Unica Asset Picker version 12.0 - Guide du développeur

Contents

Chapitre 1. Developer Guide

Ce guide fournit des informations sur le développement de plug-ins et la résolution d'incidents liés à Unica Asset Picker.

Présentation

Asset Picker simplifie l'intégration au système de gestion de contenu et permet d'y rechercher du contenu.

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

Plug-in

Asset Picker s'intègre à différents CMS à l'aide d'API REST. Il relève le défi lié à la disparité de l'interface de programmation entre les différents systèmes en exploitant les modules ou les plug-ins personnalisés créés spécifiquement pour le système cible.

Vous pouvez implémenter des plug-ins à l'aide du langage de programmation Java. Asset Picker 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 Asset Picker 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

Asset Picker 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 Asset Picker. 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

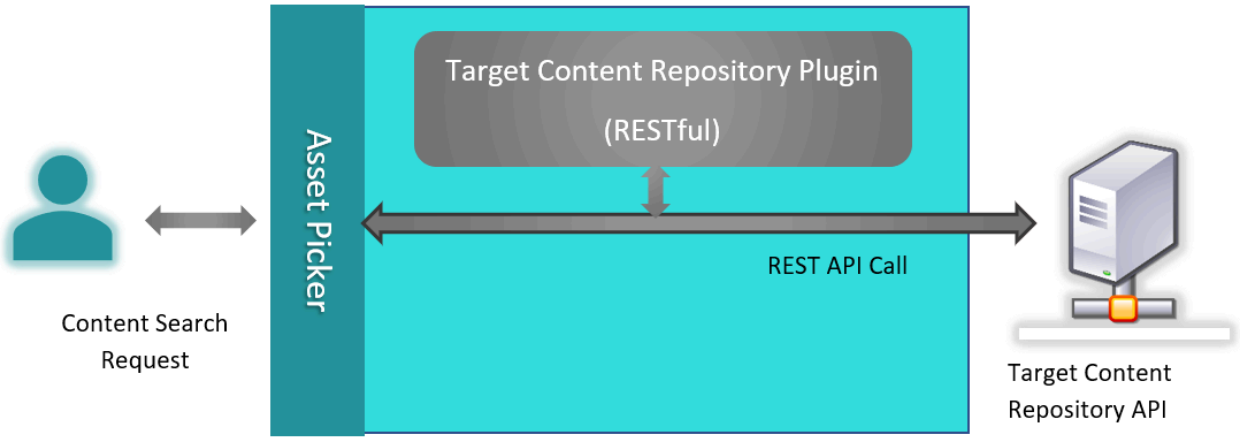
Une autre approche de développement de plug-in pour une intégration non RESTful implique une implémentation complète. Par exemple, un plug-in écrit pour récupérer le contenu de la base de données doit traiter tout ce qui contribue à l'établissement d'une connexion à la base de données, l'exécution de 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. Asset Picker reçoit d'abord la requête de recherche, qui est déléguée au plug-in respectif. En cas d'intégrations RESTful, Asset Picker 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

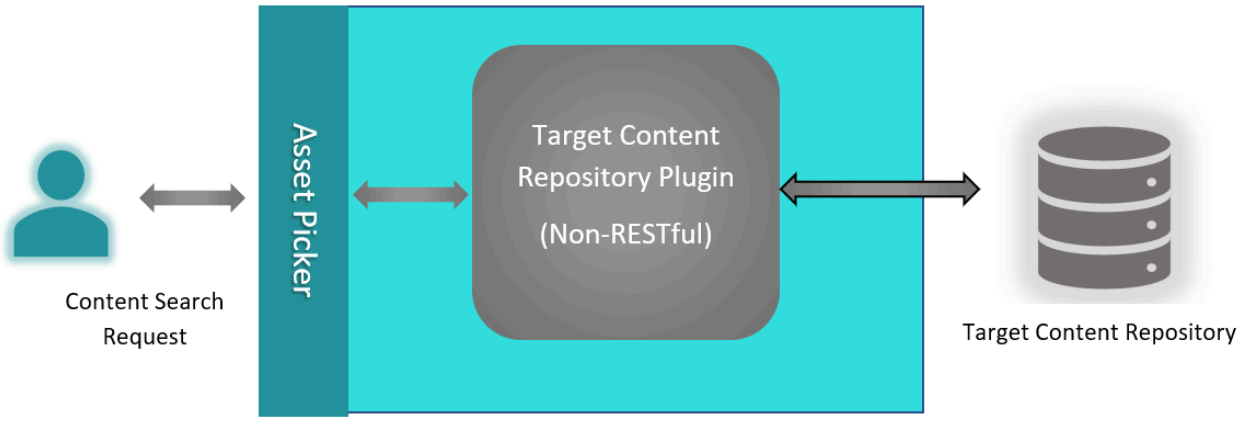


Lorsqu'Asset Picker reçoit une demande de recherche de contenu de l'utilisateur pour le système cible, il consulte le plug-in respectif pour recueillir des informations logiques spécifiques à la requête et adresse un appel d'API au système cible. Il consulte à nouveau le plug-in pour transformer la réponse 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 Asset Picker. Contrairement aux référentiels RESTful, Asset Picker ne connaîtra pas le type, l'architecture, le protocole et le mécanisme d'authentification utilisés pour communiquer avec le référentiel cible.

