectedResponse) throws ServiceExecutionException**

  Il s'agit également d'une méthode générique, dans laquelle le paramètre de type T représente le type de réponse attendu par l'appel HTTP distant. Il permet d'effectuer un appel HTTP ad hoc sur le système cible sans déclarer de service explicite pour ce dernier dans le fichier de déclaration de service. La méthode `adHocInvocationDemo` de la classe `CustomServiceClient` illustre l'utilisation de cette méthode. Elle accepte les arguments répertoriés suivants :

  - **Requête HTTPRequest**

    Il doit s'agir d'un objet de la classe `com.hcl.unica.system.model.request.HttpRequest`. `HttpRequest` fournit une interface de générateur pour la construction de l'objet avec les détails requis. Cet objet encapsule essentiellement les détails requis pour effectuer un appel HTTP, tels que l'URL absolue, la méthode de requête HTTP, les en-têtes de requête HTTP et le corps de requête HTTP ou la charge de requête HTTP.

  - **Class<T> expectedResponse**

    Cela doit indiquer le type de réponse attendu par l'URL distante. Les types Jackson et JAXB peuvent également être utilisés. La désérialisation de JSON/XML se produira automatiquement dans ce cas.

  En cas de réussite, cette méthode renvoie l'objet `com.hcl.unica.system.model.response.HttpResponse`, en encapsulant

l'objet de réponse à partir de l'appel distant. Le type de réponse encapsulé par `HttpResponse` sera le même que celui de l'argument `expectedResponse` de la méthode `executeService`. L'objet `HttpResponse` donne accès au code de statut de réponse HTTP, aux en-têtes de réponse et aux cookies de réponse, en plus de la charge de réponse.

Les deux versions de la méthode `executeService` peuvent générer l'exception `com.hcl.unica.system.integration.exception.ServiceExecutionException` ou l'un de ses sous-types si quelque chose se passe mal pendant l'exécution du service. L'objet de cette exception peut être consulté pour la cause immédiate de l'échec d'exécution de service. De même, si le service appelé représente un service REST/HTTP (les appels de service ad-hoc sont toujours des appels HTTP) et que l'échec se produit en dehors de l'interaction HTTP, un objet `HttpResponse` facultatif peut également être obtenu à partir de l'exception. Dans ce cas, l'exception `HttpServiceExecutionException` est renvoyée par les méthodes `executeService`. La présence de `HttpResponse` dépend du fait que l'interaction HTTP s'est produite ou non. L'exception `HttpServiceExecutionException` peut être reçue en raison d'une exception dans toute logique exécutée avant l'appel HTTP réel, telle que la méthode `buildRequest` d'un service déclaré.

La méthode `executeService` peut également renvoyer une exception `SystemNotFoundException` si le plug-in du système cible spécifié n'est pas présent ou si le système correspondant n'est pas intégré dans Unica Platform. De même, elle peut renvoyer une exception ServiceNotFoundException si le service spécifié n'est pas déclaré dans le fichier de déclaration de service ou n'est pas implémenté par le plug-in.

Note:

- Vous remarquerez que le type de l'entrée vers `custom-service` est le même que le type utilisé pour l'implémentation du service dans la classe `com.example.service.rest.CustomService` ou la classe `com.example.service.functional.CustomService`. Le type de sortie est le même que celui utilisé pour définir l'interface `CustomServiceGateway` dont l'objet de classe est renvoyé par la méthode `getServiceInterface` dans les deux versions d'implémentation `CustomService`.

- Les classes `com.example.service.rest.CustomService` et `com.example.service.functional.CustomService` représentent le même service implémenté avec deux approches différentes. Les fichiers de déclaration de service dans le projet `asset-integration-starter`, à savoir `META-INF/rest-content-services.yml` et `META-INF/functional-content-services.yml` ont une entrée pour `custom-service`, pointant vers les versions respectives de `factoryClass`. Ces deux versions sont uniquement fournies à titre d'illustration. A toutes fins pratiques, une seule version de l'implémentation du service est attendue par Content Integration Framework. Quelle que soit l'approche utilisée pour la mise en œuvre des services, la méthode d'appel des services reste la même.

Clients à plusieurs partitions

Depuis la perspective de l'implémentation du service, les objets `ExecutionContext` et `SystemConfig`, transmis aux différentes méthodes de rappel, contiennent des informations spécifiques à l'application client et à la partition. Depuis la perspective de l'appel de service, les services exécutés à l'aide de la méthode `executeService`, à partir de la classe `SystemGateway`, s'exécutent sur le système configuré pour l'application cliente appropriée et la partition de l'utilisateur qui accède à Unica Content Integration. Par conséquent, ni l'implémentation, ni l'utilisateur appelant ne doivent utiliser le partitionnement et d'autres détails contextuels de manière explicite. Content Integration Framework le gère automatiquement.

Contexte d'exécution

Presque toutes les méthodes du contrat d'implémentation de service reçoivent une instance de classe `com.hcl.unica.system.model.request.ExecutionContext`.

Cet objet contient toutes les informations contextuelles nécessaires pour qu'un service réalise son opération. Voici les méthodes de la classe ExecutionContext, qui peuvent être utilisées pour obtenir différents types d'informations pendant l'exécution du service :

- **T getRequest()**

Cette méthode peut être utilisée pour obtenir l'objet d'entrée ou de demande transmis au service lorsqu'il est exécuté à l'aide de la méthode `executeService` décrite dans Invocation de service (on page 30) (Le type de retour T est le paramètre de type correspondant à l'argument générique utilisé pour définir le service).

- **Map<String, Object> getAttributes()**

  Renvoie une mappe qui peut être utilisée pour stocker et extraire des attributs personnalisés pendant l'exécution du service. Elle est utile pour transporter des informations temporaires spécifiques à l'exécution sur plusieurs rappels. Par exemple, si l'implémentation de la méthode `buildRequest`depuis l'interface `RestService` ou l'interface `HttpService` doit partager certaines informations avec la méthode `transformResponse`, elle peut la partager à l'aide de cette mappe d'attributs.

  Il est important de noter que Content Integration Framework crée une instance distincte de `ExecutionContext` pour chaque appel de service individuel. Par conséquent, les attributs de contexte ne peuvent pas être partagés entre plusieurs exécutions de service. Leur portée est limitée à l'exécution d'un service individuel.

- **ServiceConfig getServiceConfig()**

  Cette méthode renvoie une instance de la classe `com.hcl.unica.system.integration.config.ServiceConfig`. L'objet `ServiceConfig` contient les configurations effectuées dans le fichier de déclaration de service pour le service respectif.

- **SystemConfig getSystemConfig()**

  Cette méthode renvoie une instance de la classe `com.hcl.unica.system.integration.config.SystemConfig`. L'objet `SystemConfig` contient toutes les configurations effectuées dans Unica Platform pour le système cible. Dans le cas de configurations à partitions multiples, cet objet sera correctement rempli par Content Integration Framework pour contenir la configuration spécifique à la partition pour l'application client concernée. Pour connaître les différents paramètres de configuration système dans Unica Platform, voir Unica Content Integration - Guide d'administration.

- **void setAttributes(Map<String, Object>)**

Cette méthode peut être utilisée pour spécifier des attributs dans `ExecutionContext` , qui peuvent être obtenus dans d'autres zones de l'implémentation du service. Cette fonctionnalité est utile pour partager des informations contextuelles personnalisées pendant l'exécution du service. La portée des attributs stockés dans le contexte d'exécution est limitée au flux d'exécution en cours uniquement. Les attributs ne peuvent pas être partagés entre plusieurs flux d'exécution du même service.

- **Locale getUserLocale ()**

  Cette méthode peut être utilisée pour obtenir les paramètres régionaux de l'utilisateur connecté.

Sources de données utilisateur

Unica Platform utilise des sources de données utilisateur pour stocker des informations sensibles, telles que des données d'identification d'API, des jetons de sécurité, des données d'identification d'utilisateur de base de données, etc. Les plug-ins doivent souvent stocker de tels détails de configuration. Content Integration fournit la configuration pertinente pour spécifier le nom de la source de données utilisateur et l'utilisateur Unica associé lors de l'intégration de systèmes à l'aide de la configuration Unica Platform.

Utilisez `ExecutionContext` pour obtenir la source de données utilisateur applicable (données d'identification) en naviguant jusqu'à l'objet `SystemConfig` :

```
executionContext.getSystemConfig().getDataSourceCredentials()
```

L'objet `DataSourceCredentials` renvoyé par la méthode `getDataSourceCredentials` contient la source de données sélectionnée basée sur la stratégie définie pour **Données d'identification utilisateur** dans la configuration de Platform. Par conséquent, les plug-ins ne devront prendre aucune décision logique concernant la sélection adéquate de la source de données utilisateur.

De même, la méthode `getUnicaToken` appelée sur l'objet `SystemConfig` renvoie un objet UnicaToken contenant le jeton Unica requis pour appeler les API d'applications Unica.

Services standard et types spécialisés

Le développeur de plug-ins doit implémenter l'interface `RestService`/`HttpService` ou `FunctionalService` afin de créer un service individuel.

Content Integration Framework tire parti de cette conception et définit certaines classes de service standard pour les services Recherche simple (`simple-search`), Répertorier des catégories de contenu (`list-content-categories`), Répertorier des dossiers (`list-folders`), Répertorier des contenus (`list-contents`), Obtenir des détails sur le contenu (`get-content-details`), Obtenir le schéma d'objet (`get-object-schema`) et Obtenir une analyse cognitive (`get-cognitive-analysis`). Le fichier standard-integrations.jar intégré au SDK de Content Integration fournit des versions spécialisées de `RestService` et `FunctionalService` pour chacun de ces services standard afin de faciliter leur implémentation à l'aide d'une approche RESTful ou fonctionnelle.

Appel de services standard

Une fois qu'ils sont déclarés dans le fichier de déclaration de service et implémentés à l'aide d'une approche RESTful ou fonctionnelle, Content Integration Framework appelle les services standard dans les scénarios suivants :

- **Recherche simple (`simple-search`)**

  Chaque fois qu'Content Integration Framework reçoit une demande de recherche de contenu ou d'actif de son application client par rapport au système cible, il appelle le service `simple-search` implémenté pour le système respectif. Content Integration Framework fournit une entrée nécessaire au service `simple-search` au moment de l'appel. Les éléments de recherche reçus du service `simple-search` sont ensuite renvoyés vers l'application client. L'identification du système cible se produit sur la base de la propriété `systemId` utilisée dans le fichier de déclaration de service et du paramètre **Identificateur système** correspondant dans Unica Platform, renseigné lors de l'intégration du système cible. Ce service doit être implémenté par le plug-in, sinon la demande de recherche de contenu aboutit en réponse 404 adressée à l'application cliente.

Le résultat de recherche produit par ce service peut être paginé ou non. La présence ou l'absence de prise en charge des résultats paginés doit être clairement indiquée à l'aide de la propriété paginatedSearch sous la section systems du fichier de déclaration de service, comme expliqué dans la rubrique Fichier de déclaration de service (on page 8).

- **Chargeur de ressource (`resource-loader`)**

  Le service `resource-loader` est exécuté par Content Integration Framework uniquement lorsqu'un accès indirect (ou authentifié) doit être établi au niveau de l'élément de recherche sur le système cible. La configuration peut être effectuée dans Unica Platform pour indiquer si le contenu est accessible directement (de façon anonyme) depuis le système cible ou non. Pour plus d'informations sur les configurations système, voir Unica Content Integration - Guide d'administration. Content Integration Framework fournit un service `resource-loader` par défaut à chaque système. Le service par défaut `resource-loader` charge simplement les ressources Web depuis le système cible en fournissant les informations d'autorisation nécessaires, le cas échéant. Les plug-ins peuvent choisir de remplacer le service `resource-loader` par défaut et d'inclure leur propre implémentation en étendant l'implémentation par défaut. Le téléchargement de contenu et le rendu de contenu peuvent échouer si l'implémentation `resource-loader` remplacée requise est manquante

- **Répertorier des catégories de contenu (`list-content-categories`)**

  S'il est implémenté, ce service est appelé pour extraire la liste des catégories de contenu prises en charge, utilisée pour remplir la liste déroulante de type de contenu dans l'interface utilisateur de Content Picker. Ces catégories permettent d'affiner la recherche de contenu au sein d'une catégorie particulière. D'autres cas d'utilisation concernant ces catégories pourront être présents dans les futures versions d'Unica Content Integration.

  Il s'agit d'un service facultatif et l'absence de son implémentation n'a pas d'impact sur la capacité de recherche de contenu dans Content Picker. D'autres alternatives sont utilisées à la place pour générer la liste des catégories de contenu pris en charge en l'absence de ce service, c'est-à-dire le paramètre standard `supportedContentTypes`

pour le service `simple-search` dans le fichier de déclaration de service ou la méthode `getSupportedContentTypes()` dans l'implémentation de service `simple-search`.

