Unica Content Integration V12.1 Entwicklerhandbuch

Contents

Chapter 1. Übersicht

Unica Content Integration erleichtert die einfache Integration mit Content Management Systemen und ermöglicht die Suche nach Inhalten aus ihnen.

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

Plug-ins

Für die Integration mit verschiedenen CMS verwendet Unica Content Integration REST APIs. Da jedes CMS über eine eindeutige Programmierschnittstelle verfügt, Unica Content Integration verwendet benutzerdefinierte Plugins oder Module, die speziell für das Ziel-CMS geschrieben wurden.

Sie können Plug-Ins mit der Programmiersprache Java implementieren. Unica Content Integration 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 Unica Content Integration, um gewünschte Inhalte aus externen Inhaltsspeichern abzurufen.

Integrationsunterstützung und Ansatz zur Plug-in-Entwicklung

Unica Content Integration 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 Content Integration Framework 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

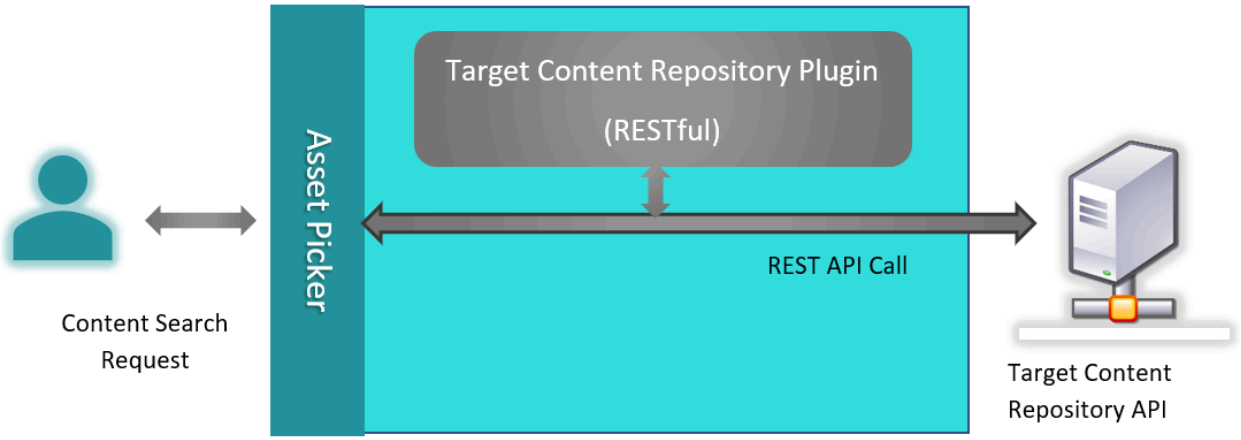
Ein alternativer Ansatz zur Entwicklung von Plug-ins für eine Nicht-RESTful-Integration erfordert eine gründliche Implementierung. Zum Beispiel muss ein Plug-in, das für das Abrufen von Inhalten aus der Datenbank geschrieben wurde, alles angehen, was mit der Herstellung der DB-Verbindung, der Ausführung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

Plug-Ins initiieren die Inhaltssuche nicht. Content Integration Framework empfängt zunächst die Suchanfrage, die an das jeweilige Plug-In delegiert wird. Im Falle von RESTful-Integrationen leitet Content Integration Framework 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:

Figure 1. Suchablauf für RESTful-Inhalte

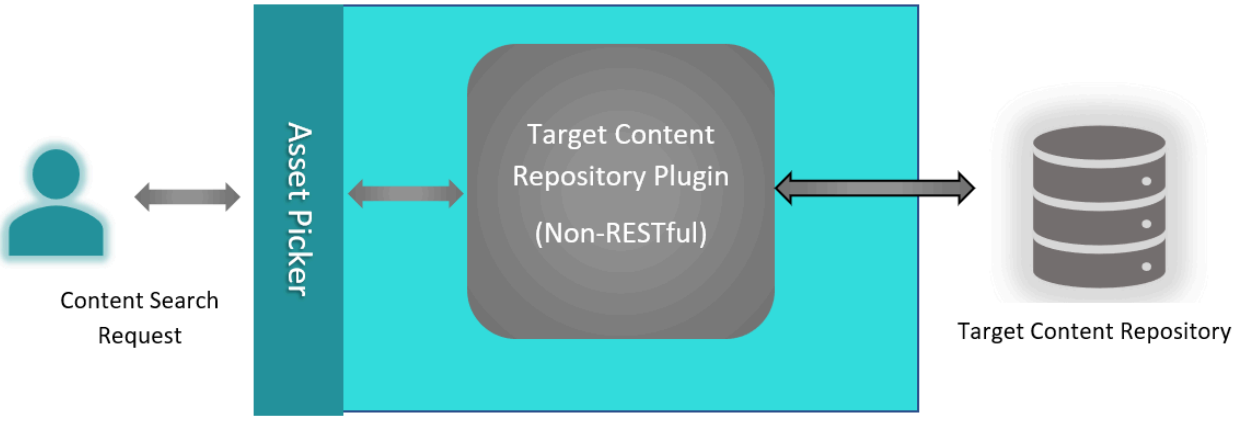


Wenn Content Integration Framework eine Inhaltssuchanforderung vom Benutzer für das Zielsystem erhält, ruft er das entsprechende Plug-in auf, um anforderungsspezifische logische Informationen zu erfassen, und führt einen API-Aufruf an das Zielsystem aus. Es ruft noch einmal das Plug-in auf, um die API-Antwort 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:

Figure 2. Suchablauf für Nicht-RESTful-Inhalte



Nicht-RESTful-Plug-ins interagieren mit dem Inhaltsrepository und liefern die Suchergebnisse an Content Integration Framework. Im Gegensatz zu RESTful-Repositorys kennt Content Integration Framework nicht den Typ, die Architektur, das Protokoll und

den Authentifizierungsmechanismus, der für die Kommunikation mit dem Zielrepository verwendet wird.

Chapter 2. Überblick über die Plug-in-Entwicklung

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

Unica Content Integration lässt sich nahtlos mit systemspezifischen, unabhängigen Plug-ins integrieren. Sobald das Plug-in entwickelt und in das `<ASSET_PICKER_HOME>/plugins/custom`-Verzeichnis auf dem Anwendungsserver, auf dem Content Integrationgehostet wird, abgelegt wurde, kann das entsprechende Inhaltsrepository in die Unica-Produktsuite integriert werden, indem einige Konfigurationen in der Unica Platform aktualisiert werden. Weitere Informationen finden Sie im Administratorhandbuch von Unica Content Integration

Note: *<ASSET_PICKER_HOME>* Bezieht sich auf das Basisinstallationsverzeichnis von Unica Content Integration in Platform-Startseite. Daher sollte jede weitere Verwendung von *<ASSET_PICKER_HOME>* in diesem Handbuch als Pfad zu dem Content Integration-Verzeichnis innerhalb der Platform-Startseite betrachtet werden.

Unica Content Integration wird mit einem Entwicklungskit bereitgestellt, das die Abhängigkeiten, Referenzprojekte und ein Starterprojekt zum Schnellstart der Plug-in-Entwicklung enthält. Das Entwicklungskit befindet sich im `<ASSET_PICKER_HOME>/dev-kits`-Verzeichnis. Vier Referenzprojekte namens `aem-integration`, `wcm-integration`, `dx-integration` und `commerce-integration` stehen jeweils für Adobe Experience Manager (AEM), IBM Web Content Manager (WCM), HCL Digital Experience und HCL Commerce zur Verfügung.

Komponenten eines Plug-ins

Ein typisches Plug-in enthält die folgenden Komponenten:

- Servicedeklarationen *(on page 6)*
- Serviceimplementierungen *(on page 20)*

Der Begriff Service stellt für eine Java-Klasse dar, 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.