Présentation du développement de plug-ins

Asset Picker simplifie l'intégration aux nouveaux référentiels de contenu sans devoir modifier la structure principale d'Asset Picker.

Asset Picker 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 aux classes du serveur d'applications hébergeant Asset Picker, le système 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 Asset Picker - Guide d'administration.

Asset Picker 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 `AssetPicker/dev-kits` dans l'accueil de Platform. Deux projets de référence, nommés `aem-integration` et `wcm-integration`, sont respectivement disponibles pour Adobe Experience Manager (AEM) et IBM Web Content Manager (WCM). Pour créer un plug-in pour un nouveau système, nous vous recommandons d'utiliser le projet de démarrage afin d'enregistrer le code d'écriture passe-partout.

Composants de plug-in

Un plug-in standard contient les composants suivants :

- Fichier de méta-informations du service *(à la page 5)*
- Implémentations de service *(à la page 11)*

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.

Le fichier de méta-informations de service est un fichier de configuration YML contenant la liste des services inclus dans le plug-in. Un service peut être standard ou personnalisé.

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

Fichier de méta-informations du service

Voici les prérequis pour le fichier de méta-informations :

- Il est prévu que le fichier de méta-informations du service se trouve dans le répertoire `META-INF` sur le chemin de classe du projet.
- Le nom du fichier de méta-informations doit se terminer par le suffixe `content-services.yml`. Exemples :
  - `wcm-content-services.yml`
  - `aem-content-services.yml`
  - `example-content-services.yml`

Les fichiers de référence se trouvent dans les projets `aem-integration`, `wcm-integration` et `asset-integration-starter` aux emplacements répertoriés suivants :

- `dev-kits\aem-integration\src\main\resources\META-INF`
- `dev-kits\wcm-integration\src\main\resources\META-INF`
- `dev-kits\asset-integration-starter\src\main\resources\META-INF`

Voici l'exemple de contenu d'un fichier du projet actif-intégration-démarreur :

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


  -
    systemId: Foo
    serviceName: resource-loader
    factoryClass: com.example.service.rest.ResourceLoaderService
```

```
-
  systemId: Foo
  serviceName: asset-selection-callback
  factoryClass: com.example.service.rest.ContentSelectionCallbackService


-
  systemId: Foo
  serviceName: custom-service
  factoryClass: com.example.service.rest.CustomService
```

Déclarations de service

Le document de méta-informations commence par la clé des services, qui est un tableau de dictionnaires contenant trois éléments nommés systemId, serviceName et factoryClass. 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. Des points, des tirets et des traits de soulignement peuvent être utilisés pour plus de lisibilité. 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
  ```

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. Asset Picker n'impose aucune restriction. De même, les entrées d'un plug-in peuvent être divisées en plusieurs fichiers de méta-informations tant que les noms de fichiers se terminent par le suffixe `content-services.yml`.

- `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`
  - `asset-selection-callback`

- `factoryClass`

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

Services standard

Le tableau ci-dessous donne une présentation des services standard d'Asset Picker :

Tableau 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 Asset Picker. 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.
`resource-loader`	Ce service est utile lorsqu'un accès indirect à l'élément de recherche est requis. Ce service n'est pas obligatoire et ne doit être mis en œuvre que lorsque les complications suivantes se présentent : 1. La valeur de retour du service `simple-search` comprend une URL (relative ou absolue) vers l'élément de recherche respectif afin que le client Asset Picker puisse charger son contenu sur le Web. En l'absence de lien Web direct vers l'élément de recherche, composez une

Nom du service standard	Description
	URL qui concerne l'élément de recherche, qui reviendra vers Asset Picker chaque fois que le client doit charger le contenu de cet élément. Cette URL doit contenir un identificateur unique pour l'élément de recherche, qui est utilisé par le service `resource-loader` pour lire le contenu de la ressource. Par exemple, si le référentiel cible est une base de données, le service `simple-search` récupère alors les enregistrements de la base de données qui correspond à la requête de recherche donnée. 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 `simple-search` compose une URL relative à la racine de contexte Asset Picker qui inclut un identificateur de l'élément. Les demandes ultérieures de chargement d'un élément de recherche individuel passeront par Asset Picker, qui déléguera la tâche de chargement des ressources au service `resource-loader`. En outre, le service de chargement de ressources identifie la ressource ou l'élément, en fonction de l'identificateur que le service `simple-search` avait fourni dans l'URL. Cette URL permet au service `resource-loader`

Nom du service standard	Description
	de lire le contenu des éléments de recherche et de répondre. 2. Si le système cible ne fournit pas d'accès anonyme aux éléments de recherche, le service `resource-loader` est utilisé comme une disposition de fortune pour récupérer le contenu des éléments individuels. Le service `resource-loader` s'authentifie avant de récupérer le contenu des ressources à partir du référentiel cible. Si le service `resource-loader` est mis en œuvre à l'aide de l'approche RESTful, Asset Picker se chargera de l'authentification prête à l'emploi, sous réserve des configurations dans Unica Platform. Pour plus d'informations sur les configurations de l'intégration de référentiel, voir *Unica Asset Picker - Guide d'administration*. **Remarque :** En production, l'acheminement des demandes de chargement de ressources à l'aide d'Asset Picker n'est recommandé dans aucun des cas. L'utilisation du service `resource-loader` doit uniquement se limiter aux démonstrations ou à la phase de développement. Le contenu devrait être accessible de manière anonyme dans le système cible en vue d'une utilisation de contenu transparente.