- **Répertorier des dossiers (`list-folders`)**

  Ce service permet de faciliter la navigation de contenu avec le service `list-contents`. Outre la recherche de contenu, le contenu peut également être localisé en naviguant dans la hiérarchie des dossiers (ou tout autre concept similaire dans le système respectif). Si ce service est implémenté, il est prévu qu'il fournisse des dossiers de niveau supérieur/racine ainsi que des sous-dossiers d'un dossier parent particulier lorsque l'utilisateur le demande lors de la navigation de contenu. Un seul niveau de liste de dossiers est attendu dans une exécution unique. L'intégralité de la hiérarchie de dossiers n'a pas besoin d'être fournie. Si ce service est implémenté, il est impératif d'implémenter également le service list-contents pour activer la fonction de navigation de contenu.

  Il s'agit d'un service facultatif et l'absence de son implémentation n'a pas d'impact sur la capacité de recherche de contenu dans Content Picker. Toutefois, la navigation de contenu est désactivée dans l'interface utilisateur de Content Picker si ce service n'est pas implémenté.

- **Répertorier des contenus (`list-contents`)**

  Ce service permet de faciliter la navigation de contenu avec le service `list-folders`. S'il est implémenté, ce service est censé fournir la liste des contenus appartenant à un dossier particulier. La liste peut être paginée ou non. La présence ou l'absence de prise en charge de la liste paginée doit être clairement indiquée à l'aide de la propriété `paginatedList` sous la section systems du fichier de déclaration de service, comme expliqué dans la rubrique Fichier de déclaration de service (on page 8).

  Si ce service est implémenté, il est impératif d'implémenter également le service `list-folders` pour activer la fonction de navigation de contenu.

  Il s'agit d'un service facultatif et l'absence de son implémentation n'a pas d'impact sur la capacité de recherche de contenu dans Content Picker. Toutefois, la navigation de contenu est désactivée dans l'interface utilisateur de Content Picker si ce service n'est pas implémenté.

- **Obtenir des détails sur le contenu (`get-content-details`)**

Tout contenu recherché à l'aide du service `simple-search` ou répertorié à l'aide du service `list-contents` peut être sélectionné et utilisé pour divers cas d'utilisation dans les applications Unica. De tels cas d'utilisation peuvent exiger ultérieurement les détails de contenus déjà choisis. Par exemple, la fonction d'aperçu de contenu dans Centralized Offer Management, dans laquelle les détails d'un contenu déjà lié avec l'attribut d'offre s'affichent. Chaque fois que les applications Unica ont besoin de détails sur un contenu individuel, le service get-content-details est appelé en fournissant l'identificateur unique du contenu requis.

Il s'agit d'un service facultatif et l'absence de son implémentation n'a pas d'impact sur la capacité de recherche de contenu dans Content Picker. Toutefois, les requêtes utilisateur ultérieures pour l'extraction des détails d'un contenu ne seront pas prises en charge si ce service n'est pas implémenté.

- **Obtenir le schéma d'objet (`get-object-schema`)**

  Ce service est appelé par les applications Unica pour extraire les détails de différents attributs présents dans le contenu. Le schéma maître complet de l'ensemble du contenu est attendu de ce service. Celui-ci doit inclure les détails de chaque attribut de contenu, tels que le type et le format de la valeur qu'il détient, ainsi qu'un identificateur unique permettant d'identifier de manière unique cet attribut pour le système donné. A partir de la version actuelle d'Unica Content Integration et d'Unica Centralized Offer Management, ces informations sont utilisées pour mapper les attributs de contenu à des attributs d'offre, puis pour remplir automatiquement les valeurs d'attribut d'offre en sélectionnant le contenu dans Content Picker. Pour plus d'informations sur cette fonctionnalité, reportez-vous au Guide d'utilisation d'Unica Centralized Offer Management.

  Il s'agit d'un service facultatif et l'absence de son implémentation n'a pas d'impact sur la capacité de recherche de contenu dans Content Picker. Toutefois, la fonction Content Integration dans Centralized Offer Management devient indisponible pour le système respectif si ce service n'est pas implémenté.

- **Obtenir une analyse cognitive (`get-cognitive-analysis`)**

  Ce service est appelé pour tenter l'analyse cognitive d'une image et extraire les détails cognitifs en conséquence. Elle n'est appelée que si le système respectif est

configuré en tant que fournisseur de services cognitifs préféré dans la configuration Platform. Pour en savoir davantage, reportez-vous à Unica Content Integration - Guide d'installation et de configuration.

Il s'agit d'un service facultatif et l'absence de son implémentation n'a pas d'impact sur la capacité de recherche de contenu ni sur toute autre fonction dans Content Picker. Toutefois, la fonctionnalité de balisage cognitif est désactivée dans Centralized Offer Management si ce service n'est pas disponible.

Types spécialisés

Voici les dérivés spécialisés des interfaces `RestService`, `HttpService` et `FunctionalService`, ainsi que leurs types associés pour tous les services standard. Utilisez le projet `asset-integration-starter` pour implémenter les détails mentionnés dans les rubriques suivantes :

- Dérivés du service Rest *(on page 41)*
- Dérivés du service Http *(on page 51)*
- Dérivés de service fonctionnel *(on page 53)*
- AbstractEntity *(on page 65)*
- Présentable *(on page 66)*

Dérivés du service Rest

Les dérivés de l'interface de service Rest simplifient la création de la mise en œuvre RESTful de services standard.

Recherche simple (`simple-search`)

Les interfaces et classes spécialisées disponibles pour le service `simple-search` sont les suivantes :

- `com.hcl.unica.system.integration.service.search.RestSearchService`

  La classe `com.example.service.rest.SimpleSearchService` du projet `asset-integration-starter` est une mise en œuvre rapide

pour le service `simple-search` RESTful. Son parent est la classe `com.hcl.unica.system.integration.service.search.RestSearchService`.

La classe `RestSearchService` a un paramètre de type RS, qui représente le type de réponse (post-désérialisation) reçue de l'API REST distante. Dans ce cas, il s'agit de la classe `SimpleSearchResponse` définie dans le projet `asset-integration-starter`.

La classe `RestSearchService` implémente l'interface `RestService` et définit la classe `SearchRequest` et tant qu'argument type RQ pour `RestService`. Dès lors, l'objet de `SearchRequest` devient une entrée pour tous les services `simple-search` (la même entrée est également utilisée pour la contrepartie fonctionnelle de la recherche simple). La classe `SearchRequest` fait partie du SDK Content Integration.

En plus de définir le type d'entrée pour le service `simple-search`, la classe `RestSearchService` écrase aussi la méthode `transformResponse` et définit une valeur de retour de cette méthode comme étant de type `ContentPage`. `ContentPage` fait aussi partie du SDK Content Integration et encapsule le résultat de la recherche et les détails de pagination associés.

Le plug-in doit étendre son implémentation `simple-search` depuis le service `com.hcl.unica.system.integration.service.search.RestSearchService` pour être reconnu en tant que service `simple-search` par Content Integration Framework (un équivalent fonctionnel, abordé plus tard, est également un choix valide pour une extension depuis les services `simple-search` implémentés à l’aide de l’approche fonctionnelle).

`RestSearchService` s'étend à partir de la classe abstraite `com.hcl.unica.system.integration.service.search .AbstractSearchService`.

Nous vous recommandons de vous référer à la classe `com.aem.service.AemSimpleSearchService` du projet `aem-integration` pour en savoir plus sur la manière dont les classes `SearchRequest` et `ContentPage` sont utilisées lors de l'implémentation du service.

Le respect du contrat de l'interface `Presentable` lors du remplissage de la liste de contenus dans `ContentPage` est une part essentielle de cette implémentation de service. L'interface `Presentable` est abordée plus en détail dans la section suivante.

- `com.hcl.unica.system.integration.service.search.AbstractSearchService`

  Il s'agit d'une classe de base commune pour des implémentations `simple-search` RESTful et fonctionnelles. Ainsi, les détails de cette classe s'appliquent également à l'implémentation fonctionnelle de `simple-search`.

  Cette classe définit l'interface `com.hcl.unica.system.integration.service.gateway.SimpleSearchServiceGateway` comme la passerelle de service pour le service `simple-search` : Les passerelles de service sont le moyen de définir par le programme les types d'entrée et de sortie du service et du travail avec le service. En regardant cette interface de plus près, on constate que le `simple-search` prend l'objet `SearchRequest` et renvoie l'objet `ContentPage`.

  En plus de définir l'interface de service pour `simple-search`, elle introduit une autre méthode abstraite pour le service `simple-search`, nommée `getSupportedContentTypes`. Chaque implémentation `simple-search` peut remplacer et implémenter cette nouvelle méthode de façon facultative. Veuillez noter que cette méthode est très spécifique à `simple-search` et n'a rien à voir avec d'autres services standard et personnalisés. La signature de cette méthode est la suivante :

  ```
  public Map<String, String> getSupportedContentTypes();
  ```

  L'implémentation de cette méthode renvoie une mappe `Map<String, String>` représentant les catégories de contenus à rechercher dans le système cible. Aucune sémantique spécifique n'est associée aux entrées de cette mappe. Il peut s’agir de n'importe quelle paire clé-valeur significative. Elle fait office de filtre pour l'application client pendant l'opération de recherche. Lors de l’implémentation actuelle de Unica Content Integration, cette mappe est utilisée pour renseigner les entrées dans une liste déroulante, où les clés de la mappe deviennent les valeurs des options, et les valeurs de la mappe deviennent des libellés d’affichage pour les options. Par conséquent, les clés peuvent porter des noms internes ou des identificateurs, et les valeurs doivent être du texte lisible et significatif. Si l’utilisateur doit rechercher un type de contenu spécifique, il peut choisir une ou plusieurs options parmi les types pris en charge. Dans ce cas, le service `simple-search` reçoit un groupe de clés correspondant aux valeurs choisies

par l'utilisateur. Vous pouvez obtenir un groupe de clés reçues de l'application client à partir de l'objet `ExecutionContext` en naviguant via la méthode `getRequest`, puis en y appelant `getTypes()`. L'implémentation `simple-search` traite ces ensembles de clés conformément à l'interface de programmation du système cible et filtre les éléments de recherche en conséquence.

Paramètre de service standard - supportedContentTypes

Ignorer la méthode `getSupportedContentTypes` n'est recommandé que si la mappe doit être générée de façon dynamique. Content Integration Framework fournit une autre approche pour définir statiquement cette mappe à l'aide d'un paramètre de service standard appelé `supportedContentTypes`, configuré sous l'élément `params` dans le fichier de déclaration de service. Par exemple, reportez-vous à la déclaration de service `simple-search` pour AEM et WCM à l'intérieur du fichier `<ASSET_PICKER_HOME>/conf/plugin-services.yml`.

Répertorier des catégories de contenu (`list-content-categories`)

Les interfaces et classes spécialisées disponibles pour le service `list-content-categories` sont les suivantes :

- ```
  com.hcl.unica.system.integration.service.content.categories.list.
   RestContentCategoriesListService
  ```

  Le `com.example.service.rest`. La classe ExampleContentCategoryListingService dans le projet asset-integration-starter constitue une mise en œuvre rapide pour le service RESTful list-content-categories. La classe ExampleContentCategoryListingService s'étend à partir de la classe RestContentCategoriesListService.

  La classe `RestContentCategoriesListService` a un paramètre de type RS, qui représente le type de réponse (post-désérialisation) reçue de l'API REST distante. Dans ce cas, il est spécifié comme `List<ContentCategoryDetails>` à titre d'exemple.

