Unica Asset Picker V12.0
Entwicklerhandbuch

Contents

Kapitel 1. Entwicklerhandbuch

Dieses Handbuch bietet Informationen zur Plug-in-Entwicklung und Fehlerbehebung bei Unica Asset Picker.

Überblick

Asset Picker erleichtert die einfache Integration mit Content Management Systemen und ermöglicht die Suche nach Inhalten aus den Content Management Systemen.

Der abgerufene Inhalt kann vom Kunden von Asset Picker für verschiedene inhaltsorientierte, geschäftliche Anwendungsfälle verwendet werden. Ein Asset Picker-Client ist ein beliebiges Produkt aus der Unica-Suite, das sich mit Asset Picker integriert, um die Inhalte von Zielsystemen zu konsumieren.

Plug-ins

Asset Picker lässt sich über REST-APIs in verschiedene CMS integrieren. Es stellt sich der Herausforderung, die Schnittstellenunterschiede zwischen verschiedenen Systemen zu programmieren, indem es die benutzerdefinierten Plug-ins oder Module nutzt, die speziell für das Zielsystem geschrieben wurden.

Sie können Plug-ins mit der Programmiersprache Java implementieren. Asset Picker erzwingt für die Entwicklung solcher Plug-ins keine Abhängigkeit von einer Drittanbieter-Bibliothek. Sie können die Plug-ins so anpassen, dass sie jede beliebige Bibliothek von Drittanbietern für ihre Implementierung nutzen. Mit Plug-ins können die logischen Lücken in Bezug auf das Zielsystem gefüllt werden.

Plug-ins ergänzen auf nicht-intrusive Weise Asset Picker, um gewünschte Inhalte aus externen Inhaltsspeichern abzurufen.

Integrationsunterstützung und Ansatz zur Plug-in-Entwicklung

Asset Picker bietet gebrauchsfertige Unterstützung für die einfache Integration mit RESTful-Schnittstellen. Es erleichtert auch alternative Ansätze der Plug-in-Entwicklung zur

Integration mit Nicht-RESTful-Systemen wie Datenbanken, Dateisystemen oder anderen Inhaltsrepositorys.

Ein typisches Plug-in, das für die REST-API-Integration geschrieben wurde, enthält keine Logik zur Herstellung einer Verbindung mit dem Zielsystem und zur Behandlung von Erfolgs- und Fehlerbedingungen auf Protokollebene. Diese Aufgaben werden vom Asset Picker ausgeführt. Plug-ins stellen nur systemspezifische Informationen zur Verfügung wie z. B:

- absolute Position der Ziel-API
- zu verwendende HTTP-Methode
- zu liefernde Header
- zu sendender Anfragekörper
- Typ der zu erwartenden Antwort
- Transformator für die empfangene Antwort

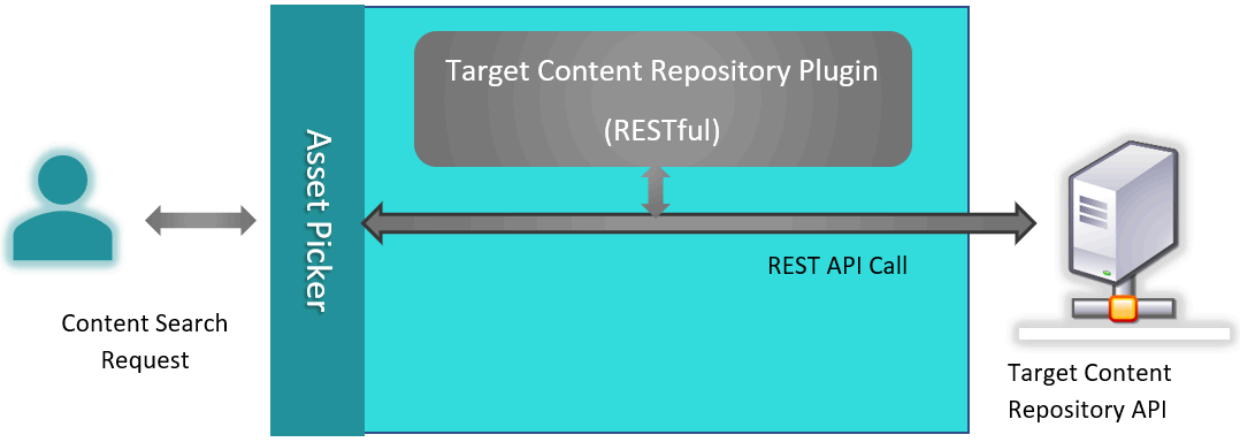
Ein alternativer Ansatz zur Entwicklung von Plug-ins für eine Nicht-RESTful-Integration erfordert eine gründliche Implementierung. Zum Beispiel muss ein Plug-in, das für das Abrufen von Inhalten aus der Datenbank geschrieben wurde, alles angehen, was mit der Herstellung der DB-Verbindung, der Ausführung von SQLs, dem Schließen von Verbindungen, der Hydratisierung der Ergebnismenge, der Fehlerbehandlung usw. zu tun hat.

Plug-ins leiten die Inhaltssuche nicht ein. Asset Picker empfängt zunächst die Suchanfrage, die an das jeweilige Plug-in delegiert wird. Im Falle von RESTful-Integrationen leitet Asset Picker die HTTP-Interaktion ein und sammelt bei Bedarf die notwendigen Informationen aus dem Plug-In.

Suchablauf für RESTful-Inhalte

Die folgende Abbildung zeigt den End-zu-End-Ausführungsablauf für die RESTful-Inhaltssuche:

Abbildung 1. Suchablauf für RESTful-Inhalte

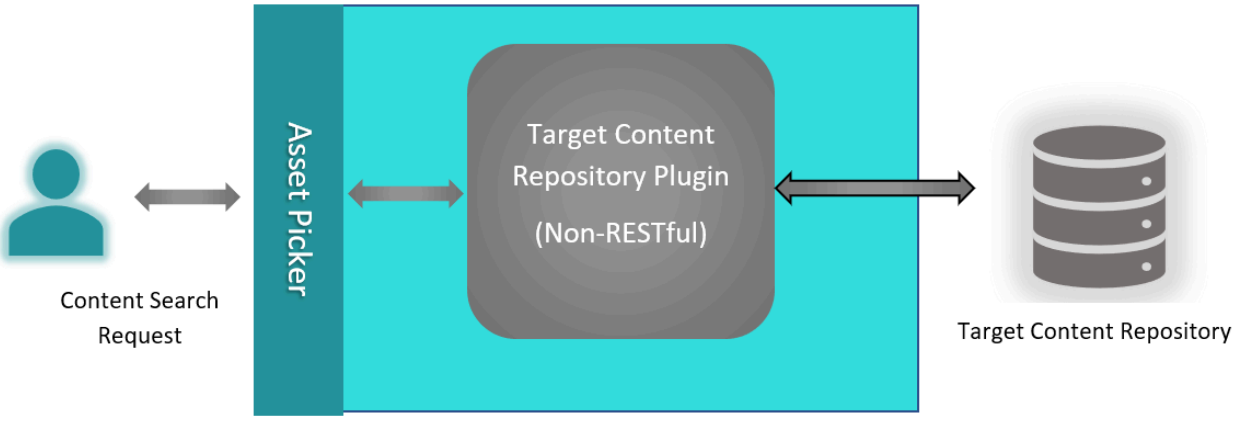


Wenn Asset Picker eine Inhaltssuchanforderung vom Benutzer für das Zielsystem erhält, ruft er das entsprechende Plug-in auf, um anforderungsspezifische logische Informationen zu erfassen, und führt einen API-Aufruf an das Zielsystem aus. Es ruft noch einmal das Plug-in auf, um die API-Antwort in ein erwartetes Format umzuwandeln, und antwortet dem Benutzer.

Suchablauf für Nicht-RESTful-Inhalte

Die folgende Abbildung zeigt den End-zu-End-Ausführungsablauf für die Nicht-RESTful-Inhaltssuche:

Abbildung 2. Suchablauf für Nicht-RESTful-Inhalte



Nicht-RESTful-Plug-ins interagieren mit dem Inhaltsrepository und liefern die Suchergebnisse an Asset Picker. Im Gegensatz zu RESTful-Repositorys kennt Asset Picker

nicht den Typ, die Architektur, das Protokoll und den Authentifizierungsmechanismus, der für die Kommunikation mit dem Zielrepository verwendet wird.

Überblick über die Plug-in-Entwicklung

Asset Picker erleichtert die einfache Integration mit neuen Inhaltsrepositorys, ohne dass das zentrale Asset Picker-Framework geändert werden muss.

Asset Picker lässt sich nahtlos mit systemspezifischen, unabhängigen Plug-ins integrieren. Sobald das Plug-in entwickelt und in den Klassenpfad des Anwendungsservers, auf dem Asset Picker gehostet wird, aufgenommen wurde, lässt sich das entsprechende System in die Unica-Produktsuite integrieren, indem einige Konfigurationen in Unica Platform aktualisiert werden. Weitere Informationen finden Sie im Administratorhandbuch von Unica Asset Picker

Asset Picker wird mit einem Entwicklungskit bereitgestellt, das die Abhängigkeiten, Referenzprojekte und ein Starterprojekt zum Schnellstart der Plug-in-Entwicklung enthält. Das Entwicklungskit befindet sich im `AssetPicker/dev-kits`-Verzeichnis innerhalb des Platform-Ausgangsverzeichnisses. Zwei Referenzprojekte namens `aem-integration` und `wcm-integration` stehen für Adobe Experience Manager (AEM) bzw. IBM Web Content Manager (WCM) zur Verfügung. Um ein Plug-in für ein neues System zu schreiben, empfehlen wir Ihnen, das Starterprojekt zu verwenden, um sich das Schreiben von Boilerplate-Code zu ersparen.

Komponenten eines Plug-ins

Ein typisches Plug-in enthält die folgenden Komponenten:

- Metainformationsdatei des Service *(auf Seite 5)*
- Serviceimplementierungen *(auf Seite 11)*

Der Begriff Service steht für eine Java-Klasse, die entweder indirekt beim Gebrauch eines externen REST-Services hilft oder direkt mit externen Web-Services oder Systemen für einen bestimmten Zweck interagiert. Das externe System muss kein Standard Content

Management System sein und die externen Services müssen nicht zu einem Standard-CMS gehören. Es kann sich um ein beliebiges System oder eine API handeln.