Nom du service standard	Description
`asset-selection-callback`	Le service est utile lorsque le plug-in effectue une action en réponse à la sélection d'un élément de recherche par le client. Lorsque le service `simple-search` renvoie les résultats de la recherche au client, l'un des éléments de recherche est choisi par le client pour d'autres cas d'utilisation. Le plug-in peut vérifier quel élément a été choisi par le client pour effectuer l'action correspondante (par exemple, verrouiller cet élément dans le référentiel sous-jacent afin qu'il ne puisse pas être mis à jour ou supprimé, etc.). Ce service n'est pas obligatoire et ne doit être mis en œuvre que si une opération de rappel est nécessaire.

Implémentations de service

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

Asset Picker fournit un SDK pour rationaliser l'implémentation du service et facilite le développement rapide de plug-ins. Le SDK Asset Picker 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 Asset Picker. Pour des détails supplémentaires, voir Développement de plug-ins SDK (à la page 16).

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 Asset Picker 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 de l'entrée donnée au service en cours d'implémentation.

- **Chaîne getEndpointUrl(ExecutionContext<RQ> executionContext)**

  Cette méthode renvoie une URL de nœud final absolue du service exécuté sur le système cible. L'URL de base du système cible est configurée dans Unica Platform. Par conséquent, le plug-in ne doit prendre aucune disposition pour la configurer. L'objet `ExecutionContext` fourni à cette méthode propose un moyen de lire l'URL de base afin de pouvoir composer l'URL absolue du service. En outre, regardez la manière dont la méthode `getEndpointUrl` est définie dans la classe `com.aem.service.AemSimpleSearchService` à l'intérieur du projet `aem-integration`. Comme on peut le constater, l'URL de base est obtenue à partir de `ExecutionContext` en naviguant jusqu'à l'objet `InstanceConfig`. Le `InstanceConfig` contient toutes les configurations effectuées dans Unica Platform pour l'instance du système cible avec laquelle votre service va communiquer. Il s'agit d'une méthode obligatoire pour le service à implémenter.
- **HttpMethod getHttpMethod()**

  Cette méthode doit renvoyer l'une des valeurs de l'énumération `HttpMethod` fournie avec le SDK Asset Picker. Comme son nom l'indique, cette méthode indique la méthode

de requête HTTP à utiliser lors de l'interaction HTTP avec le système cible. Il s'agit d'une méthode obligatoire pour le service à implémenter.

- **Map<String, Object> getHeaders(ExecutionContext<RQ> executionContext)**

  Cette méthode facultative peut être remplacée par le service s'il souhaite inclure des en-têtes de requête HTTP à l'appel HTTP sortant. La valeur de retour doit être une instance de mappe, dans laquelle les noms d'en-tête HTTP 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 implémentation, aucun en-tête personnalisé ne sera envoyé avec la requête HTTP sortante.

  Remarque : Bien que la mappe renvoyée par cette méthode 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'Asset Picker. 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.
  ```

  - L'en-tête HTTP `Content-Type` doit être renseigné en tant que clé `contentType` en raison de considérations spéciales dans la structure sous-jacente.
  - `application/json` est la valeur contentType par défaut pour les services RESTful, si la méthode `getHeaders` ne fournit aucune valeur.

- **Object buildRequest(ExecutionContext<RQ> executionContext)**

  Il s'agit également d'une méthode facultative. Si le service cible attend un corps de la requête, cette méthode peut être remplacée pour créer le corps de la requête HTTP souhaité. Le type de retour de cette méthode est Objet. Dès lors, donc tout type de corps de la requête valide peut être fourni tant que l'en-tête `Content-Type` pertinent est renseigné à l'aide de la méthode `getHeaders`.

  Remarque : Prise en charge de Jackson et JAXB - La sérialisation d'objets à l'aide de Jackson et de JAXB est entièrement prise en charge par Asset Picker. Ainsi, un objet correctement décoré avec des annotations Jackson ou JAXB peut être renvoyé à partir de cette méthode. Dans ce cas, aucun en-tête `Content-Type` ne doit être renseigné explicitement. Asset Picker se charge de fournir l'en-tête approprié lors de l'appel HTTP.

La sérialisation de l'objet fourni dans le corps de la requête est également gérée par Asset Picker lui-même. Aucune sérialisation explicite n'est donc requise.

En l'absence de cette implémentation, un corps de la requête vide sera envoyé avec la requête HTTP sortante.

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

  Cette méthode facultative transforme la réponse HTTP au format souhaité. Le premier argument supplémentaire de cette méthode est le corps de la réponse HTTP reçu du service cible. Cet argument est un type générique et est décidé en fonction du paramètre de type réel utilisé lors de l'implémentation du service. Cette réponse peut être n'importe quel objet, une chaîne contenant le texte tel que reçu du service, un tableau d'octets comprenant le contenu de la réponse ou un objet désérialisé représentant le JSON/XML de la réponse.

  Remarque : Prise en charge de Jackson et JAXB - La désérialisation d'objets à l'aide de Jackson et de JAXB est entièrement prise en charge par Asset Picker. 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 Asset Picker. 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 Asset Picker.

  Hormis ces méthodes, il existe une autre méthode dont le `getServiceInterface` a hérité de `com.hcl.unica.cms.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.

  Asset Picker 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 - Asset Picker 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 Asset Picker. 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 de l'entrée donnée au service en cours d'implémentation.

- **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.

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 Asset Picker à cet égard
    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 Picker
        - L'intégration de la logique pour tous les supports manquants prêts à l'emploi peut rendre le service fonctionnel très verbeux

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

Développement de plug-ins SDK

Le développement de plug-ins SDK fournit des informations sur les différentes classes, interfaces et énumérations du SDK Asset Picker, à l'aide d'unités logiques correspondantes

dans des projets de référence `asset-integration-starter`, `aem-integration` et `wcm-integration` qui sont intégrées dans le cadre du kit de développement avec l'application Asset Picker.

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 (à la page 11).

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, enveloppée par notre service, prend le contenu demandé et produit la réponse souhaitée. Il nécessite certaines notations génériques pour les entrées et sorties échangées pendant le flux logique de bout en bout.

Asset Picker utilise RQ pour désigner certaines entrées au service et RS pour indiquer la sortie du service ou la réponse de l'API REST distante. La définition de RS peut changer selon son lieu d'utilisation.

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 String getEndpointUrl(ExecutionContext<ServiceInput>
 executionContext) {
ServiceInput input = executionContext.getRequest();
// Remaining implementation omitted for brevity
}
```