  La classe `RestContentCategoriesListService` implémente l'interface `RestService` et définit la classe `com.hcl.unica.system.model.request.content.categories`.ContentCategoryListRequest
  ```

et tant qu'argument type RQ pour RestService. Ainsi, l'objet de ContentCategoryListRequest devient une entrée pour tous les services list-content-categories (la même entrée est également utilisée pour la contrepartie fonctionnelle des catégories list-content-categories).

En plus de définir le type d'entrée pour le service list-content-categories, la classe RestContentCategoriesListService remplace également la méthode transformResponse et impose que la valeur de retour de cette méthode soit un objet de type `List<ContentCategory>`. La classe `ContentCategory` fait partie du SDK Content Integration.

Le plug-in doit étendre l'implémentation du service list-content-categories depuis la classe com.hcl.unica.system.integration.service.content.categories.list. RestContentCategoriesListService pour être reconnu en tant que service list-content-categories valide par Content Integration Framework (la contrepartie fonctionnelle, abordée plus loin, est également un choix valide pour effectuer l'extension).

`RestContentCategoriesListService` s'étend à partir de

```
com.hcl.unica.system.integration.service.content.categories.list.AbstractContentCateg
 class
```

.

- `com.hcl.unica.system.integration.service.content.categories.list.AbstractContentCatego`

Il s'agit d'une classe de base commune pour des implémentations `RESTful` et fonctionnelles du service `list-content-categories`. Ainsi, les détails abordés ici s'appliquent également à la version fonctionnelle de `list-content-categories`.

Cette classe définit l'interface `com.hcl.unica.system.integration.service.gateway.ContentCategoriesListServiceGateway` comme la passerelle de service pour le service `list-content-categories` : Cette interface s'étend à partir de l'interface `com.hcl.unica.system.integration.service.gateway.` `ServiceGateway` et impose que les objets `ContentCategoryListRequest & List<ContentCategory>` soient les types d'entrée et de sortie pour le service `list-content-categories`.

Répertorier des dossiers (`list-folders`)

Les interfaces et classes spécialisées disponibles pour le service `list-folders` sont les suivantes :

- `com.hcl.unica.system.integration.service.folder.list.RestFolderListService`

  La classe `com.aem.service.AemFolderListService` dans le projet `aem-integration` est une implémentation de référence pour le service RESTful list-folders. La classe `AemFolderListService` s'étend à partir de la classe `RestFolderListService`.

  La classe `RestFolderListService` a un paramètre de type **RS**, qui représente le type de réponse (post-désérialisation) reçue de l'API REST distante. Dans ce cas, il s'agit de la classe `SimpleSearchResponse` définie dans le projet `aem-integration`.

  La classe `RestFolderListService` implémente l'interface `RestService` et définit la classe `com.hcl.unica.system.model.request.folder.list.FolderListRequest` et tant qu'argument type RQ pour `RestService`. Dès lors, l'objet de `FolderListRequest` devient une entrée pour tous les services `list-folders` (la même entrée est également utilisée pour la contrepartie fonctionnelle de `list-folders`).

  En plus de définir le type d'entrée pour le service `list-folders`, la classe `RestFolderListService` remplace également la méthode `transformResponse` et impose que la valeur de retour de cette méthode soit un objet de type `List<Folder>`. Folder est un type standard défini dans le SDK de Content Integration.

  Le plug-in doit étendre l'implémentation du service `list-folders` depuis la classe `com.hcl.unica.system.integration.service.folder.list.RestFolderListService` pour être reconnu en tant que service list-folders valide par Content Integration Framework (la contrepartie RESTful abordée lors d'une section précédente est également un choix valide pour effectuer l'extension).

  `RestFolderListService` s'étend à partir de la classe `com.hcl.unica.system.integration.service.folder.list.AbstractFolderListService`.
- `com.hcl.unica.system.integration.service.folder.list.AbstractFolderListService`

Il s'agit d'une classe de base commune pour des implémentations RESTful et fonctionnelles du service `list-folders`. Ainsi, les détails abordés ici s'appliquent également à la version fonctionnelle de `list-folders`.

Cette classe définit l'interface `com.hcl.unica.system.integration.service.gateway.FolderListServiceGateway` comme la passerelle de service pour le service `list-folders` : Cette interface s'étend à partir de l'interface `com.hcl.unica.system.integration.service.gateway.ServiceGateway` et impose que les objets `FolderListRequest` et `List<Folder>` soient les types d'entrée et de sortie pour le service `list-folders`.

Répertorier des contenus (`list-contents`)

Les interfaces et classes spécialisées disponibles pour le service list-contents sont les suivantes :

- `com.hcl.unica.system.integration.service.content.list.RestContentListService`

  La classe `com.aem.service.AemContentListServiceclass` dans le projet `aem-integration` est une implémentation de référence pour le service RESTful `list-contents`. La classe `AemContentListServiceclass` s'étend à partir de la classe `RestContentListService`.

  La classe `RestContentListService` a un paramètre de type **RS**, qui représente le type de réponse (post-désérialisation) reçue de l'API REST distante. Dans ce cas, il s'agit de la classe `SimpleSearchResponse` définie dans le projet `aem-integration`.

  La classe `RestContentListService` implémente l'interface `RestService` et définit la classe `com.hcl.unica.system.model.request.content.list.ContentListRequest` et tant qu'argument type **RQ** pour `RestService`. Dès lors, l'objet de `ContentListRequest` devient une entrée pour tous les services `list-contents` (la même entrée est également utilisée pour la contrepartie fonctionnelle de `list-contents`).

  En plus de définir le type d'entrée pour le service `list-contents`, la classe `RestContentListService` remplace également la méthode `transformResponse` et impose que la valeur de retour de cette méthode soit un objet de type `ContentPage`.

Ce type de retour est le même que celui utilisé pour le service `simple-search`. `ContentPage` est un type standard défini dans le SDK de Content Integration.

Le plug-in doit étendre l'implémentation du service `list-contents` depuis la classe `com.hcl.unica.system.integration.service.content.list.RestContentListService` pour être reconnu en tant que service `list-contents` valide par Content Integration Framework (la contrepartie RESTful abordée lors d'une section précédente est également un choix valide pour effectuer l'extension).

`RestContentListService` s'étend à partir de la classe `com.hcl.unica.system.integration.service.content.list.AbstractContentListService`.

- `com.hcl.unica.system.integration.service.content.list.AbstractContentListService`

  Il s'agit d'une classe de base commune pour des implémentations RESTful et fonctionnelles du service `list-contents`. Ainsi, les détails abordés ici s'appliquent également à la version fonctionnelle de `list-contents`.

  Cette classe définit l'interface `com.hcl.unica.system.integration.service.gateway.ContentListServiceGateway` comme la passerelle de service pour le service `list-contents` : Cette interface s'étend à partir de l'interface `com.hcl.unica.system.integration.service.gateway.ServiceGateway` et impose que les objets `ContentListRequest` et `ContentPage` soient les types d'entrée et de sortie pour le service `list-contents`.

Obtenir des détails sur le contenu (`get-content-details`)

Les interfaces et classes spécialisées disponibles pour le service get-content-details sont les suivantes :

- `com.hcl.unica.system.integration.service.content.details.RestContentDetailsService`

  La classe `com.aem.service.AemObjectDetailsService` dans le projet `aem-integration` est une implémentation de référence pour le service RESTful `get-content-details`. La classe `AemObjectDetailsService` s'étend à partir de la classe `RestContentDetailsService`.

La classe `RestContentDetailsService` a un paramètre de type **RS**, qui représente le type de réponse (post-désérialisation) reçue de l'API REST distante. Dans ce cas, il s'agit de la classe `SimpleSearchResponse` définie dans le projet `aem-integration`.

La classe `RestContentDetailsService` implémente l'interface `RestService` et définit la classe `com.hcl.unica.system.model.request.content.details.ContentDetailsRequest` et tant qu'argument type **RQ** pour `RestService`. Dès lors, l'objet de `ContentDetailsRequest` devient une entrée pour tous les services `get-content-details` (la même entrée est également utilisée pour la contrepartie fonctionnelle de `get-content-details`).

En plus de définir le type d'entrée pour le service `get-content-details`, la classe `RestContentDetailsService` remplace également la méthode `transformResponse` et impose que la valeur de retour de cette méthode soit un objet de type Presentable.

Le plug-in doit étendre l'implémentation du service `get-content-details` depuis la classe `com.hcl.unica.system.integration.service.content.details.RestContentDetailsService` pour être reconnu en tant que service `get-content-details` valide par Content Integration Framework (la contrepartie RESTful abordée lors d'une section précédente est également un choix valide pour effectuer l'extension).

`RestContentDetailsService` s'étend à partir de la classe `com.hcl.unica.system.integration.service.content.details.AbstractContentDetailsService`

- `com.hcl.unica.system.integration.service.content.details.AbstractContentDetailsService`

Il s'agit d'une classe de base commune pour des implémentations RESTful et fonctionnelles du service `get-content-details`. Ainsi, les détails abordés ici s'appliquent également à la version fonctionnelle de `get-content-details`.

Cette classe définit l'interface `com.hcl.unica.system.integration.service.gateway.ContentDetailsServiceGateway` comme passerelle de service pour le service `get-content-details`. Les `ServiceGateways` permettent de définir, à l'aide d'un programme, les types d'entrée et de sortie du service et de faciliter l'appel des services. En regardant cette interface

de plus près, on constate que le service `get-content-details` accepte l'objet `ContentDetailsRequest` et renvoie un objet Presentable.

Obtenir une analyse cognitive (get-cognitive-analysis)

Les interfaces et classes spécialisées disponibles pour le service get-cognitive-analysis sont les suivantes :

- `com.hcl.unica.system.integration.service.cognitive.analysis.RestCognitiveAnalysisServi`

  La classe `com.example.service.rest.ExampleCognitiveAnalysisService` du projet `asset-integration-starter` est une mise en œuvre rapide pour le service `get-cognitive-analysis` RESTful. `ExampleCognitiveAnalysisService` de la classe s'étend à partir de la classe `RestCognitiveAnalysisService`.

  La classe `RestCognitiveAnalysisService` a un paramètre de type **RS**, qui représente le type de réponse (post-désérialisation) reçue de l'API REST distante. Dans ce cas, il s'agit de la classe `CognitiveDetails` définie dans le projet `asset-integration-starter`.

  La classe `RestCognitiveAnalysisService` implémente l'interface `RestService` et définit la classe `com.hcl.unica.system.model.request.cognitive.analysis.CognitiveAnalysisRequest` et tant qu'argument type **RQ** pour `RestService`. Dès lors, l'objet de `CognitiveAnalysisRequest` devient une entrée pour tous les services `get-cognitive-analysis` (la même entrée est également utilisée pour la contrepartie fonctionnelle).

  En plus de définir le type d'entrée pour le service `get-cognitive-analysis`, la classe `RestCognitiveAnalysisService` remplace également la méthode `transformResponse` et impose que la valeur de retour de cette méthode soit un objet de type `com.hcl.unica.system.model.response.cognitive.analysis.CognitiveAnalysis`. `CognitiveAnalysis` est un type standard définit dans le SDK de Content Integration.

  Le plug-in doit étendre l'implémentation du service `get-cognitive-analysis` depuis la classe `com.hcl.unica.system.integration.service.cognitive.analysis.RestCognitiveAnalysisServi` pour être reconnu en tant que service `get-cognitive-analysis` valide par

Content Integration Framework (la contrepartie RESTful abordée lors d'une section précédente est également un choix valide pour effectuer l'extension).

`RestCognitiveAnalysisService` s'étend à partir de la classe `com.hcl.unica.system.integration.service.cognitive.analysis.AbstractCognitiveAnalysisS`

- `com.hcl.unica.system.integration.service.cognitive.analysis.AbstractCognitiveAnalysisS`

  Il s'agit d'une classe de base commune pour des implémentations RESTful et fonctionnelles du service `get-cognitive-analysis`. Ainsi, les détails abordés ici s'appliquent également à la version fonctionnelle de `get-cognitive-analysis`.

  Cette classe définit l'interface `com.hcl.unica.system.integration.service.gateway.CognitiveAnalysisServiceGateway` comme la passerelle de service pour le service `get-cognitive-analysis` : Cette interface s'étend à partir de l'interface `com.hcl.unica.system.integration.service.gateway.ServiceGateway` et impose que les objets `CognitiveAnalysisRequest` et `CognitiveAnalysis` soient les types d'entrée et de sortie pour le service `get-cognitive-analysis`.

Dérivés du service Http

Seul le service standard `resource-loader` est implémenté en tant que `HttpService`, étant donné qu'il est lié à l'opération HTTP GET standard. Vous pouvez également utiliser `RestService` sans perte de capacité.

Chargeur de ressource (`resource-loader`)

Les interfaces et classes spécialisées disponibles pour le service de chargeur de ressource sont les suivantes :

- `com.hcl.unica.system.integration.service.resourceloader.DefaultWebResourceLoaderServic`

  La classe `com.example.service.rest.ResourceLoaderService` du projet `asset-integration-starter` est une mise en œuvre rapide pour le service `resource-loader` et s'étend à partir de la classe suivante :