Die Service-Metainformationsdatei ist eine YML-Konfigurationsdatei, die die Liste der im Plug-in enthaltenen Services umfasst. Ein Service kann entweder ein Standardservice oder ein benutzerdefinierter Service sein.

Standardservices tragen eine besondere Semantik und einen besonderen Zweck in Asset Picker. Die Implementierung bestimmter Standardservices ist für Asset Picker obligatorisch, um mit dem Inhaltsrepository arbeiten zu können.

Metainformationsdatei des Service

Im Folgenden sind die Voraussetzungen für eine Metainformationsdatei aufgeführt:

- Service-Metainformationsdatei wird innerhalb des `META-INF`-Verzeichnisses auf dem Projektklassenpfad erwartet.
- Der Name der Metainformationsdatei muss mit dem Suffix `content-services.yml` enden. Beispiele:
  - `wcm-content-services.yml`
  - `aem-content-services.yml`
  - `example-content-services.yml`

Referenzdateien finden Sie innerhalb von `aem-integration`, `wcm-integration`, und `asset-integration-starter`-Projekten unter den nachfolgend aufgeführten Positionen:

- `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`

Im Folgenden finden Sie den Beispielinhalt einer Datei aus dem Asset-Integration-Starter-Projekt:

```
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
```

Servicedeklarationen

Das Metainformationsdokument beginnt mit dem Serviceschlüssel, einer Reihe von Verzeichnissen, die drei Elemente namens „systemId“, „serviceName“ und „factoryClass“ enthalten. Einzelheiten zu den Elementen sind nachstehend aufgeführt:

- `systemId`

  Dieser Zeichenkettenwert identifiziert ein Ziel-Inhaltsrepository eindeutig. Dieser Bezeichner sollte vorzugsweise nur alphanumerische Zeichen enthalten. Punkte, Striche und Unterstriche können zur Verbesserung der Lesbarkeit verwendet werden. Der einmal für das Zielsystem gewählte Bezeichner muss in allen Servicedeklarationen für dasselbe System konsistent bleiben. Dieser Bezeichner wird auch in der Konfiguration von Unica Platform für das Onboarding des jeweiligen Systems verwendet.

Im Folgenden finden Sie einige Beispiele für gültige Systembezeichner:

```
WCM
AEM
Example
WCM_1.0
AEM_1_1
```

Sie können verschiedene Plug-ins für verschiedene Versionen desselben Systems schreiben. In einem solchen Fall müssen unterschiedliche Bezeichner verwendet werden, um jede Version eindeutig zu identifizieren. Alternativ kann dasselbe Plug-in verschiedene Versionen von Serviceimplementierungen enthalten, die für verschiedene Versionen des entsprechenden Systems spezifisch sind. In diesem Fall müssen den jeweiligen Servicedeklarationen sorgfältig unterschiedliche System-Ids zugeordnet werden. Beispielsweise können zwei verschiedene Versionen von WCM, nämlich 1.0 und 2.0, unterschiedliche APIs für den Inhaltssuchservice enthalten, was zu folgenden Serviceeinträgen für die jeweiligen Versionen führt:

```
-
  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
```

Die beiden Einträge können zum selben Plug-in gehören oder aus Gründen der Übersichtlichkeit der Implementierung in zwei verschiedenen Plug-ins platziert werden. Asset Picker erlegt keine Einschränkungen auf. Ebenso können Einträge für ein Plug-in in mehrere Metainformationsdateien aufgeteilt werden, solange die Dateinamen mit dem Suffix `content-services.yml` enden.

- `serviceName`

Dieser Zeichenkettenwert identifiziert den gegebenen Service für das entsprechende System eindeutig. Es kann entweder ein Name für den Standardservice oder ein entsprechend gewählter Name für den benutzerdefinierten Service sein. Im Folgenden finden Sie die Liste der Standard-Servicenamen:

- `simple-search`
- `resource-loader`
- `asset-selection-callback`

- `factoryClass`

  Dies ist ein voll qualifizierter Pfad zur Implementierung von Services in der Java-Klasse.

Standardservices

Die folgende Tabelle enthält eine Einführung in die Standardservices von Asset Picker:

Tabelle 1. Standardservices und deren Beschreibung

Name des Standardservice	Beschreibung (Description)
`simple-search`	Der Service „Einfache Suche" antwortet auf die Suchanfragen nach Inhalten, die von Asset Picker empfangen werden. Dieser Service akzeptiert die Zeichenfolge der Suchanfrage zusammen mit den erforderlichen Angaben zur Paginierung der Ergebnisse. Basierend auf dem Erfolg des Suchvorgangs liefert es das Suchergebnis für die gegebene Suchanfrage und entsprechend der gewünschten Paginierung. Für das Plug-in handelt es sich hierbei um einen obligatorischen Service.
`resource-loader`	Dieser Service ist immer dann nützlich, wenn ein indirekter Zugriff auf den Suchbegriff erforderlich ist. Dieser Service ist nicht obligatorisch und sollte nur implementiert werden, wenn folgende Hindernisse auftreten:

Name des Standardservice	Beschreibung (Description)
	1. Der Rückgabewert des `simple-search`-Service enthält eine URL (entweder relativ oder absolut) zum jeweiligen Suchbegriff, sodass der Asset Picker-Client seine Inhalte über das Web laden kann. Wenn es keinen direkten Web-Link zu dem Suchbegriff gibt, verfassen Sie eine URL, die den Suchbegriff betrifft und die zum Asset Picker zurückkommt, wenn der Inhalt dieses Begriffs vom Client geladen werden muss. Eine solche URL sollte einen eindeutigen Bezeichner für den Suchbegriff enthalten, der vom `resource-loader`-Service zum Lesen des Ressourceninhalts verwendet wird. Wenn das Zielrepository beispielsweise eine Datenbank ist, dann ruft der `simple-search`-Service Datensätze aus der Datenbank ab, die der gegebenen Suchanfrage entsprechen. Da die Elemente aus der Datenbank geladen werden, gibt es möglicherweise keine URL, die direkt auf jeden Datensatz verweist. In solchen Fällen verfasst der `simple-search`-Service eine URL relativ zum Kontextstamm von Asset Picker, die einen Bezeichner des Elements enthält. Spätere Anfragen zum Laden einzelner Suchbegriffe durchlaufen den Asset Picker, der die Aufgabe des Ressourcenladens an den `resource-`

Name des Standardservice	Beschreibung (Description)
	`loader`-Service überträgt. Zusätzlich identifiziert der Service Ressourcenlader die Ressource oder das Element auf der Grundlage der Kennung, den der `simple-search`-Service in der URL geliefert hatte. Diese URL ermöglicht es dem `resource-loader`-Service, den Inhalt von Suchbegriffen zu lesen und darauf zu antworten. 2. Wenn das Zielsystem keinen anonymen Zugriff auf Suchbegriffe bietet, wird der `resource-loader`-Service als behelfsmäßige Bereitstellung verwendet, um den Inhalt einzelner Begriffe abzurufen. Der `resource-loader`-Service authentifiziert sich selbst, bevor er die Ressourceninhalte aus dem Zielrepository abruft. Wenn der `resource-loader`-Service mit dem RESTful-Ansatz implementiert wird, kümmert sich Asset Picker vorbehaltlich der Konfigurationen in Unica Platform um die gebrauchsfertige Authentifizierung. (Weitere Informationen zu den Onboarding-Konfigurationen von Repositorys finden Sie im *Administratorhandbuch von Unica Asset Picker*). **Anmerkung:** In der Produktion ist es nicht empfehlenswert, Anfragen zum Laden von Ressourcen mit dem Asset Picker auf

Name des Standardservice	Beschreibung (Description)
	die eine oder andere Weise weiterzuleiten. Die Nutzung von `resource-loader`-Service sollte nur auf Demonstrationen oder die Entwicklungsphase beschränkt werden. Es wird erwartet, dass die Inhalte im Zielsystem anonym zugänglich sind, um eine nahtlose Nutzung der Inhalte zu ermöglichen.
`asset-selection-callback`	Der Service ist nützlich, wenn das Plug-in als Reaktion auf die Auswahl eines beliebigen Suchbegriffs durch den Client eine Aktion ausführt. Wenn der `simple-search`-Service Suchergebnisse an den Client zurückgibt, wird eines der Suchbegriffe vom Client für weitere Anwendungszwecke ausgewählt. Das Plug-in könnte prüfen, welches Objekt vom Client ausgewählt wurde, um die entsprechende Aktion auszuführen (z. B. Sperren des Objekts im zugrunde liegenden Repository, sodass es nicht aktualisiert oder gelöscht werden kann usw.). Dieser Service ist nicht obligatorisch und sollte nur implementiert werden, wenn ein Rückruf erforderlich ist.

Serviceimplementierungen

Für jeden Service, der in der Servicemetainformationsdatei angegeben ist, muss eine Implementierung innerhalb der jeweiligen factoryClass vorhanden sein.

Der Asset Picker bietet ein SDK zur Rationalisierung der Serviceimplementierung und erleichtert die schnelle Entwicklung von Plug-ins. Das Asset Picker SDK ermöglicht zwei verschiedene Ansätze für Serviceimplementierungen: „RESTful“ und „Functional“.

In diesem Abschnitt wird eine kurze Einführung in diese Ansätze gegeben. Weitere Informationen finden Sie im `asset-integration-starter`-Projekt.

In diesem Thema werden auch bestimmte Typen, Schnittstellen, ihre generischen Typparameter und Aufzählungen aus dem Asset Picker SDK vorgestellt. Weitere Informationen finden Sie im Abschnitt SDK Plug-in-Entwicklung (auf Seite 17).

RESTful-Ansatz

Die `com.example.service.rest.CustomService`-Klasse hilft Ihnen, die REST-basierte Serviceimplementierung zu verstehen.