- **RS**

  Ce paramètre de type 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 l'objet du type `ApiResponse` est communiqué en tant que premier argument, qui correspond au paramètre de type RS de l'interface `RestService`.

  Remarque : 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 `FunctionalInterface` 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. Il s'agit de la seule méthode supplémentaire qu'un service RESTful doit implémenter et elle renvoie l'objet de classe du dérivé (interface enfant) de `ServiceGateway`. La définition de `com.hcl.unica.cms.integration.service.gateway.ServiceGateway` est la suivante :

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

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.

Remarque : 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.

Invocation de service

Le projet actif-intégration-démarreur contient une classe com.example.service.client.CustomServiceClient qui permet d'illustrer l'appel du service.

La méthode `invocationDemo` dans cette classe obtient la référence à `custom-service` en utilisant la méthode statique `getServiceGateway` depuis la classe `ServiceGatewayFactory`. La méthode `getServiceGateway` prend trois arguments pour renvoyer l'instance de service. Les arguments sont les suivants :

- `String systemId`

  Cet identificateur de système est le même que celui utilisé dans le fichier de méta-informations du service pour déclarer le service dont l'instance doit être obtenue.
- `String serviceName`

  Il s'agit du nom du service dont l'instance doit être obtenue. Il doit être identique à celui déclaré dans le fichier de méta-informations du service.
- `Class<T> gatewayClass`

  Il doit s'agir de l'objet Classe de l'interface ServiceGateway (ou de son interface enfant). Il doit correspondre à la valeur de retour de la méthode getServiceInterface dans l'implémentation de service correspondante.

La méthode invocationDemo dans la classe com.example.service.client.CustomServiceClient utilise CustomServiceGateway (gatewayClass) pour obtenir l'instance du service de service personnalisé (serviceName) pour le système Foo (systemId). Le fragment de code suivant vous est fourni à titre de référence :

```
public void invocationDemo() {
  String systemId = "Foo";

  CustomServiceGateway customService =
ServiceGatewayFactory.getServiceGateway(
    systemId,
    "custom-service",
    CustomServiceGateway.class
   );
```

```
 ServiceInput input = new ServiceInput();
 ServiceOutput output = customService.execute(input);
}
```

Le type de retour de getServiceGateway est aussi CustomServiceGateway. L'instance de service obtenue peut servir à exécuter le service respectif en fournissant l'objet d'entrée requis.

Remarque :

- Le type de retour de `getServiceGateway` est aussi `CustomServiceGateway`. L'instance de service obtenue peut servir à exécuter le service respectif en fournissant l'objet d'entrée requis. La méthode d'exécution est utilisée sur l'interface `ServiceGateway` pour exécuter le service. 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 le `com.example.service.functional.CustomService class`. Le type de sortie est le même que celui utilisé pour définir l'interface `CustomServiceGateway` dont l'objet Classe est renvoyé par la méthode `getServiceInterface` dans les deux versions de classe `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 méta-informations du service dans le projet `asset-integration-starter` utilisant le `META-INF/rest-content-services.yml` et le `META-INF/functional-content-services.yml` ont une entrée pour `custom-service` qui désigne les versions respectives de la `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 Asset Picker. 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

Dans le cas d'une application client à plusieurs partitions d'Asset Picker, la méthode d'obtention de l'instance de service mentionnée précédemment renverra de manière appropriée la partition de référence spécifique à la passerelle de service. L'objet `ExecutionContext` transmis à diverses méthodes de rappel contiendra les informations spécifiques à la partition nécessaires à son utilisation.