  ```
  com.hcl.unica.system.integration.service.resourceloader
  .DefaultWebResourceLoaderService
  ```

La classe `DefaultWebResourceLoaderService` est l'implémentation par défaut du service `resource-loader` fourni par le SDK Content Integration. Si le plug-in n'implémente pas son propre service `resource-loader`, Content Integration Framework revient à cette implémentation par défaut. L'implémentation par défaut de `resource-loader` fournie par le SDK Content Integration suit simplement l'URL de ressource donnée et extrait la ressource Web du système cible. Elle encapsule l'opération HTTP GET standard.

Si le plug-in doit avoir sa propre implémentation `resource-loader`, qui modifie légèrement l'opération HTTP GET standard, nous recommandons une extension à partir de la classe `DefaultWebResourceLoaderService`. Il n'est pas nécessaire d'étendre l'implémentation `resource-loader` à partir de `DefaultWebResourceLoaderService` si le plug-in doit utiliser une approche totalement différente pour le chargement de contenus, comme la lecture depuis un système de fichiers, une base de données, un serveur FTP, etc. Dans ce cas, elle doit s'étendre depuis `HttpWebResourceLoaderService` pour une approche basée sur HTTP ou depuis `WebResourceLoaderService` pour une approche fonctionnelle.

- `com.hcl.unica.system.integration.service.resourceloader.HttpWebResourceLoaderService`

  La classe `DefaultWebResourceLoaderService` évoquée précédemment s'étend depuis la classe abstraite `HttpWebResourceLoaderService`. Cette classe définit le type d'entrée et le type de réponse HTTP reçue depuis l'URL cible pour le service `resource-loader` en tant que `com.hcl.unica.system.model.request.resourceloader.ResourceRequest` et `byte[]`, respectivement. La classe `ResourceRequest` encapsule l'URL de ressource et l'identifiant système. De même, `resource-loader` fonctionne avec un tableau d'octets lorsque le contenu de l'URL HTTP distante est lue avec succès.

  Si le plug-in n'étend pas son implémentation `resource-loader` depuis la classe `DefaultWebResourceLoaderService`, il doit au moins s'étendre à partir de la classe `com.hcl.unica.system.integration.service.resourceloader.HttpWebResourceLoaderService` pour être reconnu en tant que service `resource-loader` par Content Integration Framework (un équivalent fonctionnel, abordé plus tard, est également un choix valide

pour une extension depuis les services `resource-loader` implémentés à l'aide de l'approche fonctionnelle).

- `com.hcl.unica.system.integration.service.resourceloader.AbstractWebResourceLoaderServi`
  La classe `HttpWebResourceLoaderService` évoquée au point précédent s'étend depuis la classe abstraite `AbstractWebResourceLoaderService`. Cette classe définit l'interface de passerelle de service suivante pour le service `resource-loader` :

  ```
  com.hcl.unica.system.integration.service.gateway
  .ResourceLoaderServiceGateway
  ```

  Pour connaître le rôle des passerelles de service dans l'appel de service, voir Invocation de service (on page 30). L'interface `ResourceLoaderServiceGatewayy` définit `ResourceRequest` et `HttpResponse<?>` en tant que types d'entrée et de sortie pour le service `resource-loader`. `HttpResponse` est une interface, implémentée par la classe `WebResource`. Elle encapsule les en-têtes de réponse, le corps ou la charge HTTP, ainsi que les cookies reçus de l'URL distante. Même si le service personnalisé `resource-loader` ne récupère pas le contenu sur le Web, il doit renvoyer l'objet de `WebResource` (ou toute autre implémentation de `HttpResponse`) renseigné avec les détails appropriés. S'il n'est pas possible de renseigner de manière appropriée `WebResource`, des problèmes de chargement de contenu peuvent se produire pour les applications client. `WebResource` fournit une API de génération afin de créer un objet avec les détails nécessaires. La chose la plus importante consiste à remplir l'en-tête `Content-Type` de sorte que l'application client puisse traiter la charge en conséquence. De même, l'en-tête `Content-Disposition` doit également être rempli de manière appropriée et contenir le nom de fichier associé au contenu.

Dérivés de service fonctionnel

Les dérivés de l'interface de service fonctionnel simplifient la création de la mise en œuvre fonctionnelle de services standard. Le service fonctionnel n'est qu'un objet avec une méthode publique qui prend une certaine entrée et génère la sortie souhaitée.

Recherche simple (`simple-search`)

Les interfaces et classes spécialisées disponibles pour le service de recherche simple sont les suivantes :

- `com.hcl.unica.system.integration.service.search.SearchService`

  La classe `com.example.service.functional.SimpleSearchService` du projet `asset-integration-starter` est une mise en œuvre rapide pour le `simple-search service` fonctionnel. Elle s'étend à partir de la classe `com.hcl.unica.system.integration.service.search.SearchService`.

  La classe `SearchService` implémente l'interface `FunctionalService` et définit la classe `SearchRequest` et la classe `ContentPage` pour qu'elles soient les arguments types RQ et RS respectivement pour le service fonctionnel. Dès lors, l'objet de `SearchRequest` devient une entrée pour tous les services `simple-search` et le `ContentPage` devrait être une sortie lors de l'exécution du service.

  Le plug-in doit étendre son implémentation `simple-search` depuis le la classe `com.hcl.unica.system.integration.service.search.SearchService` pour être reconnu en tant que service `simple-search` par Content Integration Framework (la contrepartie RESTful abordée lors d'une section précédente est également un choix valide pour effectuer l'extension depuis les services `simple-search` implémentés à l'aide de l'approche RESTful).

  `SearchService` s'étend depuis la classe abstraite `com.hcl.unica.system.integration.service.search.AbstractSearchService`. Elle introduit une méthode supplémentaire, appelée `getSupportedContentTypes`. Pour plus d’informations sur la méthode, voir Dérivés du service Rest (on page 41).

Chargeur de ressource (`resource-loader`)

Les interfaces et classes spécialisées disponibles pour le service de chargeur de ressource sont les suivantes :

- `com.hcl.unica.system.integration.service.resourceloader.WebResourceLoaderService`

La classe `com.example.service.functional.ResourceLoaderService` du projet `asset-integration-starter` est une mise en œuvre rapide pour le service `resource-loader` fonctionnel. Elle s'étend depuis la classe suivante :

```
com.hcl.unica.system.integration.service.resourceloader.WebResourceLoaderService
```

La classe `WebResourceLoaderService` implémente l'interface `FunctionalService` et définit les types `ResourceRequest` et `HttpResponse` pour qu'ils soient, respectivement, les arguments types RQ et RS pour `FunctionalService`. Dès lors, l'objet de `ResourceRequest` devient une entrée pour tous les services `resource-loader` et le `HttpResponse` ou son sous-type devrait être une sortie lors de l'exécution du service (les mêmes types d'entrée et de sortie sont utilisés pour la contrepartie RESTful de resource-loader). Pour plus d'informations sur les types `ResourceRequest` et `HttpResponse`, voir Dérivés du service Rest (on page 41).

Le plug-in doit étendre son implémentation `resource-loader` depuis le service `com.hcl.unica.system.integration.service.resourceloader.WebResourceLoaderService` pour être reconnu en tant que service `resource-loader` par Content Integration Framework (la contrepartie RESTful abordée dans une section précédente est également un choix valide pour effectuer l'extension depuis les services `resource-loader` à l'aide de l'approche HTTP).

WebResourceLoaderService s'étend depuis la classe suivante :

```
com.hcl.unica.system.integration.service.resourceloader.
AbstractWebResourceLoaderService
```

Pour plus d'informations sur cette classe, voir Dérivés du service Rest (on page 41).

Répertorier des catégories de contenu (`list-content-categories`)

Les interfaces et classes spécialisées disponibles pour le service list-content-categories sont les suivantes :

- `com.hcl.unica.system.integration.service.content.categories.list.ContentCategoriesList`

  Le plug-in peut également choisir l'approche fonctionnelle pour implémenter le service `list-content-categories` en étendant l'implémentation à partir de la

classe `ContentCategoriesListService`. La classe `ContentCategoriesListService` implémente l'interface `FunctionalService` et impose les classes `ContentCategoryListRequest` et `List<ContentCategory>` pour qu'elles soient les arguments types **RQ** et **RS** respectivement pour le `FunctionalService`. Dès lors, l'objet de `ContentCategoryListRequest` devient une entrée pour le service `list-content-categories` et l'objet de type `List<ContentCategory>` est attendu en tant que sortie lors de l'exécution du service.

- Le plug-in doit étendre son implémentation list-content-categories depuis la classe `com.hcl.unica.system.integration.service.content.categories.list.ContentCategoriesList` pour être reconnu en tant que service `list-content-categories` valide par Content Integration Framework (la contrepartie RESTful abordée lors d'une section précédente est également un choix valide pour effectuer l'extension).

  `ContentCategoriesListService` s'étend à partir de la classe `AbstractContentCategoriesListService`. Les détails de la classe `AbstractContentCategoriesListService` sont couverts dans la rubrique Dérivés du service Rest (on page 41).

Répertorier des dossiers (`list-folders`)

Les interfaces et classes spécialisées disponibles pour le service list-folders sont les suivantes :

- `com.hcl.unica.system.integration.service.folder.list.FolderListService`

  Le plug-in peut également choisir l'approche fonctionnelle pour implémenter le service `list-folders` en étendant l'implémentation à partir de la classe `FolderListService`. La classe `FolderListService` implémente l'interface `FunctionalService` et impose les classes `FolderListRequest` et `List<Folder>` pour qu'elles soient les arguments types **RQ** et **RS** respectivement pour le `FunctionalService`. Dès lors, l'objet de `FolderListRequest` devient une entrée pour le service `list-folders` et l'objet de type `List<Folder>` est attendu en tant que sortie lors de l'exécution du service.
- Le plug-in doit étendre son implémentation list-folders depuis la classe `com.hcl.unica.system.integration.service.folder.list.FolderListService` pour être reconnu en tant que service `list-folders` valide par

Content Integration Framework (la contrepartie RESTful abordée lors d'une section précédente est également un choix valide pour effectuer l'extension).

`FolderListService` s'étend à partir de la classe `AbstractFolderListService`. Les détails de la classe `AbstractFolderListService` sont couverts dans la rubrique Dérivés du service Rest (on page 41).

Répertorier des contenus (`list-contents`)

Les interfaces et classes spécialisées disponibles pour le service list-contents sont les suivantes :

- `com.hcl.unica.system.integration.service.content.list.ContentListService`

  Le plug-in peut également choisir l'approche fonctionnelle pour implémenter le service `list-contents` en étendant l'implémentation à partir de la classe `ContentListService`. La classe `ContentListService` implémente l'interface `FunctionalService` et impose les classes `ContentListRequest` et `ContentPage` pour qu'elles soient les arguments types **RQ** et **RS** respectivement pour le `FunctionalService`. Dès lors, l'objet de `ContentListRequest` devient une entrée pour le service `list-contents` et l'objet de type `ContentPage` est attendu en tant que sortie lors de l'exécution du service.
- Le plug-in doit étendre son implémentation list-contents depuis la classe `com.hcl.unica.system.integration.service.content.list.ContentListService` pour être reconnu en tant que service list-contents valide par Content Integration Framework (la contrepartie RESTful abordée lors d'une section précédente est également un choix valide pour effectuer l'extension).

  `ContentListService` s'étend à partir de la classe `AbstractContentListService`. Les détails de la classe `AbstractContentListService` sont couverts dans la rubrique Dérivés du service Rest (on page 41).

Obtenir des détails sur le contenu (`get-content-details`)

Les interfaces et classes spécialisées disponibles pour le service get-content-details sont les suivantes :

- `com.hcl.unica.system.integration.service.content.details.ContentDetailsService`

Le plug-in peut également choisir l'approche fonctionnelle pour implémenter le service `get-content-details` en étendant l'implémentation à partir de la classe `ContentDetailsService`.

La classe `ContentDetailsService` implémente l'interface `FunctionalService` et définit les classes `ContentDetailsRequest` et Presentable pour qu'elles soient les arguments types **RQ** et **RS** respectivement pour le `FunctionalService`. Dès lors, l'objet de `ContentDetailsRequest` devient une entrée pour le service `get-content-details` et l'objet de type Presentable est attendu en tant que sortie lors de l'exécution du service.

Le plug-in doit étendre son implémentation `get-content-details` depuis la classe `com.hcl.unica.system.integration.service.content.details.ContentDetailsService` pour être reconnu en tant que service `get-content-details` valide par Content Integration Framework (la contrepartie RESTful abordée lors d'une section précédente est également un choix valide pour effectuer l'extension).

`ContentDetailsService` s'étend à partir de la classe `AbstractContentDetailsService`. Les détails de la classe `AbstractContentDetailsService` sont couverts dans la rubrique Dérivés du service Rest (on page 41).

Obtenir le schéma d'objet (`get-object-schema`)

Le service `get-object-schema` est utilisé pour générer le schéma maître de l'objet ou entité de domaine utilisé par le système respectif pour représenter le contenu. Le schéma maître, dans sa forme la plus simple, n'est qu'une métadonnée hiérarchique de chaque attribut de contenu mappable. La hiérarchie d'attributs et les métadonnées doivent correspondre à la représentation JSON de l'objet de domaine. Les métadonnées d'attribut incluent principalement le type de données de l'attribut, le format de la valeur détenue dans l'attribut, l'identificateur unique de l'attribut et le titre ou libellé d'affichage de l'attribut.

Les interfaces et classes spécialisées disponibles pour le service `get-object-schema` sont les suivantes :

- `com.hcl.unica.system.integration.service.object.schema.ObjectSchemaProviderService`

La classe `ObjectSchemaProviderService` implémente l'interface `FunctionalService` et impose les classes `com.hcl.unica.system.model.ObjectSchemaRequest` et `com.hcl.unica.system.model.json.schema.ObjectSchema` pour qu'elles soient les arguments types **RQ** et **RS** respectivement pour le `FunctionalService`. Dès lors, l'objet de `ObjectSchemaRequest` devient une entrée pour le service `get-object-schema` et l'objet de type `ObjectSchema` est attendu en tant que sortie lors de l'exécution du service. Le plug-in n'a toutefois pas besoin de construire `ObjectSchema` par lui-même. Il doit simplement remplacer et implémenter la méthode abstraite suivante à partir de la classe `ObjectSchemaProviderService`.

ObjectProfile getObjectProfile(ObjectSchemaRequest objectSchemaRequest)

La méthode `getObjectProfile()` accepte `ObjectSchemaRequest` et renvoie `ObjectProfile`. (Ces types sont abordés dans la section suivante.)

Le plug-in doit étendre son implémentation `get-object-schema` depuis la classe `com.hcl.unica.system.integration.service.object.schema.ObjectSchemaProviderService` pour être reconnu en tant que service `get-object-schema` valide par Content Integration Framework. Il n'y a pas de contrepartie RESTful de cette super classe standard, car la génération de schéma d'objet n'inclut aucune interaction HTTP. Les plug-ins peuvent implémenter un service RESTful personnalisé et l'appeler en interne à partir du service `get-object-schema`, si nécessaire.

- `com.hcl.unica.system.model.ObjectSchemaRequest`

  L'objet de cette classe est fourni en tant qu'entrée au service `get-object-schema`. La méthode la plus importante de cette classe est `getObjectIdentity()`, qui renvoie un objet de type `com.hcl.unica.system.model.ObjectIdentity` encapsulant les détails du contenu choisi par l'utilisateur pour demander le schéma maître. Il inclut `applicationId` (identificateur système), `objectType` (type de contenu/identificateur de catégorie) et `objectId` (identificateur unique du contenu sélectionné). Quels que soient la catégorie et/ou le contenu choisis par l'utilisateur au moment de la configuration du mappage de contenu, le schéma généré doit inclure des attributs de tous les types de contenu pris en charge par le système respectif. En d'autres termes, un seul schéma maître est utilisé pour mapper tous les types de contenu fournis par le système donné.

La méthode `getEnrichmentObjectJson()` de la classe `ObjectSchemaRequest` peut être ignorée à partir de la version actuelle.

- `com.hcl.unica.system.integration.service.object.schema.ObjectProfile`

  Il s'agit d'un type de retour de méthode `getObjectProfile()` dans le service `get-object-schema`. Il comporte le type Java correspondant à l'entité de domaine/objet pour le système respectif. Content Integration Framework consulte ce type Java afin de générer le schéma pour les propriétés de classe non statiques publiques et non publiques (y compris Enums & Optionals). L'annotation `@MappableAttribute` peut être utilisée pour configurer chaque propriété de classe individuelle afin de contrôler le schéma généré par Content Integration Framework. Reportez-vous à l'objet de domaine `com.aem.model.response.simplesearch.SimpleSearchItem` dans le projet `aem-integration reference` pour avoir une idée de la façon dont cette annotation est utilisée. Des informations plus détaillées sur `@MappableAttribute` sont fournies dans la section suivante. `ObjectProfile` peut éventuellement inclure une instance de `com.hcl.unica.system.integration.service.object.schema.ObjectSchemaEnricher` pour ajouter/modifier/supprimer de façon dynamique des attributs du schéma ainsi généré. La section suivante explique `ObjectSchemaEnricher` en détail.

- `com.hcl.unica.system.integration.service.object.schema.ObjectSchemaEnricher`

  `ObjectSchemaEnricher` est une classe abstraite. Le plug-in doit l'étendre pour obtenir l'implémentation souhaitée. Le paramètre type de la classe `ObjectSchemaEnricher` représente le type Java contenant les détails supplémentaires requis pour enrichir le schéma d'objet généré de manière statique. Ces détails supplémentaires peuvent éventuellement être fournis par les applications client d'Unica Content Integration. A partir de la version actuelle, aucun détail supplémentaire n'est fourni. Par conséquent, le paramètre doit être réglé sur Void lors de l'implémentation de l'enrichisseur de schéma. `ObjectSchemaEnricher` déclare une seule méthode abstraite qui doit être implémentée par le plug-in :