Diese Klasse ist eine Implementierung der `RestService`-Schnittstelle und stellt somit einen REST-basierten Service dar. Da REST vollständig auf HTTP-Standards basiert, wird die `RestService`-Schnittstelle im Asset Picker SDK von der `HttpService`-Schnittstelle abgeleitet und als Markerschnittstelle definiert. Die `RestService`-Schnittstelle gibt keine zusätzliche Methode für sich selbst an. Im Folgenden sind die in der `HttpService`-Schnittstelle angegebenen Methoden aufgeführt, die eine REST-basierte Serviceimplementierung implementieren muss. Nicht alle Methoden sind obligatorisch. Alle Methoden akzeptieren ein `ExecutionContext`-Objekt, das alle Kontextinformationen enthält, die jede Methode benötigt, um die ihr zugewiesene Aufgabe zu erfüllen. Der generische Typparameter für die `ExecutionContext`-Klasse stellt den Typ des Inputs dar, der dem zu implementierenden Service zugewiesen wird.

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

  Diese Methode gibt eine absolute Endpunkt-URL des auf dem Zielsystem ausgeführten Service zurück. Die Basis-URL des Zielsystems wird in Unica Platform konfiguriert. Daher braucht das Plug-in keine Vorkehrungen zu treffen, um dies in irgendeiner Weise zu konfigurieren. Das dieser Methode zur Verfügung gestellte `ExecutionContext`-Objekt bietet eine Möglichkeit, die Basis-URL zu lesen, sodass die absolute URL des Services verfasst werden kann. Schauen Sie sich auch an, wie die `getEndpointUrl`-Methode in der `com.aem.service.AemSimpleSearchService`-Klasse innerhalb des `aem-integration`-Projekts definiert ist. Wie zu bemerken ist, wird die Basis-URL von `ExecutionContext` durch Navigation durch `InstanceConfig` Objekte bezogen. Das `InstanceConfig` enthält alle in Unica Platform vorgenommenen Konfigurationen für

genau die Zielsysteminstanz, mit der Ihr Service kommunizieren wird. Dies ist eine für den zu implementierenden Service obligatorische Methode.

- **HttpMethod getHttpMethod()**

  Diese Methode sollte einen der Werte aus dem mit dem Asset Picker SDK gelieferten `HttpMethod` enum zurückgeben. Wie der Name schon sagt, sagt diese Methode aus, welche HTTP-Anforderungsmethode während der HTTP-Interaktion mit dem Zielsystem verwendet werden sollte. Dies ist eine obligatorische Methode für den zu implementierenden Service.

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

  Diese optionale Methode kann vom Service überschrieben werden, wenn er beliebige HTTP-Anfrageheader in den ausgehenden HTTP-Aufruf aufnehmen möchte. Rückgabewert muss eine Map-Instanz sein, wobei die HTTP-Headernamen in Form von Map-Schlüsseln angegeben werden müssen und die Headerwerte als entsprechende Werte in der Map geliefert werden müssen. In Ermangelung dieser Implementierung werden keine benutzerdefinierten Header zusammen mit der ausgehenden HTTP-Anfrage gesendet.

  Anmerkung: Obwohl die von dieser Methode zurückgegebene Map die Werte des Typs Objekt (oder seiner Untertypen) akzeptiert, werden ab der aktuellen Implementierung von Asset Picker nur Zeichenkettenobjekte unterstützt. Jeder andere Werttyp wird ignoriert, und die folgende Warnmeldung wird protokolliert:

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

  - `Content-Type` HTTP-Header muss aufgrund besonderer Überlegungen im zugrunde liegenden Framework als `contentType`-Schlüssel aufgefüllt werden.
  - `application/json` ist der Standard-Inhaltstyp für RESTful-Services, wenn keiner von der `getHeaders`-Methode geliefert wird.

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

  Dies ist ebenfalls eine optionale Methode. Wenn der Zielservice irgendeinen Anfragekörper erwartet, dann kann diese Methode überschrieben werden, um den gewünschten HTTP-Anfragekörper zu erstellen. Der Rückgabetyp dieser Methode ist

„Objekt“, und daher kann jede Art von gültigem Anfragekörper geliefert werden, solange der relevante `Content-Type`-Header mit der `getHeaders`-Methode gefüllt wird.

Anmerkung: Jackson- und JAXB-Unterstützung - Die Objektserialisierung mit Jackson und JAXB wird von Asset Picker vollständig unterstützt. So können von dieser Methode mit Jackson- oder JAXB-Annotationen entsprechend dekorierte Objekte zurückgegeben werden. In diesem Fall braucht kein `Content-Type`-Header explizit ausgefüllt zu werden. Asset Picker sorgt für die Bereitstellung des entsprechenden Headers während des HTTP-Aufrufs. Die Serialisierung des gelieferten Objekts in den Anfragekörper wird ebenfalls von Asset Picker selbst vorgenommen, daher ist keine explizite Serialisierung erforderlich.

In Ermangelung dieser Implementierung wird ein leerer Anfragekörper zusammen mit der ausgehenden HTTP-Anfrage gesendet.

- **Objekt transformResponse(RS-Antwort, ExecutionContext<RQ> executionContext)**

  Diese optionale Methode wandelt die HTTP-Antwort in ein gewünschtes Format um. Das zusätzliche, erste Argument zu dieser Methode ist der HTTP-Antwortkörper, der vom Zieldienst empfangen wird. Dieses Argument ist ein generischer Typ und wird auf der Grundlage des bei der Implementierung des Service verwendeten tatsächlichen Typparameters entschieden. Diese Antwort kann ein beliebiges Objekt sein, entweder eine Zeichenkette, die den vom Service empfangenen Text enthält, ein Bytearray, das den Inhalt der Antwort enthält, oder ein deserialisiertes Objekt, das die Antwort JSON/XML darstellt.

  Anmerkung: Jackson- und JAXB-Unterstützung - Die Objektdeserialisierung mit Jackson und JAXB wird von Asset Picker vollständig unterstützt. Daher kann ein entsprechend dekoriertes Objekt mit Jackson- oder JAXB-Annotationen als Argument für diese Methode akzeptiert werden. Die Deserialisierung des Antwortkörpers in einen bestimmten Typ wird von Asset Picker durchgeführt, daher ist bei der Antwortumwandlung innerhalb dieser Methode keine explizite Deserialisierung erforderlich.

  In Ermangelung dieser Implementierung wird von Asset Picker keine implizite Transformation durchgeführt.

Zusätzlich zu diesen Methoden gibt es noch eine weitere Methode, die `getServiceInterface` von `com.hcl.unica.cms.integration.service.AbstractService interface` übernommen hat und die vom Service implementiert werden muss. Ihre Implementierung ist jedoch eher für den Serviceaufruf als für die Implementierung des Service relevant.

Asset Picker kümmert sich um die reale HTTP-Interaktion mit dem Zielsystem und konsultiert einfach das Serviceobjekt, um die oben genannten Details zu erhalten.

Fehlerbehandlung - Fehler oder Ausnahmen, die während eines HTTP-Aufrufs empfangen werden, werden von Asset Picker behandelt. Die oben aufgeführten Methoden dürfen keine überprüfte Ausnahme auslösen. Nicht überprüfte Ausnahmen können bei Bedarf ausgelöst werden.

Funktionaler Ansatz

Für ein besseres Verständnis der Implementierung des funktionalen Service siehe com.example.service.functional.CustomService-Klasse.

Diese Klasse ist eine Implementierung der Funktionsservice-Schnittstelle. Im Gegensatz zu einem REST-basierten Service gibt es bei dieser Art von Serviceimplementierung keine HTTP-spezifischen Rückrufmethoden. Eigentlich muss der funktionale Service nicht unbedingt mit einem HTTP-Aufruf verknüpft sein. Diese Art von Service kann jede Operation einschließen, für die es keine vorkonfigurierte Unterstützung von Asset Picker gibt. Er kann mit der Datenbank kommunizieren, den Webservice eines Dritten aufrufen, das Dateisystem bedienen usw.

Implementieren Sie die folgende Methode für einen Funktionsservice. Diese Methode akzeptiert auch ein Argument vom Typ ExecutionContext, das die für die Ausführung der gewünschten Aufgabe erforderlichen Kontextinformationen enthält. Der generische Typparameter der ExecutionContext-Klasse stellt den Typ des Inputs dar, der dem zu implementierenden Service zugewiesen wird.

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

Diese Methode erfüllt die ihr zugewiesene Aufgabe unter Verwendung der ihr übermittelten Kontextinformationen. Im Gegenzug gibt sie nach Beendigung ihrer Operation den gewünschten Wert an. Der in dieser Signatur angezeigte Rückgabewert ist ein generischer Typ und basiert auf dem bei der Implementierung der Funktionsservice-Schnittstelle verwendeten Typ.

Fehlerbehandlung

Die obige Methode darf keine überprüfte Ausnahme auslösen. Nicht überprüfte Ausnahmen können bei Bedarf ausgelöst werden.

Auswahl des besten Ansatzes

Obwohl es möglich ist, einen Service unter Verwendung beider Ansätze zu implementieren, hat jeder Ansatz einige Vorteile und Einschränkungen, wenn es um die Fähigkeiten geht.

1. **RESTful-Ansatz**
    a. Vorteile
        - Weniger ausführlich und eher an die typische HTTP-Interaktion angelehnt
        - Gebrauchsfertige Fehlerbehandlung auf Transportebene
        - Gebrauchsfertige Unterstützung für Wiederaufnahmeverfahren im Falle von vorübergehenden Ausfällen
        - Gebrauchsfertige Unterstützung für über den Proxyserver aufgebaute Konnektivität
        - Gebrauchsfertige Unterstützung für zukünftige Verbesserungen in Asset Picker in diesem Zusammenhang
    b. Einschränkungen
        - Kann nicht für Nicht-RESTful- oder Nicht-HTTP-Integrationen, wie z. B. Datenbank- oder Dateisysteminteraktionen, verwendet werden
2. **Funktionaler Ansatz**
    a. Vorteile
        - Kann für Nicht-RESTful- oder Nicht-HTTP-Integrationen verwendet werden, wie z. B. Datenbank- oder Dateisystem-Interaktionen
    b. Einschränkungen

- Keine gebrauchsfertige Unterstützung verfügbar für Fehlerbehandlung auf Transportebene, Neuversuche, über den Proxyserver aufgebaute Konnektivität und alle zukünftigen Erweiterungen von Asset Picker
- Die Einbeziehung der Logik für alle fehlenden gebrauchsfertigen Unterstützungen kann den funktionalen Service sehr ausführlich werden lassen.