Contexte d'exécution

Presque toutes les méthodes du contrat d'implémentation de service reçoivent une instance de classe `com.hcl.unica.cms.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 d'exécution sur l'interface de `ServiceGateway`. (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 des attributs supplémentaires relatifs à l'exécution du service en cours, tels que l'état de réponse HTTP et les en-têtes pour l'appel HTTP en cours.

  Remarque : L'en-tête HTTP `Content-Type` est renseigné en tant que clé `contentType` en raison de considérations spéciales dans la structure sous-jacente.
- **ServiceConfig getServiceConfig()**

  Cette méthode renvoie une instance de la classe `com.hcl.unica.cms.integration.config.ServiceConfig`. Cet objet contient les configurations effectuées dans le fichier de méta-informations de service pour le service respectif.
- **InstanceConfig getInstanceConfig()**

Cette méthode renvoie une instance de la classe `com.hcl.unica.cms.integration.config.InstanceConfig`. Cet objet contient toutes les configurations effectuées dans Unica Platform pour le système cible (l'instance de ce nom de méthode fait référence à l'instance du système cible et non à l'instance de service). Dans le cas de configurations à partitions multiples, cet objet sera correctement rempli par Asset Picker pour contenir la configuration spécifique à la partition. Pour connaître les différents paramètres de configuration d'instance dans Unica Platform, voir Unica Asset Picker - Guide d'administration.

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

  L'utilisation de cette méthode est strictement limitée à Asset Picker. Evitez d'utiliser cette méthode avec les plug-ins.

Sources de données utilisateur

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

```
executionContext.getInstanceConfig().getDataSourceCredentials()
```

L'objet `DataSourceCredentials` renvoyé par la méthode `etDataSourceCredentials` 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 prendront aucune décision logique concernant la sélection adéquate de la source de données utilisateur.

De même, la méthode `getUnicaToken` appelée sur l'objet `InstanceConfig` 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.

En outre, il est possible d'exécuter le service implémenté en obtenant une référence à celui-ci à l'aide de la méthode `ServiceGatewayFactory.getServiceGateway`. Asset Picker tire parti de cette conception et définit certains services standard. Il s'agit des services

Recherche simple (`simple-search`), Chargeur de ressource (`resource-loader`) et Rappel de sélection d'actif (`asset-selection-callback`). Asset Picker propose des interfaces spécialisées étendues à partir de `RestService` et de `FunctionalService` pour chacun de ces services standard afin de faciliter leur mise en œuvre à l'aide d'une approche RESTful ou fonctionnelle.

Appel de services standard

Une fois déclarés dans le fichier de méta-informations de service et implémentés à l'aide d'une approche RESTful ou fonctionnelle, Asset Picker les appelle dans les scénarios suivants :

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

  Chaque fois que Asset Picker 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. Asset Picker 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 méta-informations 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.
- **Chargeur de ressource (`resource-loader`)**

  Le service `resource-loader` est exécuté par Asset Picker uniquement lorsqu'un accès indirect (ou authentifié) doit être établi au niveau de l'élément de recherche sur le système cible. En l'absence d'une URL directe vers un élément de recherche individuel sur le système cible, le service `resource-loader` peut être implémenté par le plug-in. En tant que service facultatif, Asset Picker reconnaît automatiquement si l'implémentation de `resource-loader` existe pour le système cible et l'appelle en conséquence.
- **Rappel de sélection d'actif (`asset-selection-callback`)**

Le sélecteur d'actifs exécute ce service pour le système cible lorsque l'application client sélectionne l'un des éléments de recherche renvoyés par le service `simple-search`. Etant donné que le service est facultatif, Asset Picker peut automatiquement reconnaître l'existence de ce service et l'appeler en conséquence.

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 *(à la page 25)*
- Dérivés du service Http *(à la page 29)*
- Dérivés de service fonctionnel *(à la page 31)*

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.cms.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.cms.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 Asset Picker.

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 Asset Picker 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 `RestSearchService` afin d'être reconnu en tant que service `simple-search` par Asset Picker.

`RestSearchService` s'étend à partir de la classe abstraite `com.hcl.unica.cms.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.

- `com.hcl.unica.cms.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.cms.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`. Chaque implémentation `simple-search` doit remplacer et implémenter cette nouvelle méthode. 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 nouvelle méthode est la suivante :

```
abstract public List<String> getSupportedContentTypes();
```

L'implémentation de cette méthode renvoie une liste de chaînes représentant les catégories de contenu ou d'actifs à rechercher dans le système cible. Aucune sémantique spécifique n'est associée aux valeurs de cette liste. Il peut être n'importe quel texte significatif. Elle fait office de filtre pour l'application client pendant l'opération de recherche. L'application client peut envoyer des valeurs à partir de cette liste pour filtrer les éléments de recherche. Il est possible d'obtenir les valeurs reçues par l'application client depuis l'objet `ExecutionContext` en naviguant jusqu'à la méthode `getRequest` et en y appelant `getTypes()`. `getRequest()` renvoie l'objet `SearchRequest` qui contient l'ensemble des types pris en charge que l'application cliente a envoyés pour filtrer le résultat de la recherche. L'implémentation de recherche simple traite ces ensembles de valeurs conformément à l'interface de programmation du système cible et filtre les éléments de recherche en conséquence. Examinez la méthode `getSupportedContentTypes` dans la classe `com.aem.service.AemSimpleSearchService` du projet `aem-integration` et la manière dont la méthode `restrictContentTypes` dans la classe `com.aem.service.simplesearch.SimpleSearchRequestBuilder` limite le résultat de la recherche aux types sélectionnés.