```
abstract public ObjectSchema enrich(
  ObjectSchema objectSchema,
  ObjectSchemaEnrichmentRequest<T> objectSchemaEnrichmentRequest
)
```

Le premier argument de cette méthode est une instance de la classe `com.hcl.unica.system.model.json.schema.ObjectSchema`. Elle contient le schéma d'objet de domaine généré automatiquement, dérivé du type Java fourni dans `ObjectProfile`. Essentiellement, `ObjectSchema` est juste un `Map<String, AttributeSchema>`, dans lequel les noms de propriété de classe forment les clés de cette mappe et les métadonnées de propriété deviennent des objets de `AttributeSchema`. Si la propriété de classe fait à son tour référence à un autre objet, l'objet `AttributeSchema` correspondant aura un autre `Map<String, AttributeSchema>` contenant les attributs de ce type d'objet, etc.

Note: Il est important de noter que les noms d'attribut utilisés en tant que clés dans une mappe d'attributs correspondent aux propriétés JSON qui finissent dans la représentation JSON de l'objet de domaine. Par conséquent, si l'annotation `@JsonProperty` est utilisée pour remplacer le nom de propriété JSON pour certains attributs de classe, Content Integration Framework la détecte automatiquement et utilise le nom de propriété remplacé.

`ObjectSchema` et `AttributeSchema` s'étendent à partir de la classe abstraite `com.hcl.unica.system.model.json.schema.AttributeContainer`. `AttributeContainer` fournit des méthodes pratiques aux classes `ObjectSchema` et `AttributeSchema` pour naviguer dans la hiérarchie des attributs, ainsi que pour ajouter, modifier et supprimer des attributs à n'importe quel niveau de la hiérarchie afin de faciliter l'enrichissement du schéma. Les attributs à n'importe quel niveau de la hiérarchie sont accessibles et manipulés à l'aide de leurs noms tels qu'ils apparaissent dans la représentation JSON.

- `com.hcl.unica.system.model.json.schema.generator.annotations.MappableAttribute`

  L'annotation `@MappableAttribute` permet de contrôler la façon dont Content Integration Framework génère un schéma d'objet à partir du type Java respectif. L'utilisation de `@MappableAttribute` n'est pas obligatoire. S'il n'est pas utilisé, Content Integration Framework détecte automatiquement les métadonnées de propriété. Si nécessaire, cette annotation doit être placée au-dessus des propriétés de classe souhaitées. Les attributs d'annotation suivants peuvent être utilisés pour contrôler la génération du schéma :

- **hidden** – Définissez cette propriété sur true pour exclure explicitement certaines propriétés du schéma d'objet (`@JsonIgnore` n'est actuellement pas pris en compte par Content Integration Framework. Par conséquent, toute propriété exclue de la représentation JSON à l'aide de `@JsonIgnore` doit être explicitement exclue du schéma)
- **id** – Fournissez un identificateur unique pour la propriété.
  Content Integration Framework a besoin d'un identificateur unique pour chaque propriété de classe mappable. Si `@MappableAttribute` n'est pas utilisé ou si l'**ID** n'est pas spécifié, il en génère un automatiquement en fonction de l'emplacement de la propriété dans la classe.

  La génération automatique de l'identificateur d'attribut est soumise au nom et à l'emplacement hiérarchique de la propriété de classe dans le graphique d'objet de domaine. Cela implique que si le nom de la propriété est modifié et/ou déplacé vers le haut ou vers le bas dans la hiérarchie du graphique d'objet, l'identificateur qui lui est associé sera modifié. Une telle restructuration peut tromper Content Integration Framework lors de la lecture des valeurs des attributs restructurés et peut entraîner des données non prévues dans des contenus mappés (tels que les offres dans COM). Par conséquent, pour éviter de telles modifications involontaires dans les identificateurs d'attribut, nous vous recommandons d'affecter manuellement des identificateurs d'attribut uniques, qui restent constants quels que soient le nom et l'emplacement des propriétés de classe.
- **title** – Titre/libellé d'affichage de la propriété. S'il est omis,
  Content Integration Framework en génère un à l'aide du nom de la propriété.
- **type** – L'une des valeurs de
  `com.hcl.unica.system.model.json.schema.generator.annotations.AttributeType`.
  S'il est omis, Content Integration Framework détecte automatiquement le type approprié.
- **format** – L'une des valeurs de
  `com.hcl.unica.system.model.json.schema.generator.annotations.AttributeFormat`.
  Content Integration Framework peut identifier automatiquement les types Java

Temporal standard (`Date`, `LocalDateTime`, `Instant`) et définir le type d'attribut sur `DATETIME`. D'autres formats doivent être explicitement déclarés.

- **implementation** – Doit être utilisée pour les références polymorphes afin de déclarer explicitement le type Java à prendre en compte pour la génération automatique de schéma.
- **hiddenProperties** –L'annotation `@MappableAttribute` peut être utilisée au niveau de la classe pour masquer plusieurs propriétés à un seul endroit. `hiddenProperties` prend un tableau de chaînes contenant les noms des propriétés (directes et héritées) à exclure du schéma généré automatiquement. Elle est particulièrement utile pour masquer les propriétés héritées d'une classe parent tierce.

Mappage de Type Java à AttributeType

Le tableau suivant récapitule le mappage entre le type Java et AttributeType/AttributeFormat utilisé par Content Integration Framework pour la génération automatique de schéma :

Type Java	AttributeType	AttributeFormat
◦ `String` ◦ `Character` ◦ `Char` ◦ `CharSequence` ◦ `LocalDate` ◦ `LocalTime` ◦ `ZonedDateTime` ◦ `OffsetDateTime` ◦ `OffsetTime` ◦ `ZoneId` ◦ `Calendar` ◦ `UUID`	`STRING`	
◦ `Boolean` ◦ `boolean`	`BOOLEAN`	
◦ `BigInteger`	`INTEGER`	

Type Java	AttributeType	AttributeFormat
◦ `Integer` ◦ `Int` ◦ `Long` ◦ `Long` ◦ `Short` ◦ `Short` ◦ `Byte` ◦ `byte`		
◦ `BigDecimal` ◦ `Number` ◦ `Double` ◦ `Double` ◦ `Float` ◦ `float`	`NUMBER`	
◦ `Date` ◦ `LocalDateTime` ◦ `Instant` Content Integration Framework s'attend à ce que les valeurs de date soient exprimées en heure UTC standard. Les valeurs temporelles exprimées dans tout autre fuseau horaire peuvent être à l'origine d'erreurs de calcul de temps dans d'autres cas d'utilisation.	`INTEGER`	`DATETIME`

Obtenir une analyse cognitive (`get-cognitive-analysis`)

Les interfaces et classes spécialisées disponibles pour le service get-cognitive-analysis sont les suivantes :

- `com.hcl.unica.system.integration.service.cognitive.analysis.CognitiveAnalysisService`

  Le plug-in peut également choisir l'approche fonctionnelle pour implémenter le service `get-cognitive-analysis` en étendant l'implémentation à partir de la classe `CognitiveAnalysisService`. La classe `CognitiveAnalysisService` implémente l'interface `FunctionalService` et impose les classes `CognitiveAnalysisRequest` et `CognitiveAnalysis` pour qu'elles soient les arguments types **RQ** et **RS** respectivement pour le `FunctionalService`. Dès lors, l'objet de `CognitiveAnalysisRequest` devient une entrée pour le service `get-cognitive-analysis` et l'objet de type `CognitiveAnalysis` est attendu en tant que sortie lors de l'exécution du service.
- Le plug-in doit étendre son implémentation `get-cognitive-analysis` depuis la classe `com.hcl.unica.system.integration.service.cognitive.analysis.CognitiveAnalysisService` pour être reconnu en tant que service `get-cognitive-analysis` valide par Content Integration Framework (la contrepartie RESTful abordée lors d'une section précédente est également un choix valide pour effectuer l'extension).

  CognitiveAnalysisService s'étend à partir de la classe `AbstractCognitiveAnalysisService`. Les détails de la classe `AbstractCognitiveAnalysisService` sont abordés dans la rubrique Dérivés du service Rest (on page 41).

AbstractEntity

La classe `com.hcl.unica.system.model.AbstractEntity` représente une entité de domaine générale. Pour la version en cours, cette classe abstraite ne contient aucune implémentation.

Toutefois, pour Content Integration Framework, les plug-ins doivent étendre leurs entités de domaine de la classe `com.hcl.unica.system.model.AbstractEntity`. Cela permet de s'assurer que `AbstractEntity` est la base pour traiter les entités de domaine au sein de Content Integration Framework.

Comme pour les implémentations de plug-in, la classe utilisée pour représenter un contenu individuel renvoyé par les services `simple-search`, `list-contents` et `get-content-details` doit s'étendre à partir de la classe `AbstractEntity`.

Présentable

Pour pouvoir rendre un contenu individuel renvoyé par les services simple-search, list-contents et get-content-details, la classe d'entité de domaine utilisée par ces services doit implémenter l'interface `com.hcl.unica.system.model.presentation.Presentable` et remplacer la méthode `getPresentationDetails()`. L'objet `com.hcl.unica.system.model.presentation.Presentable$PresentationDetails` renvoyé par la méthode `getPresentationDetails()` doit fournir le `TextualPresentation`, ainsi que des détails `MultimediaPresentation`.

`TextualPresentation` contient les détails suivants :

- **Note:** Les champs en gras sont obligatoires. Pour les autres champs, fournissez des détails, si disponibles.
- **`heading` – Titre du contenu**
- `subheadings` – Liste des sous-en-têtes pour le contenu
- `summary` – Récapitulatif ou description du contenu
- **`name` – Doit être utilisé pour le nom de fichier associé au contenu**
- `tags` – Les balises associées au contenu (les plug-ins prêts à l'emploi l'utilisent pour communiquer le type ou la catégorie MIME du contenu)

Alors que `MultimediaPresentation` contient les détails suivants :

- **Note:** Les champs en gras sont obligatoires. Pour les autres champs, fournissez des détails, si disponibles.
- **`id` – Identificateur unique du contenu**
- **`folderId` – Identificateur unique du dossier auquel le contenu respectif appartient**
- **`mimeType` – Type MIME du contenu d'origine**
- `size` – Taille du contenu d'origine en octets
- **`resourceUrl` – URL absolue menant au contenu d'origine**
- `thumbnailUrl` – URL absolue menant à la miniature du contenu, si celle-ci est disponible
- **`fileName` – Nom de fichier associé au contenu d'origine**

- `type` **– Identificateur de type/catégorie du contenu (doit être l'une des valeurs des types de contenu pris en charge, définies à l'aide de l'une des alternatives applicables fournies par Content Integration Framework)**
- `list of variants` – Chaque variante prend en charge presque les mêmes détails que les détails MultimediaPresentation principaux, à l'exception de thumbnailUrl (elle ne peut avoir que sa propre resourceUrl), folderId et ses variantes (une variante ne peut pas avoir d'autres variantes)

API de générateur

Presque tous les types standard abordés dans les sections précédentes fournissent l'API de générateur pour faciliter la construction d'objets.

Par exemple, `TextualPresentation` peut être créé à l'aide de la syntaxe suivante au lieu de la fractionner en opérations de constructeur et de configurateur :