Wie Sie sehen können, eignet sich der Funktionsansatz gut für nicht-RESTful- oder nicht-HTTP-basierte Integrationen. Jeder Service, der unter Verwendung des RESTful-Ansatzes implementiert wird, kann auch unter Verwendung des funktionalen Ansatzes implementiert werden, wobei alle erforderlichen, von Asset Picker bereitgestellten, gebrauchsfertigen Funktionalitäten berücksichtigt werden. Während der funktionale Ansatz Flexibilität in Bezug auf die Gestaltung der Implementierung bietet, reduziert er aber auch einige nützliche Funktionalitäten.

SDK Plug-in-Entwicklung

Die SDK-Plug-in-Entwicklung liefert Informationen über die verschiedenen Klassen, Schnittstellen und Aufzählungen aus dem Asset Picker SDK mit Hilfe von entsprechenden logischen Einheiten in `asset-integration-starter`-, `aem-integration`- und `wcm-integration`-Referenzprojekten, die als Teil des Development Kits zusammen mit der Asset Picker-Anwendung eingebettet sind.

Generische Typparameter

Generische Typparameter werden für die Implementierung von Serviceschnittstellen verwendet. Weitere Informationen zu Serviceschnittstellen finden Sie unter Serviceimplementierungen (auf Seite 11).

Ein in einem Plug-in befindlicher Service ist lediglich eine Programmiereinheit, die einige Inputs entgegennimmt und den erwarteten Output zurückgibt. In ähnlicher Weise nimmt die REST-API, die von unserem Service umhüllt wird, den angeforderten Inhalt entgegen und erzeugt die gewünschte Antwort. Es erfordert bestimmte generische Notationen für die während des logischen End-to-End-Ablaufs ausgetauschten In- und Outputs.

Asset Picker verwendet RQ, um bestimmte Inputs für den Service zu bezeichnen, sowie RS, um entweder die Output des Service oder die Antwort der entfernten REST-API zu bezeichnen. Die Definition von RS könnte sich je nach dem Ort, an dem sie verwendet wird, ändern.

RestService<RQ, RS>

Verweisen Sie auf die `com.example.service.rest.CustomService`-Klasse aus dem `asset-integration-starter`-Projekt, um einen Überblick über die in der `RestService`-Schnittstelle verwendeten Typparameter zu erhalten. `RestService` ist nur eine von `HttpService` abgeleitete Markerschnittstelle. Die Definition dieser Typparameter ist auch für das `HttpService` ähnlich.

- **RQ**

  Ein Service benötigt einen Input, um seine Operation auszuführen. RQ entspricht dem Typ des Inputs oder der Anfrage, die der Service erfordert, wenn er aufgerufen wird. Das `com.example.service.rest.CustomService` nimmt einen Input vom Typ `ServiceInput` entgegen. Im `ExecutionContext`-Objekt, das an alle Methoden in der `RestService`- oder der `HttpService`-Schnittstelle übergeben wird, wird derselbe Typparameter verwendet. Das an den Service übergebene Input- oder Anforderungsobjekt, wenn es aufgerufen wird, wird durch Aufruf der `getRequest`-Methode im `ExecutionContext`-Objekt erhalten.

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

- **RS**

  Dieser Typparameter entspricht der Art der Antwort (Post-Deserialisierung), die von der entfernten REST-API empfangen wird. Die Serviceimplementierung wählt diesen Parameter basierend auf der Art des Objekts, mit dem sie in der `transformResponse`-Methode arbeiten möchte. Wenn Sie sich die Signatur der `transformResponse`-Methode in der `com.example.service.rest.CustomService`-Klasse ansehen, werden Sie

feststellen, dass das Objekt vom Typ `ApiResponse` als erstes Argument geliefert wird, was dem Typparameter RS der `RestService`-Schnittstelle entspricht.

Anmerkung: Die Deserialisierung erfolgt nach Maßgabe des `Content-Type`-Headers, der in der von der REST-API empfangenen HTTP-Antwort enthalten ist. Der Typ, der als zweites generisches Argument für `RestService` oder das `HttpService` verwendet wird, muss entsprechend annotiert werden, wenn eine Jackson- oder JAXB-Deserialisierung erwartet wird.

Funktionsservice<RQ, RS>

Die `FunctionalService`-Schnittstelle ist analog zur `java.util.function.Function`-Schnittstelle aus der Standard-Java-Bibliothek. Die Typparameter von `FunctionalInterface` haben eine ähnliche Semantik wie die Typparameter der `java.util.function.Function`-Schnittstelle.

- **RQ**

  Stellt die Art des Inputs dar, das dem Service beim Aufruf gegeben wird.
- **RS**

  Stellt den Typ des Wertes dar, der vom Service nach Abschluss zurückgegeben wird.

ServiceGateway<RQ, RS>

Diese Schnittstelle wird für die Implementierung der `getServiceInterface`-Methode aus der `AbstractService<RQ, RS>`-Schnittstelle verwendet. `AbstractService` ist eine wichtige Schnittstelle von `RestService` oder `HttpService` und `FunctionalService` . Die Semantik für RQ und RS für AbstractService ist dieselbe wie bei `RestService` oder `HttpService`. Sie definiert die `getServiceInterface`-Methode, die von einem Service implementiert werden muss. Dies ist die einzige zusätzliche Methode, die ein RESTful-Service implementieren muss, und sie gibt das Klassenobjekt der Ableitung (untergeordnete Schnittstelle) von `ServiceGateway` zurück. Die Definition von `com.hcl.unica.cms.integration.service.gateway.ServiceGateway` sieht wie folgt aus:

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

```
}
```

Die Semantik für den Typparameter RQ ist dieselbe wie oben erwähnt. Der andere Typparameter, RS, stellt das Output des Services dar, der im Plug-in enthalten ist. Er stellt nicht die von der entfernten REST-API oder anderen Zielsystemen erhaltene Antwort dar. Für die Klasse `com.example.service.rest.CustomService` wird das `CustomServiceGateway` als untergeordnete Schnittstelle von `ServiceGateway` definiert, indem Argumente vom Typ `ServiceInput` und `ServiceOutput` verwendet werden, da der Service einen Input vom Typ `ServiceInput` erhält und nach Abschluss den Wert vom Typ `ServiceOutput` zurückgibt.

Anmerkung: Die `getServiceInterface`-Methode in der `com.example.service.rest.CustomService`-Klasse gibt das Klassenobjekt von `CustomServiceGateway` zurück. Die `ServiceGateway`-Schnittstelle (oder ihre untergeordnete Schnittstelle) liefert Informationen über den Input und den Output der Serviceimplementierung. Die `ServiceGateway`Schnittstelle wird weiterhin verwendet, um die Referenz der Serviceinstanz aufzunehmen und ihre Ausführung aufzurufen.

Serviceaufruf

Das Asset-Integration-Starter-Projekt enthält eine com.example.service.client.CustomServiceClient-Klasse zur Veranschaulichung des Serviceaufrufs.

Die `invocationDemo`-Methode in dieser Klasse erhält die Referenz auf `custom-service` durch Verwendung der statischen `getServiceGateway`-Methode aus der `ServiceGatewayFactory`-Klasse. Die `getServiceGateway`-Methode benötigt drei Argumente, um die Serviceinstanz zurückzugeben. Die Argumente sehen wie folgt aus:

- `String systemId`

  Dieser Systembezeichner ist derselbe wie derjenige, der in der Service-Metainformationsdatei verwendet wird, um den Service zu deklarieren, dessen Instanz abgerufen werden muss.
- `String serviceName`

Dies ist der Name des Service, dessen Instanz abgerufen werden muss. Der Name muss mit dem in der Meta-Informationsdatei des Service angegebenen Namen übereinstimmen.

- `Class<T> gatewayClass`

  Dabei muss es sich um das Klassenobjekt der ServiceGateway-Schnittstelle (oder ihrer untergeordneten Schnittstelle) handeln. Er muss mit dem Rückgabewert der getServiceInterface-Methode in der entsprechenden Service-Implementierung übereinstimmen.

Die invocationDemo-Methode in der com.example.service.client.CustomServiceClient-Klasse verwendet CustomServiceGateway (gatewayClass), um die Serviceinstanz des benutzerdefinierten Service (serviceName) für das System Foo (systemId) zu erhalten. Der folgende Codeschnipsel dient zu Ihrer Information:

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

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

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

Der Rückgabetyp von getServiceGateway ist ebenfalls CustomServiceGateway. Die erhaltene Serviceinstanz kann verwendet werden, um den jeweiligen Service auszuführen, indem das erforderliche Eingabeobjekt geliefert wird.

Anmerkung:

- Der Rückgabetyp von `getServiceGateway` ist ebenfalls `CustomServiceGateway`. Die erhaltene Serviceinstanz kann verwendet werden, um den jeweiligen Service auszuführen, indem das erforderliche Eingabeobjekt geliefert wird. Die Methode Execute wird auf der `ServiceGateway`-Schnittstelle verwendet, um den Service auszuführen. Sie werden feststellen, dass der Typ der Eingabe an `custom-service` derselbe ist wie der Typ, der für die Serviceimplementierung in der Klasse `com.example.service.rest.CustomService` oder dem `com.example.service.functional.CustomService class` verwendet wird. Es handelt sich um denselben Ausgabetyp, der für die Definition der `CustomServiceGateway`-Schnittstelle verwendet wird, deren Klassenobjekt in beiden Versionen der `getServiceInterface`-Klasse von der `CustomService`-Methode zurückgegeben wird.
- Die Klasse `com.example.service.rest.CustomService` und die Klasse `com.example.service.functional.CustomService` repräsentieren denselben Service, der mit zwei verschiedenen Ansätzen implementiert wurde. Die Service-Metainformationsdateien im `asset-integration-starter`-Projekt, die das `META-INF/rest-content-services.yml` und das `META-INF/functional-content-services.yml` verwenden, haben einen Eintrag für `custom-service`, der auf die jeweiligen Versionen des `factoryClass` verweist. Diese beiden Versionen werden nur zu Illustrationszwecken vorgelegt. Für alle praktischen Zwecke wird vom Asset Picker nur eine Version der Serviceimplementierung vorausgesetzt. Unabhängig vom Ansatz, der für die Implementierung des Service verwendet wird, bleibt die Methode für den Serviceaufruf dieselbe.