Rappel de sélection d'actif (`asset-selection callback`)

Les interfaces et classes spécialisées disponibles pour le service `asset-selection-callback` sont les suivantes :

- `com.hcl.unica.cms.integration.service.assetselectioncallback.RestAssetSelectionCallbackService`

La classe `com.example.service.rest.ContentSelectionCallbackService` du projet `asset-integration-starter` est une mise en œuvre rapide pour le service `asset-selection-callback` RESTful. Son parent est la classe suivante :

```
com.hcl.unica.cms.integration.service.assetselectioncallback
.RestAssetSelectionCallbackService
```

La classe `RestAssetSelectionCallbackService` 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 Chaîne définie dans la bibliothèque Java standard.

La classe `RestAssetSelectionCallbackService` implémente l'interface `RestService` et définit la classe `com.hcl.unica.cms.model.request.assetselectioncallback.AssetSelectionDetails` pour qu'elle soit l'argument type RQ pour RestService. Dès lors, l'objet de `AssetSelectionDetails` devient une entrée pour tous les services `asset-selection-callback` (la même entrée est également utilisée pour la contrepartie fonctionnelle d'actif-sélection-rappel). La classe `AssetSelectionDetails` fait partie du SDK Asset Picker. La classe `AssetSelectionDetails` encapsule les détails d'un actif (élément de recherche) sélectionné par l'application client et les informations contextuelles, telles que la requête de recherche qui mène au résultat de recherche contenant l'élément sélectionné.

Le plug-in doit étendre son implémentation `asset-selection-callback` depuis le service `RestAssetSelectionCallbackService` afin d'être reconnu en tant que service `asset-selection-callback` par Asset Picker (la contrepartie fonctionnelle est également un choix valide pour effectuer l'extension).

`RestAssetSelectionCallbackService` s'étend depuis la classe abstraite suivante :

```
com.hcl.unica.cms.integration.service.assetselectioncallback
.AbstractAssetSelectionCallbackService
```

- ```
  com.hcl.unica.cms.integration.service.assetselectioncallback
  .AbstractAssetSelectionCallbackService
  ```
  ```

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

La classe suivante définit l'interface comme la passerelle de service pour le service `asset-selection-callback` :

```
com.hcl.unica.cms.integration.service.gateway
.AssetSelectionCallbackServiceGateway
```

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 `asset-selection-callback` prend l'objet `AssetSelectionDetails` et renvoie n'importe quel objet. Actuellement, la valeur de retour de `asset-selection-callback` est ignorée par Asset Picker.

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.cms.integration.service.resourceloader.DefaultWebResourceLoaderService`
  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.cms.integration.service.resourceloader
  .DefaultWebResourceLoaderService
  ```

La classe `DefaultWebResourceLoaderService` est l'implémentation par défaut du service `resource-loader` fourni par le SDK Asset Picker. Si le plug-in n'implémente pas son propre service `resource-loader`, Asset Picker revient à cette implémentation par défaut. L'implémentation par défaut de `resource-loader` fournie par le SDK Asset Picker 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`.

- `com.hcl.unica.cms.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.cms.model.request.resourceloader.ResourceRequest` et octet[], respectivement. La classe `ResourceRequest` encapsule l'URL de ressource relative et l'identificateur d'instance (L'identificateur d'instance est exactement le même que le `systemId` utilisé partout dans Asset Picker). 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 depuis la classe `HttpWebResourceLoaderService` afin d'être reconnu en tant que service `resource-loader` par Asset Picker. (La contrepartie fonctionnelle est également un choix valide pour effectuer l'extension.)

- `com.hcl.unica.cms.integration.service.resourceloader.AbstractWebResourceLoaderService`

  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.cms.integration.service.gateway
```

```
.ResourceLoaderServiceGateway
```

Pour connaître le rôle des passerelles de service dans l'appel de service, voir Invocation de service (à la page 19). L'interface `ResourceLoaderServiceGateway` définit `ResourceRequest` et `WebResource<?>` en tant que types d'entrée et de sortie pour le service de chargeur de ressource. La classe `com.hcl.unica.cms.model.response.resourceloader.WebResource` est simplement un encapsuleur pour les en-têtes de réponse HTTP, le corps et les cookies reçus de l'URL distante.

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.cms.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. Son parent est la classe `com.hcl.unica.cms.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 service `SearchService` afin d'être reconnu en tant que service `simple-search` par Asset Picker (la contrepartie RESTful est également un choix valide pour effectuer l'extension).

Le `SearchService` s'étend à partir de la classe `com.hcl.unica.cms.integration.service.search .AbstractSearchService abstract`. Il introduit une méthode plus abstraite, intitulée `getSupportedContentTypes` pour implémenter le service `simple-search`.

Rappel de sélection d'actif (`asset-selection-callback`)

Les interfaces et classes spécialisées disponibles pour le service `asset-selection-callback` sont les suivantes :

- `com.hcl.unica.cms.integration.service.assetselectioncallback.AssetSelectionCallbackService`
  La classe `com.example.service.functional.ContentSelectionCallbackService` du projet `asset-integration-starter` est une mise en œuvre rapide pour le service `asset-selection-callback` fonctionnel. Son parent est la classe suivante :