```
TextualPresentation.builder()
 .heading("Content title")
 .subheadings(Collections.emptyList())
 .name("photo.jpg")
 .tags(Collections.singletonList("Image"))
 .build();
```

Il n'est pas obligatoire d'utiliser l'API de générateur pour créer des objets standard. Toutefois, elle permet sans aucun doute d'implémenter des plug-ins de façon propre tout en traitant des objets complexes.

Exceptions standard

Les exceptions standard comprennent les exceptions fournies par le SDK Content Integration, qui peuvent être utilisées par les plug-ins afin de transmettre différentes conditions de défaillance pendant l'exécution du service.

Approche RESTful

Content Integration Framework gère les conditions d'erreur résultant des services mis en œuvre à l'aide de l'approche RESTful.

En outre, Content Integration Framework lance et gère l'exécution d'un appel d'API distant pour les intégrations RESTful, afin qu'il puisse suivre le bon déroulement de toutes les opérations HTTP. Dès lors, les plug-ins ne nécessitent aucune exception spéciale pour communiquer l'échec de l'appel REST. Si quelque chose se passe mal dans la mise en œuvre du service, toute exception non vérifiée appropriée suffit pour communiquer l'échec de l'opération. Ces exceptions sont transmises en tant que réponse HTTP 502 au client.

Approche fonctionnelle

Etant donné qu'Content Integration Framework n'initie et ne gère pas les connexions sortantes en cas de services fonctionnels, il ne peut pas suivre le bon déroulement de bout en bout.

Par conséquent, il fournit certaines exceptions standard, que les implémentations de service peuvent déclencher pour transmettre les conditions d'échec pertinentes. Ces exceptions sont liées à la communication avec le système cible et sont présentes au sein du package `com.hcl.unica.system.integration.exception`.

- **SystemNotFoundException**

  Cette exception doit être utilisée lorsqu'il n'est pas possible de localiser le système cible ou le référentiel de contenu. De même, il est possible d'utiliser également `java.net.UnknownHostException`. Cette exception est transmise en tant que réponse HTTP 404 au client.
- **ServiceNotFoundException**

  Cette exception doit être utilisée lorsque le point de terminaison distant renvoie 404 ou si le service cible n'existe plus. L'absence du système cible et l'absence du service requis sont considérées comme des choses différentes. Par conséquent, le `ServiceNotFoundException` véhicule la présence du système cible et l'absence du service requis, ou de la fonctionnalité, sur le système cible. Par exemple, en cas de contenu extrait de la base de données, l'absence de la table requise (ou l'absence

de droit d'accès) peut être transmise à l'aide de cette exception. Cette exception est transmise en tant que réponse HTTP 404 au client.

- **UnreachableSystemException**

  Cette exception doit être utilisée pour transmettre des systèmes cible inaccessibles ou inaccessibles, tels que le délai d'expiration de la connexion. De même, il est possible d'utiliser également `java.net.ConnectException`. Cette exception est transmise en tant que réponse HTTP 503 au client.

- **SluggishSystemException**

  Lorsque la réponse du système cible n'est pas reçue au cours du délai prévu, cette exception doit être utilisée pour communiquer la lenteur du système cible. De même, il est possible d'utiliser également `java.net.SocketTimeoutException`. Cette exception est transmise en tant que réponse HTTP 504 au client.

- **InternalSystemError**

  Cette exception doit être utilisée si le plug-in reçoit une erreur temporaire ou inattendue du système cible pour communiquer les problèmes qu'il contient. Cette exception est transmise en tant que réponse HTTP 502 au client.

Toutes les autres exceptions sont transmises en tant que réponse HTTP 502 au client. Dans tous les cas, le message de l'exception n'est jamais renvoyé au client. Chaque code de réponse HTTP contient un message fixe, générique et localisé.

Content Integration Framework encapsule les exceptions reçues des implémentations de service dans `com.hcl.unica.system.integration.exception.ServiceExecutionException` ou dans son sous-type. Les exceptions reçues à partir de services REST ou de services basés sur HTTP sont encapsulées dans `com.hcl.unica.system.integration.exception.HttpServiceExecutionException`, tandis que celles reçues des services fonctionnels sont encapsulées dans `com.hcl.unica.system.integration.exception.ServiceExecutionException`.

Comme expliqué dans Invocation de service (on page 30), `HttpServiceExecutionException` fournit une méthode pour obtenir un objet

`Optional<HttpResponse>`. Si l'exécution du service échoue avant de lancer un appel HTTP, cet objet facultatif ne contient aucune `HttpResponse`.

Gestionnaires de journalisation

Content Integration Framework fournit l'interface de journalisation à l'aide de la bibliothèque `slf4j`. En ajoutant des dépendances pour la bibliothèque `slf4j`, les plug-ins peuvent utiliser son API pour ajouter des enregistreurs dans les implémentations de service.

Le déclencher, ainsi que les projets de référence inclus dans `dev-kits` gèrent leurs dépendances à l'aide d'Apache Maven. L'entrée suivante se trouve dans le fichier POM :

```
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>1.7.26</version>
 </dependency>
```

Utilisez la version 1.7.26 ou ultérieure de `slf4j-api` pour éviter tout conflit. Une fois la dépendance requise ajoutée, il est possible d'obtenir l'objet du consignateur en accédant directement à l'API `slf4j`.

```
Logger log = LoggerFactory.getLogger(YOUR_CLASS.class);
```

Alternativement, le projet Lombok peut également être utilisé pour obtenir l'objet du consignateur pour votre classe. Lombok fournit une annotation @Slf4j annotation, qui peut être utilisé pour injecter la propriété mentionnée précédemment dans la classe annotée. Pour de plus amples informations sur le projet Lombok, consultez sa page Web officielle.

En outre, les journaux d'application se trouvent dans le répertoire `AssetPicker/logs` sous l'accueil de la plateforme. Par défaut, tous les consignateurs de votre plug-in se trouveront dans le fichier journal standard configuré dans le fichier `AssetPicker/conf/logging/log4j2.xml`. Vous pouvez modifier le fichier de configuration `log4j2.xml` afin d'acheminer les consignateurs vers un autre fichier, à des fins de résolution des incidents lors du développement. La configuration de `log4j2` n'entre pas dans le champ

d'application de ce guide. Reportez-vous à la documentation officielle d'Apache Log4j2 pour plus d'informations.

Chapter 4. Configuration de l'environnement de développement

Configurez l'environnement de développement dans Eclipse IDE pour l'écriture de vos plug-ins. Utilisez n'importe quel IDE Java EE de votre choix et effectuez les configurations requises mentionnées dans cette rubrique. Vous avez besoin de certains artefacts `<ASSET_PICKER_HOME>` pour terminer la configuration de l'environnement. Cette rubrique fournira des informations sur la génération et le packaging de projet à l'aide d'Apache Maven, ce qui permet de s'assurer qu'Apache Maven est installé.

Pour configurer l'environnement de développement, suivez la procédure ci-dessous :

1. Depuis l'emplacement `<ASSET_PICKER_HOME>/dev-kits/`, copiez le projet `asset-integration-starter` et insérez-le dans votre espace de travail de développement.
2. Ouvrez l'IDE Eclipse.
3. Sélectionnez **Fichier > Importer**.
   La boîte de dialogue **Sélectionner** apparaît.
4. Sélectionnez **Fichier WAR**, puis cliquez sur **Suivant**.
   La boîte de dialogue **Importation WAR** s'affiche.
5. Cliquez sur Parcourir, accédez à `<ASSET_PICKER_HOME>` et au fichier `select asset-viewer.war`.
6. Cliquez sur **Terminer**.
   La boîte de dialogue **Importation WAR : bibliothèques Web** s'affiche.
7. Cliquez sur **Terminer**.
8. Sélectionnez **Fenêtre > Afficher la vue > Autre**.
   La boîte de dialogue **Afficher la vue** apparaît.
9. Sélectionnez **Serveurs**, puis cliquez sur **Ouvrir**.

A titre d'exemple, nous allons illustrer l'utilisation d'Apache Tomcat 9.0 pour l'exécution de Content Integration. Vous pouvez utiliser n'importe quel serveur d'applications pris en charge et effectuer les configurations requises.

a. Ouvrez le fichier `conf/server.xml` à partir du répertoire d'installation d'Apache Tomcat 9.0 et ajoutez l'entrée suivante, avec les détails de base de données appropriés dans l'élément *<GlobalNamingResources>*. Veuillez remplacer *<DRIVER_CLASS_NAME>*, *<URL_TO_YOUR_PLATFORM_DATABASE>*, *<DATABASE_USERNAME>*, et *<DATABASE_PASSWORD>* avec les détails de la base de données Platform :

```
<Resource auth="Container" driverClassName="{DRIVER_CLASS_NAME}"
                           maxActive="20"
                           maxIdle="0"
                           maxWait="10000"
                           name="UnicaPlatformDS"
                           password="{DATABASE_PASSWORD}"
                           username="{DATABASE_USERNAME}"
                           type="javax.sql.DataSource"
                           url="{URL_TO_YOUR_PLATFORM_DATABASE}"/>
```

b. Ouvrez le fichier `conf/context.xml` à partir du répertoire d'installation d'Apache Tomcat 9.0 et ajoutez l'entrée suivante dans l'élément *<Context>* :

```
<ResourceLink auth="Container" global="UnicaPlatformDS"
 name="UnicaPlatformDS"
type="javax.sql.DataSource"/>
```

10. Pour ajouter Apache Tomcat 9.0 en tant que nouveau serveur dans Eclipse, procédez comme suit :

    a. Dans l'onglet **Serveurs**, cliquez sur le lien pour créer un nouveau serveur.
    La boîte de dialogue **Définition d'un nouveau serveur** s'ouvre.

    b. Sélectionnez **Serveur Tomcat 9.0** et fournissez des valeurs pour le **Nom d'hôte du serveur** et le **Nom du serveur**.

c. Cliquez sur **Suivant**.
   Le serveur a été ajouté.

d. Dans l'onglet **Serveurs** , cliquez deux fois sur l'entrée de serveur que vous venez d'ajouter.
   La boîte de dialogue **Présentation** apparaît.

e. Cliquez sur le lien **Ouvrir la configuration de lancement**.
   La boîte de dialogue **Edition des propriétés de configuration de lancement**.

f. Modifiez les configurations de lancement pour ajouter les arguments JVM suivants :