Mehrfach partitionierte Clients

Im Falle einer mehrfach partitionierten Client-Anwendung von Asset Picker gibt die oben erwähnte Methode zur Beschaffung der Serviceinstanz in geeigneter Weise die für den Servicegateway spezifische Referenzpartition zurück. Das `ExecutionContext`-Objekt, das an verschiedene Rückrufmethoden übergeben wird, enthält die notwendigen partitionsspezifischen Informationen, mit denen gearbeitet werden kann.

Ausführungskontext

Nahezu jede Methode in einem Serviceimplementierungsvertrag erhält eine Instanz der `com.hcl.unica.cms.model.request.ExecutionContext`-Klasse.

Dieses Objekt enthält alle Kontextinformationen, die erforderlich sind, damit ein Service seine Operation ausführen kann. Im Folgenden sind die Methoden der ExecutionContext-Klasse aufgeführt, mit denen verschiedene Arten von Informationen während der Serviceausführung abgerufen werden können:

- **T getRequest()**

  Diese Methode kann verwendet werden, um das Input- oder Anforderungsobjekt zu erhalten, das an den Service übergeben wird, wenn dieser mit der execute-Methode auf der `ServiceGateway`-Schnittstelle ausgeführt wird. (Der Rückgabetyp T ist der Typparameter, der dem generischen Argument entspricht, das zur Definition des Service verwendet wird.)
- **Map<String, Object> getAttributes()**

  Gibt zusätzliche Attribute zurück, die sich auf die aktuelle Serviceausführung beziehen, wie z. B. HTTP-Antwortstatus und Header für den aktuellen HTTP-Aufruf.

  Anmerkung: `Content-Type` Der HTTP-Header wird aufgrund besonderer Überlegungen im zugrunde liegenden Framework als `contentType`-Schlüssel aufgefüllt.
- **ServiceConfig getServiceConfig()**

  Diese Methode gibt eine Instanz der `com.hcl.unica.cms.integration.config.ServiceConfig`-Klasse zurück. Dieses Objekt enthält die Konfigurationen, die in der Service-Metainformationsdatei für den jeweiligen Service vorgenommen wurden.
- **InstanceConfig getInstanceConfig()**

  Diese Methode gibt eine Instanz der `com.hcl.unica.cms.integration.config.InstanceConfig`-Klasse zurück. Dieses Objekt enthält alle Konfigurationen, die in Unica Platform für das Zielsystem vorgenommen wurden (Instanz in diesem Methodennamen bezieht sich auf die Zielsysteminstanz und nicht auf die Serviceinstanz). Im Falle von Konfigurationen

mit mehreren Partitionen wird dieses Objekt von Asset Picker entsprechend gefüllt, um eine partitionsspezifische Konfiguration aufzunehmen. Für Informationen zu den verschiedenen Instanzkonfigurationseinstellungen in Unica Platform siehe das Administratorhandbuch von Unica Asset Picker.

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

  Die Verwendung dieser Methode ist auf Asset Picker beschränkt. Vermeiden Sie es, diese Methode mit den Plug-ins zu verwenden.

Benutzerdatenquelle

Verwenden Sie `ExecutionContext`, um die zutreffende Benutzerdatenquelle (Berechtigungsnachweise) abzurufen, indem Sie durch das `InstanceConfig`-Objekt navigieren:

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

Das von der `getDataSourceCredentials`-Methode zurückgegebene `DataSourceCredentials`-Objekt enthält die ausgewählte Datenquelle auf der Grundlage der Strategie, die in der Platform-Konfiguration für **Benutzer-Berechtigungsnachweise** festgelegt wurde. Daher werden Plug-ins keine logische Entscheidung bezüglich der richtigen Auswahl der Benutzerdatenquelle treffen.

In ähnlicher Weise gibt die `getUnicaToken`-Methode, die auf dem `InstanceConfig`-Objekt aufgerufen wird, ein UnicaToken-Objekt zurück, das das Unica Token enthält, das für den Aufruf von APIs von Unica-Anwendungen erforderlich ist.

Standardservices und spezielle Typen

Der Plug-in-Entwickler muss eine `RestService/HttpService`oder `FunctionalService`-Schnittstelle implementieren, um einen individuellen Service zu erstellen.

Außerdem kann der implementierte Service ausgeführt werden, indem man mit der `ServiceGatewayFactory.getServiceGateway`-Methode eine Referenz darauf abruft. Der Asset Picker baut auf diesem Design auf und definiert bestimmte Standardleistungen. Dies sind die Services „Einfache Suche“ (`simple-search`), „Ressourcenlader“ (`resource-loader`) und „Asset-Auswahlrückruf“ (`asset-selection-callback`). Der Asset Picker bietet

spezialisierte Schnittstellen, die von `RestService` und `FunctionalService` für jeden dieser Standardservices abgeleitet wurden, um deren Implementierung mit dem RESTful- oder Functional-Ansatz zu erleichtern.

Anrufung von Standardservices

Einmal in der Meta-Informationsdatei des Service festgelegt und entweder mit dem RESTful- oder dem funktionalen Ansatz implementiert, ruft Asset Picker sie in den folgenden Szenarien auf:

- **Einfache Suche (`simple-search`)**

  Immer wenn Asset Picker von seiner Client-Anwendung gegen das Zielsystem Inhalte oder Asset-Suchanfragen erhält, ruft er den für das jeweilige System implementierten `simple-search`-Service auf. Asset Picker liefert bei Anruf den notwendigen Input für den `simple-search`-Service. Von `simple-search`-Service erhaltene Suchbegriffe werden dann an die Client-Anwendung zurückgegeben. Die Identifizierung des Zielsystems erfolgt auf Grundlage der `systemId`-Eigenschaft, die in der Service-Metainformationsdatei verwendet wird, und der entsprechenden **Systembezeichner**-Einstellung in Unica Platform, die während des Onboardings des Zielsystems eingegeben wird. Dieser Service muss durch das Plug-in implementiert werden, sonst landet die Inhaltssuchanfrage als 404-Antwort auf der Client-Anwendung.
- **Ressourcenlader (`resource-loader`)**

  Der `resource-loader`-Service wird von Asset Picker nur dann ausgeführt, wenn ein indirekter (oder authentifizierter) Zugriff auf das Suchobjekt auf dem Zielsystem erforderlich ist. Da es keine direkte URL zu einem einzelnen Suchbegriff auf dem Zielsystem gibt, kann der `resource-loader`-Service durch das Plug-in implementiert werden. Da es sich um einen optionalen Service handelt, erkennt Asset Picker automatisch, ob die `resource-loader`-Implementierung für das Zielsystem vorhanden ist und ruft sie entsprechend auf.
- **Asset-Auswahlrückruf (`asset-selection-callback`)**

  Asset Picker führt diesen Service für das Zielsystem aus, wenn die Client-Anwendung eines der vom `simple-search`-Service zurückgegebenen Suchbegriffe auswählt. Da der

Service optional ist, kann Asset Picker die Verfügbarkeit dieses Services automatisch erkennen und ihn entsprechend aufrufen.

Spezielle Typen

Nachfolgend sind die spezialisierten Ableitungen von `RestService`, `HttpService`, und `FunctionalService`-Schnittstellen und ihre verwandten Typen für alle Standardservices aufgeführt. Nutzen Sie das `asset-integration-starter`-Projekt zur Implementierung der in den folgenden Themen erwähnten Details:

- Ableitungen von RestService *(auf Seite 26)*
- Ableitungen von HttpService *(auf Seite 30)*
- Ableitungen von Funktionsservice *(auf Seite 32)*

Ableitungen von RestService

Ableitungen der Schnittstelle von RestService erleichtern die Erstellung einer RESTful-Implementierung von Standardservices.

Einfache Suche (`simple-search`)

Im Folgenden sind die für den `simple-search`-Service verfügbaren speziellen Schnittstellen und Klassen aufgeführt:

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

  Die `com.example.service.rest.SimpleSearchService`-Klasse in `asset-integration-starter`-Projekt ist eine Schnellstartimplementierung für den RESTful `simple-search`-Service. Seine übergeordnete Klasse ist `com.hcl.unica.cms.integration.service.search.RestSearchService`-Klasse.

  Die `RestSearchService`-Klasse verfügt über einen Typparameter RS, der die Art der von der entfernten REST-API empfangenen Antwort (Post-Deserialisierung) darstellt. In diesem Fall ist es die `SimpleSearchResponse`-Klasse, die innerhalb des `asset-integration-starter`-Projekts definiert ist.

  `RestSearchService`-Klasse implementiert die `RestService`-Schnittstelle und definiert die `SearchRequest`-Klasse als das Typargument RQ für `RestService`. Somit wird

das Objekt von `SearchRequest` zum Input für alle `simple-search`-Services (derselbe Input wird auch für das funktionale Pendant der einfachen Suche verwendet). Die `SearchRequest`-Klasse ist Teil des Asset Picker SDK.

Zusätzlich zur Definition des Inputtyps für den `simple-search`-Service setzt die `RestSearchService`-Klasse auch die `transformResponse`-Methode außer Kraft und definiert den Rückgabewert dieser Methode als vom `ContentPage`-Typ. `ContentPage` ist auch Teil des Asset Picker SDK und enthält das Suchergebnis und die zugehörigen Paginierungsdetails.

Das Plug-in muss seine `simple-search`-Implementierung von `RestSearchService`-Service ableiten, um von Asset Picker als `simple-search`-Service erkannt zu werden.

`RestSearchService` wird von der abstrakten Klasse `com.hcl.unica.cms.integration.service.search .AbstractSearchService` abgeleitet.

Wir empfehlen, sich die `com.aem.service.AemSimpleSearchService`-Klasse aus dem `aem-integration`-Projekt anzusehen, um mehr darüber zu erfahren, wie die `SearchRequest`-Klasse und `ContentPage`-Klasse während der Serviceimplementierung verwendet werden.

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

  Dies ist eine gemeinsame Basisklasse sowohl für RESTful- als auch für funktionale `simple-search`-Implementierungen. Die Details dieser Klasse gelten also auch für die funktionale Implementierung von `simple-search`.