  ```
  com.hcl.unica.cms.integration.service.assetselectioncallback
  .AssetSelectionCallbackService
  ```

  La classe `AssetSelectionCallbackService` implémente l'interface `FunctionalService` et définit la classe `AssetSelectionDetails` et les classes `Object` pour qu'elles soient les arguments types RQ et RS respectivement pour le `FunctionalService`. Dès lors, l'objet de `AssetSelectionDetails` devient une entrée pour tous les services `asset-selection-callback` et le `Object` 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 d'actif-sélection-rappel). La classe `AssetSelectionDetails` fait partie du SDK Asset Picker.

  Le plug-in doit étendre son implémentation `asset-selection-callback` depuis le service `AssetSelectionCallbackService` afin d'être reconnu en tant que service `asset-selection-callback` par Asset Picker (la contrepartie RESTful est également un choix valide pour effectuer l'extension).

  Le `AssetSelectionCallbackService` s'étend depuis la classe abstraite suivante :

  ```
  com.hcl.unica.cms.integration.service.assetselectioncallback
  ```

```
.AbstractAssetSelectionCallbackService
```

Chargeur de ressource (`resource-loader`)

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

- `com.hcl.unica.cms.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. Son parent est la classe `com.hcl.unica.cms.integration.service.resourceloader.WebResourceLoaderService`.

  La classe `WebResourceLoaderService` implémente l'interface `FunctionalService` et définit les classes `ResourceRequest` et `WebResource` pour qu'elles soient les arguments types RQ et RS respectivement pour le `FunctionalService`. Dès lors, l'objet de `ResourceRequest` devient une entrée pour tous les services `resource-loader` et le `WebResource` 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`).

  Le plug-in doit étendre son implémentation `resource-loader` depuis le service `WebResourceLoaderService` pour être reconnu en tant que service `resource-loader` par Asset Picker (la contrepartie RESTful est également un choix valide pour effectuer l'extension).

  Le `WebResourceLoaderService` s'étend depuis la classe abstraite suivante :

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

Exceptions standard

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

Approche RESTful

Asset Picker gère les conditions d'erreur résultant des services mis en œuvre à l'aide de l'approche RESTful.

En outre, Asset Picker 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'Asset Picker 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 référentiel de contenu cible et sont présentes au sein du package com.hcl.unica.cms.integration.exception.

- **RepositoryNotFoundException**

  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 également 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 également transmise en tant que réponse HTTP 404 au client.

- **InaccessibleRepositoryException**

  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 également transmise en tant que réponse HTTP 503 au client.

- **SluggishRepositoryException**

  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 également transmise en tant que réponse HTTP 504 au client.

- **InternalRepositoryErrorException**

  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 également 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.

Gestionnaires de journalisation

Asset Picker 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.

Vérification et dépannage des incidents

Vous devez vérifier l'intégration de bout en bout après le développement du plug-in. Placez le fichier JAR, contenant l'implémentation du plug-in, dans le chemin d'accès aux classes du serveur d'applications sur lequel le sélecteur d'actifs est déployé. En outre, configurez le référentiel de contenu correspondant dans la configuration de Platform sous l'application prise en charge.

Remarque : Actuellement, seul Unica Centralized Offer Management peut accéder à Asset Picker.

Pour plus d'informations sur les détails de configuration, voir Unica Asset Picker - Guide d'administration.

Une fois le plug-in déployé et les configurations personnalisées, redémarrez l'application Asset Picker. Les modifications apportées à la source de données utilisateur ne nécessitent pas de redémarrage.

Vérification de l'intégration

Bien que vous puissiez vérifier Asset Picker à l'aide de nœuds finaux REST, nous vous recommandons de vérifier l'intégration de bout en bout en exécutant l'interface utilisateur appropriée dans Unica.

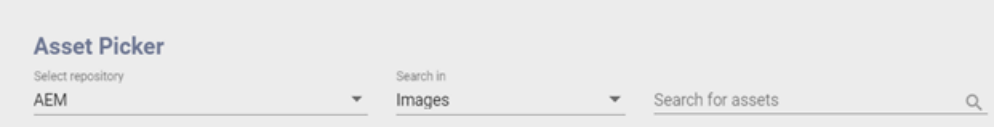
Dans cette édition, l'attribut personnalisé, pour une URL d'actif dans les offres, permet de travailler avec le sélecteur d'actifs. En accédant à l'écran correspondant dans la gestion des offres, vous pouvez lancer Asset Picker pour un attribut personnalisé de sélecteur d'URL. Une fois la fenêtre contextuelle Asset Picker lancée, utilisez les instructions suivantes pour vérifier les différents services inclus dans votre plug-in :

- Vérifier enregistrement du système
- Vérifier le service `simple-search`
- Vérifier le service `resource-loader`
- Vérifier le service `asset-selection-callback`

Vérifier enregistrement du système

Votre référentiel de contenu doit être répertorié dans la première liste déroulante, comme indiqué dans l'image suivante. De plus, tous les types de contenu pris en charge doivent apparaître dans la liste déroulante suivante.

Figure 3. Vérification de l'enregistrement du système



Utilisez les outils de développement de votre navigateur pour dépanner la réponse reçue du système dorsal d'Asset Picker. L'accès à l'URL du nœud final suivante se fait lorsqu'Asset Picker se lance pour récupérer la liste des référentiels de contenu disponibles pour l'utilisateur connecté :

`/api/AssetPicker/instances`

Si votre référentiel de contenu n'est pas répertorié, assurez-vous que l'identificateur système utilisé dans le plug-in et l'identificateur système utilisé dans la configuration de la plateforme correspondent. En outre, vous pouvez vous référer aux journaux d'application pour vérifier toute erreur ou exception possible.

Vérifier le service de recherche simple

Après avoir effectué une recherche valide sur votre référentiel de contenu, le résultat de recherche attendu avec les détails pertinents apparaît.

Utilisez les outils de développement sur votre navigateur pour dépanner la réponse reçue du système dorsal d'Asset Picker. L'accès à l'URL de nœud final ci-dessous se fera lors de l'exécution de l'opération de recherche :

`/api/AssetPicker/WCM/assets?query=none&types=Images&page=0&size=10 10`

Dans l'URL mentionnée précédemment :

- `WCM` est l'identificateur système de votre référentiel de contenu.
- `query` contient le mot clé de la recherche.
- `types` contient la liste des types de contenu pris en charge pour filtrer le résultat de la recherche.
- `page` correspond au nombre de pages du résultat de la recherche.
- `size` est le nombre maximum d'éléments attendus sur une seule page.

Vérifier le service de chargeur de ressource

Dans la configuration de Platform, si **Contenu anonyme** est configuré sur `Non` pour votre référentiel de contenu, les éléments de recherche seront accessibles via Asset Picker au

lieu de se connecter directement au référentiel cible. Pour garantir l'exactitude du service `resource-loader`, vérifiez l'accessibilité ou la visibilité de chaque élément de recherche.

Utilisez les outils de développement de votre navigateur pour dépanner la réponse reçue du système dorsal d'Asset Picker. L'accès à l'URL sert à récupérer un élément de recherche individuel :