```
-DASSET_PICKER_HOME=<Point this to <ASSET_PICKER_HOME> directory>
-Dspring.profiles.active=platform-disintegrated
```

g. Cliquez sur **OK**.

11. Pour exécuter le fichier `asset-viewer.war` importé sur Apache Tomcat 9.0, cliquez avec le bouton droit de la souris sur le fichier `asset-viewer.war` et sélectionnez **Exécuter en tant que > Exécuter sur le serveur**.
    La boîte de dialogue **Exécuter sur le serveur** s'ouvre.

12. Cliquez sur **Terminer**.
    Le fichier `asset-viewer.war` commencera à s'exécuter sur Apache Tomcat. Une fois la configuration vérifiée, arrêtez le serveur et importez le projet de démarrage du développement de plug-in.

13. Pour installer le SDK Content Integration, procédez comme suit :

    a. Dans les répertoires suivants, supprimez les SDK qui sont déjà installés :
       - `<LOCAL_M2_REPOSITORY>\com\hcl\unica\integration-api\0.0.1-SNAPSHOT`

- `<LOCAL_M2_REPOSITORY>\com\hcl\unica\standard-integrations\0.0.1-SNAPSHOT`
- `<LOCAL_M2_REPOSITORY>\com\hcl\unica\asset-integration-api\0.0.1-SNAPSHOT`
- `<LOCAL_M2_REPOSITORY>\com\hcl\unica\entity-mapper-api\0.0.1-SNAPSHOT`

Sous UNIX ou Mac OS X, `<LOCAL_M2_REPOSITORY>` fait référence au répertoire `~/.m2/repository`.

Sous Microsoft Windows, `<LOCAL_M2_REPOSITORY>` fait référence au répertoire `C:\Users\{your-username}\.m2\repository`.

b. Servez-vous des commandes suivantes pour installer le SDK Content Integration dans votre référentiel Maven local. Recherchez `asset-integration-api.jar`, `integration-api.jar`, `standard-integrations.jar` et `entity-mapper-api.jar` dans le répertoire `<ASSET_PICKER_HOME>/dev-kits/sdk`.

```
mvn install:install-file -Dfile=<ASSET_PICKER_HOME>/dev-kits/sdk/asset-integration-api.jar -DgroupId=com.hcl.unica -DartifactId=asset-integration-api -Dversion=0.0.1-SNAPSHOT -Dpackaging=jar

mvn install:install-file -Dfile=<ASSET_PICKER_HOME>/dev-kits/sdk/integration-api.jar -DgroupId=com.hcl.unica -DartifactId=integration-api -Dversion=0.0.1-SNAPSHOT -Dpackaging=jar

mvn install:install-file -Dfile=<ASSET_PICKER_HOME>/dev-kits/sdk/standard-integrations.jar -DgroupId=com.hcl.unica -DartifactId=standard-integrations -Dversion=0.0.1-SNAPSHOT -Dpackaging=jar
```

```
mvn install:install-file -Dfile=<ASSET_PICKER_HOME>/dev-kits/sdk/
entity-mapper-api.jar -DgroupId=com.hcl.unica -DartifactId=entity-
mapper-api -Dversion=0.0.1-SNAPSHOT -Dpackaging=jar
```

14. Pour importer le projet de démarrage du développement de plug-in, sélectionnez **Fichier > Importer**.
    La boîte de dialogue **Sélectionner** apparaît.

15. Sélectionnez Projets Maven existants et cliquez sur **Suivant**.
    La boîte de dialogue **Projets Maven** s'affiche.

16. Cliquez sur **Parcourir** pour sélectionner le projet et cliquez sur **Terminer**.

17. Pour mettre à jour les dépendances Maven du projet `asset-integration-starter`, cliquez avec le bouton droit de la souris sur le projet `asset-integration-starter` et sélectionnez **Maven > Mettre à jour le projet**.

18. Assurez-vous que le nouveau projet importé utilise Java 8 pour compiler des sources.
    Ouvrez les propriétés du projet et procédez comme suit pour configurer le compilateur :
    a. Sélectionnez **Compilateur Java**.
    b. Si le Niveau de conformité du compilateur est non éditable, sélectionnez **Activer les paramètres spécifiques projet**.
    c. Définissez le Niveau de conformité du compilateur sur `1.8`.
    d. Cliquez sur **Appliquer et fermer**.

19. Pour vous assurer que la bibliothèque Java appropriée est configurée dans le chemin de génération, procédez comme suit :

    a. Sélectionnez **Chemin de génération Java > Bibliothèques**.

    b. Sélectionnez **Bibliothèque système JRE (J2SE-1.5)**.

    c. Cliquez sur **Retirer**.

    d. Cliquez sur **Ajouter une bibliothèque**.
    La boîte de dialogue **Ajouter une bibliothèque** s'ouvre.

e. Sélectionnez **Bibliothèque système JRE > Suivant**.
   La **Bibliothèque système JRE** s'affiche.

f. Sélectionnez une bibliothèque appropriée et cliquez sur **Terminer**.

20. Pour activer le traitement des annotations, procédez comme suit :
    a. Sélectionnez **Compilateur Java > Traitement des annotations** .
    b. Sélectionnez **Activer les paramètres propres au projet**.
    c. Sélectionner **Appliquer et fermer**.

21. Pour installer Lombok, procédez comme suit :

    a. Cliquez deux fois sur le fichier `LOCAL_M2_REPOSITORY\org\projectlombok\lombok\1.18.16\lombok-1.18.16.jar`.
       La boîte de dialogue du programme d'installation s'ouvre.

    b. Pour spécifier l'emplacement d'installation de votre IDE, cliquez sur **Spécifier l'emplacement**.

    c. Cliquez sur **Installer/Mettre à jour** pour effectuer l'installation.

    d. Après l'installation de Lombok, redémarrez l'IDE.

22. Pour modifier le nom du projet, procédez comme suit :
    a. Ouvrez le fichier `pom.xml` et modifiez ses propriétés de projet Maven.
    b. Cliquez avec le bouton droit sur le projet asset-integration-starter et sélectionnez **Refactoriser > Renommer**.

23. Dans le fichier `<ASSET_PICKER_HOME>/conf/custom-plugin-services.yml`, déclarez les services de plug-in. Vous pourrez accéder à ce fichier ultérieurement pour ajouter des déclarations lorsque vous introduirez des services pour vos plug-ins.

24. Pour ajouter un projet de plug-in à l'assemblage de déploiement du projet `asset-viewer.war`, procédez comme suit :

a. Cliquez avec le bouton droit sur le projet `asset-viewer.war` et sélectionnez **Propriétés**.
   La boîte de dialogue **Propriétés pour asset-viewer** s'affiche.

b. Sélectionnez **Deployment Assembly**.

c. Sélectionnez **Ajouter**.
   La boîte de dialogue de **Sélectionner le type de directive** s'affiche.

d. Sélectionnez **Projet** et cliquez sur **Suivant**.

e. Sélectionnez le projet de plug-in `asset-integration-starter` que vous avez importé lors des étapes précédentes, puis cliquez sur **Terminer**.

25. Si nécessaire, nettoyez les projets.

26. Effectuez la configuration adaptée à votre système dans `<ASSET_PICKER_HOME>/conf/systems.properties` (reportez-vous au fichier `sample-systems.properties` disponible dans le projet `<ASSET_PICKER_HOME>/dev-kits/asset-integration-starter`). Toutes les configurations d'intégration du système mentionnées dans le *Guide d'administration d'Unica Content Integration* sont prises en charge dans `systems.properties` à l'aide des propriétés pertinentes.

27. Lorsque vous développez votre plug-in, vérifiez-le en exécutant le projet `asset-viewer.war` sur un serveur d'applications précédemment configuré. Etant donné que le projet est déjà ajouté à l'assemblage de déploiement de `asset-viewer.war`, les modifications apportées à votre projet de plug-in seront déployées à chaque fois que vous exécuterez le projet `asset-viewer.war`.

28. Lorsque vous développez votre plug-in, en y ajoutant des services, utilisez un outil de votre choix pour atteindre les points d'extrémité REST suivants (modifiez la racine de contexte pour qu'elle corresponde à votre configuration) afin de vérifier la précision de votre implémentation :

    a. **S'assurer de l'intégration du système**

URL de noeud final	`http://localhost:8888/asset-viewer/api/AssetPicker/instances`
Méthode de requête	`GET`

b. **Vérifier le service simple-search**

URL de noeud final	`http://localhost:8888/asset-viewer/api/AssetPicker/mysystem/assets?query=mountain&page=0&size=10&types=Phto` où, • `mysystem` représente l'identificateur système choisi par l'implémentation du plug-in. • `query` contient le mot clé de recherche du contenu à rechercher. • `page & size` contient les détails de pagination, où page est le numéro de série des pages à extraire et size est le nombre total d'éléments de recherche sur une seule page. • `types` est l'une des catégories de contenu prises en charge (types) permettant de filtrer les éléments de recherche.
Méthode de requête	`GET`

Lorsque vous atteignez l'URL, assurez-vous que le JSON de réponse contient le résultat attendu. Seuls les détails de présentation sont inclus pour chaque objet de recherche. D'autres propriétés de contenu sont exclues pour des raisons de concision et de performances.

c. **Vérifier le service resource-loader**

URL de noeud final	`http://localhost:8888/asset-viewer/api/AssetPicker/mysystem/download?resource=`
	`http://repository_base_url/contents/sample_image.jpg%26resourceId=12345"`
	où
	• `mysystem` représente l'identificateur système choisi par l'implémentation du plug-in.
	• `resource` contient l'URL absolue du contenu à télécharger.
	• `resourceId` contient l'identificateur du contenu à télécharger.
	(Le plug-in peut choisir d'utiliser resource et/ou resourceId pour charger le contenu.)

Méthode de requête	`GET`

d. **Vérifier le service list-folders**

URL de noeud final	`http://localhost:8888/asset-viewer/api/AssetPicker/mysystem/folders?parentFolderId=1234` où : • `mysystem` représente l'identificateur système choisi par l'implémentation du plug-in. • `parentFolderId` contient l'identificateur du dossier parent dont les sous-dossiers immédiats sont attendus en réponse. Ce

	paramètre de requête est facultatif et n'est pas fourni lors du listage des dossiers de niveau supérieur/racine.
Méthode de requête	`GET`

e. **Vérifier le service list-contents**

URL de noeud final	`http://localhost:8888/asset-viewer/api/AssetPicker/mysystem/folders/1234/contents` où : • `mysystem` représente l'identificateur système choisi par l'implémentation du plug-in. • `1234` représente l'identificateur du dossier dont le contenu immédiat est attendu en réponse.
Méthode de requête	`GET`

Seuls les détails de présentation sont inclus pour chaque contenu répertorié par le service `list-contents`. D'autres propriétés de contenu sont exclues pour des raisons de concision et de performances.

f. **Vérifier le service get-content-details**

URL de noeud final	`http://localhost:8888/asset-viewer/api/AssetPicker/mysystem/assets/Images/1234` où : • `mysystem` représente l'identificateur système choisi par l'implémentation du plug-in. • `Images` représente l'ID de catégorie du contenu dont les détails sont attendus en réponse.

	• `1234` représente l'identificateur du contenu dont les détails sont attendus en réponse.
Méthode de requête	`GET`

La réponse JSON produite par le service `get-content-details` inclut toutes les propriétés de contenu, en plus des détails de présentation.

g. **Vérifier le service get-object-schema**

URL de noeud final	`http://localhost:8888/asset-viewer/api/AssetPicker/object-mapping/application/mysystem/object/Images/1234/schema` où : • mysystem représente l'identificateur système choisi par l'implémentation du plug-in. • Les images représentent la catégorie du contenu de référence utilisé pour la génération de schéma. • 1234 représente l'identificateur du contenu de référence utilisé pour la génération de schéma. A partir de la version 12.1.0.4, l'identificateur et la catégorie de contenu ne sont plus très pertinents, car le schéma est censé inclure des attributs pour toutes les catégories de contenu prises en charge.
Méthode de requête	`GET`

La réponse JSON doit contenir la liste aplatie de tous les attributs mappables et de leurs métadonnées.

h. **Vérifier le service list-content-categories**

URL de noeud final	`http://localhost:8888/asset-viewer/api/AssetPicker/mysystem/categories` où : • `mysystem` représente l'identificateur système choisi par l'implémentation du plug-in.
Méthode de requête	`GET`

i. **Vérifier le service get-cognitive-analysis**

URL de noeud final	`http://localhost:8888/asset-viewer/api/AssetPicker/actions/cognize?url=absolute_image_url` où : • `url` contient l'URL absolue de l'image pour laquelle extraire l'analyse cognitive
Méthode de requête	`GET`

Chapter 5. Vérification et dépannage des incidents

Pour vérifier l'intégration de bout en bout, placez le fichier `JAR`, contenant l'implémentation du plug-in, dans le chemin d'accès aux classes du serveur d'applications sur lequel Content Integration est déployé. En outre, configurez le référentiel de contenu correspondant dans le fichier `<ASSET_PICKER_HOME>/conf/systems.properties` (vous pouvez faire référence au fichier `sample-systems.properties` dans le projet `<ASSET_PICKER_HOME>/dev-kits/asset-integration-starter`).

Toutes les configurations d'intégration du système mentionnées dans le Guide d'administration d'Unica Content Integration sont prises en charge dans `systems.properties` à l'aide des propriétés pertinentes. Vous devez fournir l'argument JVM `-Dspring.profiles.active=platform-disintegrated` pour que `systems.properties` prenne effet (vous pouvez toujours utiliser les configurations de Platform au lieu de systems.properties en supprimant l'argument JVM `-Dspring.profiles.active=platform-disintegrated`).

Note: Actuellement, seuls Unica Centralized Offer Management et Unica Plan peuvent accéder à Content Integration.

Une fois le plug-in déployé et les configurations du système effectuées, redémarrez l'application Content Integration.

Bien que vous puissiez vérifier Content Integration à l'aide des nœuds finaux REST mentionnés dans la section précédente, nous vous recommandons de vérifier l'intégration de bout en bout en exécutant l'interface utilisateur appropriée dans Unica Centralized Offer Management et Unica Plan. Reportez-vous aux guides utilisateur correspondants pour savoir comment accéder aux fonctions Content Integration dans les produits respectifs.

Utilisez les outils de développement fournis par les navigateurs pris en charge pour dépanner les appels d'API, si nécessaire.

Présentation des consignateurs

Comme indiqué dans Vérification de l'intégration, la configuration de la journalisation pour Content Integration est disponible dans le fichier `log4j2.xml`, placé dans le dossier `AssetPicker/conf/logging` dans l'accueil de Platform.

Content Integration utilise Apache `Log4j2` pour la gestion des journaux. L'appender `RandomAccessFilePlatform` ainsi que le consignateur `com.unica` configuré dans `log4j2.xml` contrôlent les journaux produits par les fichiers `unica-common.jar` et `unica-helper.jar` de Platform utilisés dans Content Integration. Les paramètres restants contrôlent la journalisation pour d'autres activités principales de Content Integration.

Dans les deux cas, le niveau de journalisation par défaut est défini sur `WARN`, ce qui doit être suffisant pour répondre aux conditions de résolution des incidents pour le développement de plug-ins. La plupart des consignateurs, produits par Content Integration au niveau INFO et DEBUG, ne sont pas très pertinents pour le développement et l'intégration de plug-ins. Les rubriques suivantes ne traitent que des consignateurs pertinents. Ces consignateurs sont déjà présents dans le fichier `log4j2.xml` et leur mise en commentaire doit être annulée, si nécessaire. Veillez à ce que le niveau de journalisation ne soit jamais défini sur `DEBUG` ou `TRACE` pour ces consignateurs en production, étant donné qu'ils peuvent générer des informations sensibles.

Le fichier `log4j2.xml` contient également les configurations nécessaires pour acheminer tous les consignateurs d'un utilisateur spécifique vers un fichier journal dédié. Par défaut, ces configurations sont commentées. Une description appropriée est ajoutée dans `log4j2.xml` en haut de chaque élément de configuration pour vous aider à activer le fichier journal dédié.

Consignateurs utiles dans le fichier log4j2.xml

Le tableau suivant répertorie les consignateurs utiles dans le fichier `log4j2.xml` :

Table 2. Consignateurs utiles dans le fichier log4j2.xml

Gestionnaires de journalisation	Informations
`org.springframework.web`	La définition de ce consignateur sur le niveau TRACE produit des détails de requête et de réponse HTTP pour toutes les requêtes HTTP entrantes vers Content Integration. Ce consignateur peut être utile si vous voulez voir ce qui est échangé entre le système frontal et dorsal.
com.hcl.unica.cms.integration .flow.interceptor.logger	Ce consignateur est le plus utile pour le développement de plug-ins. Il journalise l'interaction HTTP entre Content Integration Framework et le référentiel cible. Pour tout service implémenté à l'aide de l'approche RESTful (en implémentant RestService, HTTPService ou leurs dérivés spécialisés), ce consignateur écrira les détails de requête et de réponse HTTP pour toutes les interactions HTTP sortantes avec le système cible. Pour éviter une faille de sécurité, les valeurs des en-têtes confidentiels sont masqués avant la journalisation. Seuls les quatre derniers caractères ne sont pas masqués pour la résolution des incidents. Ces en-têtes incluent l'en-tête standard Autorisation ou tout en-tête personnalisé non standard défini dans la demande ou reçu en réponse.
`org.springframework.retry`	La définition de ce consignateur sur le niveau `TRACE` permet d'ajouter des informations liées aux tentatives de relance, tout en adressant des appels HTTP au référentiel cible. Cela s'avère utile pour vérifier la stratégie de

Gestionnaires de journalisation	Informations
	relance configurée sous la section QOS pour le système respectif dans la configuration de Platform.

Autres consignateurs importants

Les autres consignateurs importants sont utiles pour résoudre les incidents liés à Asset Picker.Content Integration En plus de repérer les avertissements et les erreurs, ces consignateurs fournissent des informations utiles d'un point de vue fonctionnel.

Le tableau suivant répertorie les autres consignateurs importants :

- **Applications client** - Si le niveau du consignateur root est défini sur le niveau INFO, les lignes suivantes vous indiquent le nombre d'applications client et les applications client qu'Content Integration peut identifier :