  Diese Klasse definiert die `com.hcl.unica.cms.integration.service.gateway.SimpleSearchServiceGateway`-Schnittstelle als Servicegateway für den `simple-search`-Service. ServiceGateways sind die Mittel zur programmatischen Definition von Input- und Outputtypen des Service und der Arbeit mit dem Service. Ein genauerer Blick auf diese Schnittstelle lässt erkennen, dass das `simple-search` das `SearchRequest`-Objekt nimmt und das `ContentPage`-Objekt zurückgibt.

  Zusätzlich zur Definition der Serviceschnittstelle für `simple-search` wird eine weitere abstrakte Methode für den `simple-search`-Service eingeführt. Jede `simple-search`-

Implementierung muss diese neue Methode überschreiben und implementieren. Bitte beachten Sie, dass diese Methode sehr `simple-search`-spezifisch ist und nichts mit anderen Standard- und kundenspezifischen Services zu tun hat. Die Signatur dieser neuen Methode sieht wie folgt aus:

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

Die Implementierung dieser Methode liefert eine Liste von Zeichenfolgen, die die Kategorien von Inhalten oder Assets repräsentieren, nach denen im Zielsystem gesucht werden soll. Mit den Werten in dieser Liste ist keine spezifische Semantik verbunden. Dabei kann es sich um einen beliebigen aussagekräftigen Text handeln. Es wirkt während des Suchvorgangs als Filter für die Client-Anwendung. Die Client-Anwendung kann Werte aus dieser Liste senden, um die Suchbegriffe zu filtern. Von der Client-Anwendung empfangene Werte können aus dem `ExecutionContext`-Objekt abgerufen werden, indem man durch die `getRequest`-Methode navigiert und dann `getTypes()` darauf aufruft. `getRequest()` gibt das `SearchRequest`-Objekt zurück, das die Gruppe der unterstützten Typen enthält, die die Client-Anwendung gesendet hat, um das Suchergebnis zu filtern. Die Implementierung der einfachen Suche behandelt diese Gruppe von Werten entsprechend der Programmierschnittstelle des Zielsystems und filtert die Suchbegriffe nach diesen Werten. Schauen Sie sich die `getSupportedContentTypes`-Methode in `com.aem.service.AemSimpleSearchService`-Klasse in `aem-integration`-Projekt an, und erfahren Sie, wie die `restrictContentTypes`-Methode in `com.aem.service.simplesearch.SimpleSearchRequestBuilder`-Klasse das Suchergebnis auf die ausgewählten Typen einschränkt.

Asset-Auswahlrückruf (`asset-selection callback`)

Im Folgenden sind die für den `asset-selection-callback`-Service verfügbaren speziellen Schnittstellen und Klassen aufgeführt:

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

Die `com.example.service.rest.ContentSelectionCallbackService`-Klasse in `asset-integration-starter`-Projekt ist eine Schnellstartimplementierung für den RESTful `asset-selection-callback`-Service. Seine übergeordnete Klasse ist die folgende Klasse:

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

Die `RestAssetSelectionCallbackService`-Klasse verfügt über einen Typparameter RS, der die Art der von entfernten REST-API empfangenen Antwort (Post-Deserialisierung) darstellt. In diesem Fall ist es die Zeichenkettenklasse, die in der Standard-Java-Bibliothek definiert ist.

Die `RestAssetSelectionCallbackService`-Klasse implementiert die `RestService`-Schnittstelle und definiert die `com.hcl.unica.cms.model.request.assetselectioncallback.AssetSelectionDetails`-Klasse als das Typargument RQ für RestService. Somit wird das Objekt von `AssetSelectionDetails` zum Input für alle `asset-selection-callback`-Services (derselbe Input wird auch für das funktionale Pendant des Asset-Auswalrückrufs verwendet). Die `AssetSelectionDetails`-Klasse ist ein Teil des Asset Picker SDK. Die `AssetSelectionDetails`-Klasse enthält die Details eines von der Client-Anwendung ausgewählten Assets (Suchbegriffs) und die Kontextinformationen wie z. B. die Suchanfrage, die zum Suchergebnis mit dem ausgewählten Begriff geführt hat.

Plug-in muss seine `asset-selection-callback`-Implementierung vom `RestAssetSelectionCallbackService`-Service ableiten, um vom Asset Picker als `asset-selection-callback`-Service erkannt zu werden (funktionelles Pendant ist ebenfalls eine gültige Option, von der es abgeleitet werden kann).

`RestAssetSelectionCallbackService` wird von der folgenden abstrakten Klasse abgeleitet:

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

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

```
.AbstractAssetSelectionCallbackService
```

Dies ist eine gemeinsame Basisklasse sowohl für RESTful- als auch für funktionale `asset-selection-callback`-Implementierungen. Die hier erwähnten Details dieser Klasse gelten also auch für die funktionale Implementierung von `asset-selection-callback`.

Die folgende Klasse definiert Schnittstelle als Servicegateway für den `asset-selection-callback`-Service:

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

ServiceGateways sind die Mittel zur Definition von Input- und Outputtypen des Service und dienen der programmatischen Arbeit mit dem Service. Ein genauerer Blick auf diese Schnittstelle lässt erkennen, dass das `asset-selection-callback` das `AssetSelectionDetails`-Objekt übernimmt und ein beliebiges Objekt zurückgibt. Derzeit wird der Rückgabewert von `asset-selection-callback` vom Asset Picker ignoriert.

Ableitungen von HttpService

Nur der `resource-loader`-Standardservice ist als `HttpService` implementiert, da er sich auf die standardmäßige HTTP-GET-Operation bezieht. Sie können auch `RestService` verwenden, ohne dass dadurch irgendwelche Fähigkeiten verloren gehen.

Ressourcenlader (`resource-loader`)

Im Folgenden sind die spezialisierten Schnittstellen und Klassen aufgeführt, die für den Ressourcenlader-Service zur Verfügung stehen:

- `com.hcl.unica.cms.integration.service.resourceloader.DefaultWebResourceLoaderService`
  Die `com.example.service.rest.ResourceLoaderService`-Klasse in `asset-integration-starter`-Projekt ist eine Schnellstartimplementierung für den `resource-loader`-Service und wird von der folgenden Klasse abgeleitet:

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

```
.DefaultWebResourceLoaderService
```

`DefaultWebResourceLoaderService`-Klasse ist die Standardimplementierung des vom Asset Picker SDK bereitgestellten `resource-loader`-Service. Wenn das Plug-in keinen eigenen `resource-loader`-Service implementiert, greift Asset Picker auf diese Standardimplementierung zurück. Die vom Asset Picker SDK bereitgestellte Standardimplementierung von `resource-loader` folgt einfach der angegebenen Ressourcen-URL und ruft die Webressource vom Zielsystem ab. Es enthält die standardmäßige HTTP-GET-Operation.

Wenn das Plug-in eine eigene `resource-loader`-Implementierung benötigt, die den Standard-HTTP-GET leicht modifiziert, empfehlen wir, es von der `DefaultWebResourceLoaderService`-Klasse x abzuleiten.

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

  Die zuvor behandelte `DefaultWebResourceLoaderService`-Klasse wird von der abstrakten Klasse `HttpWebResourceLoaderService` abgeleitet. Diese Klasse definiert den Eingabetyp und den Typ der HTTP-Antwort, die von der Ziel-URL für den `resource-loader`-Service als `com.hcl.unica.cms.model.request.resourceloader.ResourceRequest` bzw. Byte empfangen wird. Die `ResourceRequest`-Klasse enthält die relative Ressourcen-URL und den Instanzbezeichner (der Instanzbezeichner ist genau derselbe wie das `systemId`, das überall in Asset Picker verwendet wird). In ähnlicher Weise arbeitet `resource-loader` mit einem Bytearray, wenn der Inhalt von einer entfernten HTTP-URL erfolgreich gelesen wurde.

  Wenn das Plug-in seine `resource-loader`-Implementierung nicht von der `DefaultWebResourceLoaderService`-Klasse ableitet, muss es zumindest von der `HttpWebResourceLoaderService`-Klasse abgeleitet werden, um von Asset Picker als `resource-loader`-Service erkannt zu werden. (Ein funktionelles Pendant ist ebenfalls eine gültige Option, von der es abgeleitet werden kann.)

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

  Die im vorigen Punkt behandelte `HttpWebResourceLoaderService`-Klasse wird von der abstrakten Klassen `AbstractWebResourceLoaderService` abgeleitet. Diese Klasse

definiert die folgende Schnittstelle des Servicegateways für den `resource-loader`-Service:

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

Informationen über die Rolle von Servicegateways beim Serviceaufruf finden Sie unter Serviceaufruf (auf Seite 20). `ResourceLoaderServiceGateway`-Schnittstelle definiert `ResourceRequest` und `WebResource<?>` als Input- und Outputtypen für den Ressourcenlader-Service. Die `com.hcl.unica.cms.model.response.resourceloader.WebResource`-Klasse ist lediglich ein Wrapper für HTTP-Antwortheader, Hauptteil und Cookies, die von einer entfernten URL empfangen werden.

Ableitungen von Funktionsservice

Ableitungen der Schnittstelle von Funktionsservice erleichtern die Schaffung einer funktionalen Implementierung von Standardservices. Funktionsservice ist nur ein Objekt mit einer öffentlichen Methode, das einen bestimmten Input übernimmt und den gewünschten Output erzeugt.

Einfache Suche (`simple-search`)

Im Folgenden sind die spezialisierten Schnittstellen und Klassen aufgeführt, die für die einfache Suche verwendet werden können:

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

  Die `com.example.service.functional.SimpleSearchService`-Klasse im `asset-integration-starter`-Projekt ist eine Schnellstartimplementierung für den `simple-search service`-Funktionsservice. Seine übergeordnete Klasse ist die `com.hcl.unica.cms.integration.service.search.SearchService`-Klasse.

  Die `SearchService`-Klasse implementiert die `FunctionalService`-Schnittstelle und definiert die `SearchRequest`-Klasse und `ContentPage`-Klasse als die Typargumente RQ & RS für den Funktionsservice. Dadurch wird das Objekt des `SearchRequest` zu einem

Input für alle `simple-search`-Services, und das `ContentPage` wird nach Beendigung des Services als Output erwartet.

Das Plug-in muss seine `simple-search`-Implementierung von dem `SearchService`-Service ableiten um vom Asset Picker als `simple-search`-Service erkannt zu werden (RESTful-Pendant ist ebenfalls eine gültige Option, von der es abgeleitet werden kann).