```
/api/AssetPicker/WCM/?resource /wps/wcm/connect/9350fd83-cd83-465a-a847-967f59048c0c/unnamed.jpg?MOD=AJPERES&CVID=n2B8-11
```

Dans l'URL précédente, WCM est l'identificateur système de votre référentiel de contenu et la ressource contient l'URL relative à l'élément de recherche correspondant. Cette URL est similaire à celle que le service de recherche simple renseigne dans l'attribut d'URL de ressource de l'actif correspondant, lors de la transformation du résultat de la recherche.

Dans la configuration de Platform, si **Contenu anonyme** est configuré sur `Oui` pour votre référentiel de contenu, tous les éléments de recherche seront récupérés directement à partir de votre référentiel de contenu, au lien de passer par Asset Picker. Dans ce cas, le service `resource-loader` ne sera pas appelé.

Vérifier le service système actif-sélection-rappel

L'édition 12.0 d'Asset Picker ne prend pas en charge le service `asset-selection-callback`.

Présentation des consignateurs

Comme indiqué dans Vérification de l'intégration (à la page 37), la configuration de la journalisation pour Asset Picker est disponible dans les fichiers `log4j.xml` et `log4j2.xml`, placés dans le dossier `AssetPicker/conf/logging` dans l'accueil de Platform.

Le fichier `log4j.xml` est utilisé pour les consignateurs provenant de `unica_common.jar` et de `unica_helper.jar` de Platform. Par contre, `log4j2.xml` est utilisé pour les consignateurs provenant d'une autre partie d'Asset Picker.

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 Asset Picker 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.

Consignateurs utiles dans le fichier log4j2.xml

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

Tableau 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 Asset Picker. 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 Asset Picker 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

Gestionnaires de journalisation	Informations
	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 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. 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'Asset Picker 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 Asset Picker :

```
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 Asset Picker.

```
PlatformConfigurationCategoryResolver: Platform configuration: Reading
 list of entries for path {Affinium|Offer|partitions|partition1|
assetPicker|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 Asset Picker :

```
YamlConfigReader: 2 service configuration file(s) found.
YamlConfigReader: Parsing service configuration file (YAML):
 {jar:file:/{DEPLOYMEN_LOCATION}/asset-viewer/WEB-INF/lib/aem-
integration-0.0.1-SNAPSHOT.jar!/META-INF/aem-content-services.yml}...
YamlConfigReader: Parsing service configuration file (YAML):
 {jar:file:/{DEPLOYMEN_LOCATION}/asset-viewer/WEB-INF/lib/wcm-
integration-0.0.1-SNAPSHOT.jar!/META-INF/wcm-content-services.yml}...
```

- **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...
```