```
SupportedClientApplications: Found {1} supported client applications.
SupportedClientApplications: Registered {Offer} as supported client
 application.
```

- **CORS** - Si le consignateur root est défini sur le niveau INFO, les lignes suivantes peuvent fournir des informations à propos de la prise en charge du partage de ressources d'origine croisée par Content Integration :

```
RegexCorsConfig: CORS: Enabling CORS for {hcl.com} & its subdomains.
 Allowed HTTP methods - {[GET, POST]}, allowed headers - {[*]}
RegexCorsConfig: CORS: Allowed origins set to {[http(s)?://([^\.]+
\.)*hcl.com(:[0-9]+)?]}
```

- **Configuration de Platform - Référentiels de contenu** - La définition du niveau du consignateur root sur INFO nous donne des informations sur les référentiels de contenu identifiés par Content Integration Framework.

```
PlatformConfigurationCategoryResolver: Platform configuration: Reading
 list of entries for path {Affinium|Offer|partitions|partition1|Content
 Integration|dataSources}...
```

```
PlatformCmsConfigurationReader: Platform configuration: Imported
 settings for {AEM#119[partition1]}
PlatformCmsConfigurationReader: Platform configuration: Imported
 settings for {WCM#119[partition1]}
PlatformCmsConfigurationReader: Platform configuration: Imported
 settings for {Bing#119[partition1]}
```

- **Fichiers de méta-informations de service** - Les lignes suivantes sont également consignées au niveau INFO pour indiquer le nombre de fichiers de méta-informations identifiés par Content Integration Framework :

```
c.h.u.s.c.s.PluginServicesYamlConfigReader: Scanning & parsing service
 configuration files.
c.h.u.s.c.s.PluginServicesYamlConfigReader: Seeking file at
 {<ASSET_PICKER_HOME>\conf\plugin-services.yml}.
c.h.u.s.c.s.PluginServicesYamlConfigReader: Found service config file
 at {<ASSET_PICKER_HOME>/conf/plugin-services.yml}
c.h.u.s.c.s.PluginServicesYamlConfigReader: Parsing service
 configuration file (YAML): {<ASSET_PICKER_HOME>/conf/plugin-
services.yml}...
c.h.u.s.c.s.PluginServicesYamlConfigReader: Seeking file at
 {<ASSET_PICKER_HOME>\conf\custom-plugin-services.yml}.
c.h.u.s.c.s.PluginServicesYamlConfigReader: {1} service declaration(s)
 found for {COM} - {[COM:get-object-schema]}
c.h.u.s.c.s.PluginServicesYamlConfigReader: {12} service declaration(s)
 found for {WCM} - {[WCM:item-details, WCM:simple-search, WCM:content-
list, WCM:logon-service, WCM:list-contents, WCM:library-list, WCM:get-
content-details, WCM:folder-list, WCM:get-object-schema, WCM:list-
folders, WCM:library-by-id, WCM:resource-loader]}
c.h.u.s.c.s.PluginServicesYamlConfigReader: {31} service declaration(s)
 found for {Deliver} - {[Deliver:update-folder, Deliver:simple-
search, Deliver:list-by-ids, Deliver:zip-file-upload, Deliver:delete-
content, Deliver:move-folder, Deliver:create-content, Deliver:list-
```

```
folders, Deliver:zip-upload-template-unknown, Deliver:move-
content, Deliver:list-sub-folders, Deliver:download-content-
variant, Deliver:download-file-attachment, Deliver:get-user-
entitlements, Deliver:list-top-folders, Deliver:update-dynamic-
content, Deliver:create-folder, Deliver:find-libraries-by-name,
 Deliver:resource-loader, Deliver:zip-upload-content, Deliver:adopt-
dynamic-content, Deliver:get-folder, Deliver:create-dynamic-content,
 Deliver:list-contents, Deliver:get-content-details, Deliver:patch-
content, Deliver:delete-folder, Deliver:get-library, Deliver:update-
content, Deliver:get-library-file, Deliver:adopt-content]}
c.h.u.s.c.s.PluginServicesYamlConfigReader: {1} service declaration(s)
 found for {Azure} - {[Azure:get-cognitive-analysis]}
c.h.u.s.c.s.PluginServicesYamlConfigReader: {1} service declaration(s)
 found for {DX-CORE} - {[DX-CORE:logon-service]}
c.h.u.s.c.s.PluginServicesYamlConfigReader: {7} service declaration(s)
 found for {DX} - {[DX:simple-search, DX:list-contents, DX:get-content-
details, DX:rendition-details, DX:get-object-schema, DX:list-folders,
 DX:resource-loader]}
c.h.u.s.c.s.PluginServicesYamlConfigReader: {7} service declaration(s)
 found for {Commerce} - {[Commerce:simple-search, Commerce:list-
contents, Commerce:get-content-details, Commerce:get-search-query-
suggestions, Commerce:list-content-categories, Commerce:get-object-
schema, Commerce:list-folders]}
c.h.u.s.c.s.PluginServicesYamlConfigReader: {7} service declaration(s)
 found for {AEM} - {[AEM:simple-search, AEM:list-contents, AEM:get-
content-details, AEM:get-object-schema, AEM:get-content-fragment-model,
 AEM:list-folders, AEM:sample-inbound-service]}
c.h.u.s.c.s.PluginServicesYamlConfigReader: {2} service declaration(s)
 found for {Bing} - {[Bing:simple-search, Bing:get-content-details]}
```

- **Protocoles d'authentification** - Les lignes suivantes, consignées au niveau INFO, confirment que le protocole d'authentification est identifié pour le référentiel de contenu donné :

```
AssetPickerRestTemplate: Setting up {BASIC} authentication for
 {Offer[partition1].WCM:simple-search} service...
```

- **Invalidation du cache de configuration de Platform et réinitialisations de service** - Toutes les configurations de Platform pour Content Integration sont mises en cache lors du démarrage de l'application. Ces configurations sont actualisées après un intervalle donné (toutes les 30 minutes par défaut, sauf si un autre intervalle a été configuré). Le consignateur suivant est produit au niveau INFO, à chaque fois que l'actualisation de la configuration commence :

  ```
  INFO  [scheduling-1] c.h.u.s.c.s.ServiceBootstrapper: Re-initializing
   services...
  ```

  De même, les lignes suivantes sont générées au niveau INFO chaque fois que la configuration se termine :

  ```
  INFO  [scheduling-1] c.h.u.s.c.s.ServiceBootstrapper: Finished service
   initializations.
  INFO  [scheduling-1] c.h.u.s.c.s.ServiceBootstrapper: Re-initialization
   completed in 3692 milliseconds. YAML read time: 15 milliseconds,
   DB Read Time: 3608 milliseconds, Service initialization time: 68
   milliseconds
  ```