Das `SearchService` wird von der `com.hcl.unica.cms.integration.service.search .AbstractSearchService` `abstract`-Klasse abgeleitet. Es stellt eine weitere abstrakte Methode namens `getSupportedContentTypes` zur Implementierung des `simple-search`-Service vor.

Asset-Auswahlrückruf (`asset-selection-callback`)

Im Folgenden sind die für den `asset-selection-callback`-Service verfügbaren speziellen Schnittstellen und Klassen aufgeführt:

- `com.hcl.unica.cms.integration.service.assetselectioncallback.AssetSelectionCallbackService`
  Die `com.example.service.functional.ContentSelectionCallbackService`-Klasse im `asset-integration-starter`-Projekt ist eine Schnellstartimplementierung für `asset-selection-callback`-Funktionsservice. Seine übergeordnete Klasse ist die folgende Klasse:

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

  Die `AssetSelectionCallbackService`-Klasse implementiert die `FunctionalService`-Schnittstelle und definiert die `AssetSelectionDetails`-Klasse und `Object`-Klasse als die Typargumente RQ & RS für den `FunctionalService`. Somit wird das Objekt des `AssetSelectionDetails` zu einem Input für alle `asset-selection-callback`-Services, und das `Object` oder sein Untertyp wird nach Beendigung des Services als Output erwartet (die gleichen Input- und Outputtypen werden für das RESTful-Pendant des Asset-Auswahlrückrufs verwendet). Die `AssetSelectionDetails`-Klasse ist Teil des Asset Picker SDK.

Plug-in muss seine `asset-selection-callback`-Implementierung von dem `AssetSelectionCallbackService`-Service ableiten, um vom Asset Picker als `asset-selection-callback`-Service erkannt zu werden (RESTful-Pendant ist ebenfalls eine gültige Option, von der es abgeleitet werden kann).

Das `AssetSelectionCallbackService` wird von der folgenden abstrakten Klasse abgeleitet:

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

Ressourcenlader (`resource-loader`)

Im Folgenden sind die spezialisierten Schnittstellen und Klassen aufgeführt, die für den Ressourcenlader-Service zur Verfügung stehen:

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

  Die `com.example.service.functional.ResourceLoaderService`-Klasse im `asset-integration-starter`-Projekt ist eine Schnellstartimplementierung für `resource-loader` Funktionsservice. Seine übergeordnete Klasse ist die `com.hcl.unica.cms.integration.service.resourceloader.WebResourceLoaderService`-Klasse.

  Die `WebResourceLoaderService`-Klasse implementiert die `FunctionalService`-Schnittstelle und definiert die `ResourceRequest` und `WebResource`-Klassen als die Typargumente RQ & RS für den `FunctionalService`. Somit wird das Objekt des `ResourceRequest` zu einem Input für alle `resource-loader`-Services, und das `WebResource` wird nach Abschluss des Services als Output erwartet (die gleichen Input- und Outputtypen werden für das RESTful-Pendant des `resource-loader` verwendet).

  Das Plug-in muss seine `resource-loader`-Implementierung von dem `WebResourceLoaderService`-Service ableiten, um vom Asset Picker als `resource-loader`-Service erkannt zu werden (RESTful-Pendant ist ebenfalls eine gültige Option, von der es abgeleitet werden kann).

  Das `WebResourceLoaderService` wird von der folgenden abstrakten Klasse abgeleitet:

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

Standardausnahmen

Zu den Standardausnahmen gehören die vom Asset Picker SDK bereitgestellten Ausnahmen, die von den Plug-ins verwendet werden können, um verschiedene Fehlerbedingungen während der Serviceausführung zu übermitteln.

RESTful-Ansatz

Asset Picker behandelt Fehlerzustände, die durch Services entstehen, die mit dem RESTful-Ansatz implementiert wurden.

Darüber hinaus leitet Asset Picker die Ausführung von entfernten API-Aufrufen für RESTful-Integrationen ein und handhabt diese, sodass die erfolgreiche Durchführung der gesamten HTTP-Operation verfolgt werden kann. Daher benötigen die Plug-ins keine spezielle Ausnahme, um das Fehlschlagen des REST-Aufrufs zu übermitteln. Wenn innerhalb der Serviceimplementierung etwas schief geht, reicht jede entsprechende ungeprüfte Ausnahme aus, um den Operationsfehler zu vermitteln. Solche Ausnahmen werden weiterhin als 502 HTTP-Antwort an den Client übermittelt.

Funktionaler Ansatz

Da Asset Picker im Falle von Funktional Services die ausgehenden Verbindungen nicht initiiert und verwaltet, kann er den End-to-End-Erfolg nicht verfolgen.

Daher sieht er bestimmte Standardausnahmen vor, die die Serviceimplementierungen auslösen können, um relevante Fehlerbedingungen zu vermitteln. Diese Ausnahmen beziehen sich auf die Kommunikation mit dem Ziel-Inhaltsrepository und sind innerhalb des Ausnahmepakets com.hcl.unica.cms.integration.exception vorhanden.

- **RepositoryNotFoundException**

  Diese Ausnahme muss verwendet werden, wenn das Zielsystem oder Inhaltsrepository nicht gefunden werden kann. Als Alternative kann auch

`java.net.UnknownHostException` verwendet werden. Diese Ausnahme wird auch als 404-HTTP-Antwort an den Client übermittelt.

- **ServiceNotFoundException**

  Diese Ausnahme muss verwendet werden, wenn der entfernte Endpunkt 404 zurückgibt oder wenn der Zielservice nicht mehr existiert. Das Fehlen des Zielsystems und das Fehlen des erforderlichen Service werden als unterschiedliche Faktoren betrachtet. Somit vermittelt das `ServiceNotFoundException` die Anwesenheit des Zielsystems und das Fehlen des erforderlichen Services oder Features auf dem Zielsystem. Beispielsweise kann im Falle von Inhalten, die aus der Datenbank geholt werden, das Fehlen der erforderlichen Tabelle (oder das Fehlen der Zugriffsberechtigung) mit Hilfe dieser Ausnahme übermittelt werden. Diese Ausnahme wird auch als 404-HTTP-Antwort an den Client übermittelt.
- **InaccessibleRepositoryException**

  Diese Ausnahme muss verwendet werden, um nicht erreichbare oder unzugängliche Zielsysteme zu übermitteln, wie z. B. Verbindungs-Timeout. Als Alternative kann auch `java.net.ConnectException` verwendet werden. Diese Ausnahme wird auch als 503-HTTP-Antwort an den Client übermittelt.
- **SluggishRepositoryException**

  Wenn die Antwort vom Zielsystem nicht innerhalb der erwarteten Zeit eintrifft, muss diese Ausnahme genutzt werden, um die Langsamkeit des Zielsystems zu vermitteln. Als Alternative kann auch `java.net.SocketTimeoutException` verwendet werden. Diese Ausnahme wird auch als 504-HTTP-Antwort an den Client übermittelt.
- **InternalRepositoryErrorException**

  Diese Ausnahme muss verwendet werden, wenn das Plug-in einen temporären oder unerwarteten Fehler vom Zielsystem erhält, um die Probleme im Zielsystem zu übermitteln. Diese Ausnahme wird auch als 502-HTTP-Antwort an den Client übermittelt.

Alle anderen Ausnahmen werden als 502 HTTP-Antwort an den Client übermittelt. In jedem Fall wird die Nachricht in der Ausnahme niemals an den Client zurückgegeben. Jeder HTTP-Antwortcode trägt eine feste, generische und lokalisierte Nachricht.

Protokollfunktionen (Loggers)

Asset Picker bietet eine Protokollierungsschnittstelle unter Verwendung der `slf4j`-Bibliothek. Durch Hinzufügen von Abhängigkeiten für die `slf4j`-Bibliothek können die Plug-ins deren API verwenden, um Protokollfunktionen (Loggers) innerhalb von Serviceimplementierungen hinzuzufügen.

Sowohl die Starter- als auch die in `dev-kits` enthaltenen Referenzprojekte verwalten ihre Abhängigkeiten mit Apache Maven. Der folgende Eintrag befindet sich in der POM-Datei:

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

Verwenden Sie 1.7.26 oder eine spätere Version von `slf4j-api`, um Konflikte zu vermeiden. Sobald die erforderliche Abhängigkeit hinzugefügt ist, kann das das Logger-Objekt durch direkten Zugriff auf die `slf4j`-API erhalten werden.

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

Alternativ kann auch das Projekt Lombok verwendet werden, um das Logger-Objekt für Ihre Klasse zu erhalten. Lombok stellt @Slf4j-Annotation zur Verfügung, mit der die zuvor erwähnte Eigenschaft innerhalb der annotierten Klasse injiziert werden kann. Für weitere Informationen über das Projekt Lombok besuchen Sie bitte die offizielle Webseite.

Zusätzlich sind die Anwendungsprotokolle im `AssetPicker/logs`-Verzeichnis unter „platform home“ zu finden. Standardmäßig werden alle Loggers Ihres Plug-ins in der gemeinsamen Protokolldatei gespeichert, die in der `AssetPicker/conf/logging/log4j2.xml`-Datei konfiguriert ist. Sie können die `log4j2.xml`-Konfigurationsdatei ändern, um Ihre Loggers zur Fehlerbehebung während der Entwicklung in eine andere Datei zu leiten. Die Konfiguration von `log4j2` gehört nicht zum Umfang dieses Handbuchs. Weitere Informationen finden Sie in der offiziellen Dokumentation von Apache Log4j2.

Überprüfung und Fehlerbehebung

Sie sollten die End-zu-End-Integration überprüfen, nachdem das Plug-in entwickelt worden ist. Legen Sie die JAR-Datei, die die Plug-in-Implementierung enthält, in den Klassenpfad des Anwendungsservers, auf dem der Asset Picker bereitgestellt wird. Konfigurieren Sie zusätzlich das entsprechende Inhaltsrepository in der Platform-Konfiguration unter unterstützte Anwendung.

Anmerkung: Derzeit kann nur Unica Centralized Offer Management auf Asset Picker zugreifen.

Weitere Informationen zu Konfigurationsdetails finden Sie im Administrationshandbuch für Unica Asset Picker.

Nachdem das Plug-in bereitgestellt und die Konfigurationen angepasst wurden, starten Sie die Asset Picker-Anwendung neu. Die Änderungen in der Benutzerdatenquelle erfordern keinen Neustart.

Überprüfung der Integration

Obwohl Sie Asset Picker mit REST-Endpunkten überprüfen können, empfehlen wir Ihnen, die End-zu-End-Integration zu überprüfen, indem Sie die entsprechende Benutzeroberfläche in Unica durchlaufen.

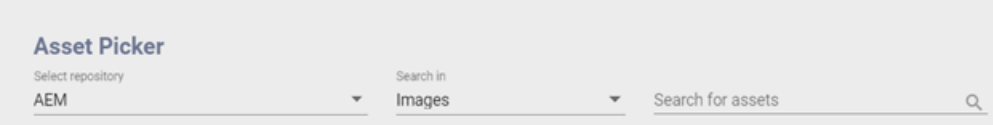
In diesem Release ermöglicht das benutzerdefinierte Attribut für eine Asset-URL in Angeboten die Arbeit mit Asset Picker. Wenn Sie zum entsprechenden Bildschirm in das Angebotsmanagment navigieren, können Sie Asset Picker für ein benutzerdefiniertes URL-Picker-Attribut aufrufen. Sobald das Popupfenster von Asset Picker aufgerufen wird, verwenden Sie die folgenden Richtlinien, um die verschiedenen in Ihrem Plug-in enthaltenen Services zu überprüfen:

- Systemregistrierung überprüfen
- Den `simple-search`-Service überprüfen
- Den `resource-loader`-Service überprüfen
- Den `asset-selection-callback`-Service überprüfen

Systemregistrierung überprüfen

Ihr Inhaltsrepository muss in der ersten Dropdownliste aufgeführt sein, wie in der folgenden Abbildung gezeigt. Zusätzlich sollten alle unterstützten Inhaltstypen in dem nächsten Dropdownfeld angezeigt werden.

Abbildung 3. Systemregistrierung überprüfen



Verwenden Sie die Entwicklertools Ihres Browsers zur Fehlerbehebung der vom Backend des Asset Picker empfangenen Antwort. Auf die folgende Endpunkt-URL wird zugegriffen, wenn Asset Picker gestartet wird, um die Liste der für den angemeldeten Benutzer verfügbaren Inhaltsrepositorys abzurufen:

```
/api/AssetPicker/instances
```

Wenn Ihr Inhaltsrepository nicht aufgeführt ist, stellen Sie sicher, dass der Systembezeichner, der im Plug-in verwendet wird, und der Systembezeichner, der in der Platform-Konfiguration verwendet wird, übereinstimmen. Zusätzlich können Sie die Anwendungsprotokolle einsehen, um nach möglichen Fehlern oder Ausnahmen zu suchen.

Den Service „Einfache Suche“ überprüfen.

Nachdem Sie eine gültige Suche in Ihrem Inhaltsrepository ausgeführt haben, erscheint das erwartete Suchergebnis mit relevanten Details.

Verwenden Sie die Entwicklertools Ihres Browsers zur Fehlerbehebung der vom Backend des Asset Picker empfangenen Antwort. Bei der Ausführung des Suchvorgangs wird auf die folgende Endpunkt-URL zugegriffen:

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

In der oben genannten URL:

- `WCM` ist der Systembezeichner für Ihr Inhaltsrepository.
- `query` enthält den Suchbegriff.

- `types` enthalten die Liste der unterstützten Inhaltstypen zum Filtern des Suchergebnisses.
- `page` ist die Seitenzahl des Suchergebnisses.
- `size` ist die maximale Anzahl von Elementen, die auf einer einzelnen Seite vorgesehen sind.

Verifizieren Sie den Ressourcenlader-Service

Wenn in der Platform-Konfiguration der **anonyme Inhalt** für Ihr Inhaltsrepository auf `Nein` konfiguriert ist, wird auf die Suchobjekte über Asset Picker zugegriffen, statt direkt mit dem Zielrepository verbunden zu werden. Um die Genauigkeit des `resource-loader`-Service zu gewährleisten, überprüfen Sie die Zugänglichkeit bzw. Sichtbarkeit jedes Suchbegriffs.

Verwenden Sie die Entwicklertools Ihres Browsers zur Fehlerbehebung der vom Backend des Asset Picker empfangenen Antwort. Auf die folgende URL wird zugegriffen, um einzelne Suchbegriffe abzurufen:

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

In der früheren URL ist WCM der Systembezeichner für Ihr Inhaltsrepository, und die Ressource enthält die relative URL zum entsprechenden Suchobjekt. Diese URL ähnelt der URL, die der Service für die einfache Suche beim Umwandeln des Suchergebnisses in das Ressourcen-URL-Attribut des entsprechenden Assets einfügt.

Wenn in der Platform-Konfiguration der **Anonyme Inhalt** für Ihr Inhaltsrepository auf `Ja` konfiguriert ist, werden alle Suchobjekte direkt aus Ihrem Inhaltsrepository abgerufen, anstatt den Asset Picker zu durchlaufen. In einem solchen Fall wird der `resource-loader`-Service nicht aufgerufen.

Den Asset-Auswahlrückruf-Service überprüfen.

Das Release 12.0 von Asset Picker unterstützt den `asset-selection-callback`-Service nicht.

Übersicht der Logger (Protokollfunktionen)

Wie unter Überprüfung der Integration (auf Seite 38) erwähnt, ist die Protokollierungskonfiguration für Asset Picker in den Dateien `log4j.xml` und `log4j2.xml` verfügbar, die sich im Ordner `AssetPicker/conf/logging` innerhalb von Platform home befinden.

Die Datei `log4j.xml` wird für die Loggers verwendet, die aus `unica_common.jar` und `unica_helper.jar` von Platform stammen. Auf der anderen Seite wird `log4j2.xml` für die Loggers verwendet, die von überall sonst in Asset Picker stammen.

Die Standardprotokollebene ist in beiden Fällen auf `WARN` eingestellt, was für die Fehlerbehebung bei der Plug-in-Entwicklung ausreichend sein sollte. Die meisten Loggers, die vom Asset Picker auf INFO & DEBUG-Ebene erzeugt werden, sind für die Entwicklung und Integration von Plug-ins nicht besonders relevant. Die folgenden Themen erläutern nur die relevanten Loggers. Diese Loggers sind bereits in der `log4j2.xml`-Datei vorhanden und müssen bei Bedarf unkommentiert bleiben. Bitte stellen Sie sicher, dass bei diesen Loggern in der Produktion die Protokollierungsstufe niemals auf `DEBUG` oder `TRACE` gesetzt ist, da sie sensible Informationen erzeugen können.

Nützliche Logger in der log4j2.xml-Datei

In der folgenden Tabelle sind die nützlichen Logger in der `log4j2.xml`-Datei aufgelistet:

Tabelle 2. Nützliche Logger in der log4j2.xml-Datei

Protokollfunktionen (Loggers)	Information
`org.springframework.web`	Die Einstellung dieses Loggers auf TRACE-Ebene erzeugt HTTP-Anforderungs- und Antwortdetails für alle eingehenden HTTP-Anforderungen an Asset Picker. Dieser Logger kann nützlich sein, wenn Sie verfolgen wollen, was zwischen Frontend und Backend ausgetauscht wird.
com.hcl.unica.cms.integration .flow.interceptor.logger	Dieser Logger ist besonders nützlich für die Entwicklung von Plug-ins. Er protokolliert die HTTP-Interaktion zwischen Asset Picker und

Protokollfunktionen (Loggers)	Information
	dem Zielrepository. Für jeden Service, der mit dem RESTful-Ansatz implementiert wurde (durch Implementierung von RestService, HTTPService oder deren spezialisierten Ableitungen), schreibt dieser Logger HTTP-Anfrage- und Antwortdetails für alle ausgehenden HTTP-Interaktionen mit dem Zielsystem. Um Sicherheitslücken zu vermeiden, werden Werte von vertraulichen Headern vor der Protokollierung maskiert. Nur die letzten vier Zeichen werden für die Fehlerbehebung unmaskiert gelassen. Solche Headers umfassen Standardheader-Autorisierung oder alle nicht standardmäßigen benutzerdefinierten Header, die in der Anfrage gesetzt oder als Antwort empfangen wurden.
`org.springframework.retry`	Wenn dieser Logger auf `TRACE`-Ebene eingestellt wird, werden Informationen über Wiederholungsversuche hinzugefügt, während HTTP-Aufrufe an das Zielrepository erfolgen. Dies ist nützlich, um die Wiederholungsrichtlinie zu überprüfen, die im Abschnitt QOS für das jeweilige System in der Platform-Konfiguration festgelegt wurde.

Andere wichtige Loggers

Andere wichtige Loggers sind bei der Fehlersuche in Asset Picker von Nutzen. Zusammen mit der Erkennung von Warnungen und Fehlern liefern diese Loggers Informationen, die aus funktioneller Sicht nützlich sind.

Die folgende Tabelle listet die anderen wichtigen Loggers auf:

- **Client-Anwendungen** - Wenn die Root-Logger-Ebene auf INFO-Ebene eingestellt ist, erfahren Sie in den folgenden Zeilen die Anzahl der Client-Anwendungen und welche Client-Anwendungen Asset Picker identifizieren kann:

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

- **CORS** - Wenn die Root-Logger-Ebene auf INFO-Ebene eingestellt ist, können die folgenden Zeilen Informationen über die Unterstützung von Asset Picker für die gemeinsame Nutzung von Ressourcen aus verschiedenen Quellen liefern:

```
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]+)?]}
```

- **Platform-Konfiguration** - Inhaltsrepositorys- Die Einstellung der Root-Logger-Ebene auf INFO informiert uns über die Inhaltsrepositorys, die von Asset Picker identifiziert werden.

```
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]}
```

- **Service-Metainformationsdateien** - Die folgenden Zeilen werden ebenfalls auf INFO-Ebene protokolliert und geben an, wie viele Service-Metainformationsdateien von Asset Picker identifiziert wurden:

```
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}...
```

- **Authentifizierungsprotokolle** - Die folgenden Zeilen, die auf INFO-Ebene protokolliert werden, bestätigen, dass das Authentifizierungsprotokoll für das jeweilige Inhaltsrepository identifiziert wurde:

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