Jeder Service, der vom Plug-in implementiert wird, muss in einer zentral verwalteten Servicedeklarationsdatei deklariert werden. Eine Servicedeklarationsdatei ist eine YML-Konfigurationsdatei, die die Liste der von allen verfügbaren Plug-ins implementierten Services enthält. Die Dienstdeklarationsdatei muss den Namen `custom-plugin-services.yml` haben. Sie sollte innerhalb des Verzeichnisses `<ASSET_PICKER_HOME>/conf` verfügbar sein. Die Struktur der `custom-plugin-services.yml`-Datei muss der `plugin-services.yml`-Datei ähnlich sein, die sich im selben Verzeichnis befindet. Die `plugin-services.yml` Datei enthält Servicedeklarationen für out-of-the-box-Systemintegrationen. Ein Service kann entweder ein Standardservice oder ein benutzerdefinierter Service sein.

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

Servicedeklarationen

Referenzservicedeklarationen finden Sie im `asset-integration-starter`-Projekt innerhalb des `dev-kits\asset-integration-starter\src\main\resources\META-INF`-Verzeichnisses.

Im folgenden finden Sie Beispielservicedeklarationen aus dem `asset-integration-starter`-Projekt:

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

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

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

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

Servicedeklarationsdatei

Die Servicedeklarationsdatei enthält das `services`- Element, das eine Reihe von einzelnen Servicedeklarationen darstellt. Eine Servicedeklaration ist ein Wörterbuch, das drei

obligatorische Elemente mit dem Namen, `systemId`, `serviceName` und `factoryClass` und ein optionales Element mit dem Namen `params`enthält. Einzelheiten zu den Elementen sind nachstehend aufgeführt:

- `systemId`

  Dieser Zeichenkettenwert identifiziert ein Ziel-Inhaltsrepository eindeutig. Dieser Bezeichner sollte vorzugsweise nur englische alphanumerische Zeichen enthalten. Punkte, Striche und Unterstriche können zur Verbesserung der Lesbarkeit verwendet werden. Vermeiden Sie alle anderen Sonderzeichen und Unicode- Zeichen. 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
  DX-CORE
  DX
  ```

  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. Content Integration Frameworkerlegt keine Einschränkungen auf.

- `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`
- `factoryClass`

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

  Bietet eine Möglichkeit zur Bereitstellung von statischen Parametern an den Service, um das Serviceverhalten entsprechend den Parameterwerten zu steuern oder zu ändern. Kurz gesagt, `params` kann verwendet werden, um die statische Schlüsselwertkonfiguration für Dienstimplementierungen zu speichern. Dies kann bestimmte Standardserviceparameter sowie alle benutzerdefinierten Parameter enthalten, die ein Service möglicherweise verwenden möchte. Parameterwerte werden in die Objekt der nächsten übereinstimmenden primitiven Wrapperklassen konvertiert, z. B. Ganzzahl, Long, Double, String usw. Ein Parameterwert kann auch eine Karte, ein Array oder eine Auflistung anderer Werte sein (Plugins müssen den Laufzeittyp dieser Werte überprüfen, bevor sie verwendet werden).

Die Dienstdeklarationsdatei enthält auch bestimmte Eigenschaften, die sich auf das Zielinhalts-Repository beziehen. Diese Eigenschaften werden unter dem Stammelement des Systems behandelt. Es folgt ein Beispiel für einen solchen Eintrag, der alle unterstützten Eigenschaften enthält:

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

Dieser Beispieleintrag zeigt die Standardwerte, die für jede hier erwähnte Eigenschaft berücksichtigt werden, falls für das angegebene Ziel-Repository kein solcher Eintrag vorhanden ist. Daher ist dieser Eintrag optional, es sei denn, eine oder mehrere dieser Standardüberlegungen gelten nicht für das Zielinhalts-Repository. Im folgenden Abschnitt wird die Bedeutung der einzelnen Eigenschaften beschrieben:

`params` - Bietet eine Möglichkeit, dem jeweiligen Plugin statische Parameter bereitzustellen, um das Plugin-Verhalten gemäß den Parameterwerten zu steuern oder zu ändern. Kurz gesagt, Params kann verwendet werden, um die statische Schlüsselwertkonfiguration für Plugin-Implementierungen zu speichern. Dies kann vordefinierte Standardsystemparameter sowie alle benutzerdefinierten Parameter enthalten, die ein jeweiliges Plugin möglicherweise verwenden möchte. Parameterwerte werden in die Objekt der nächsten übereinstimmenden primitiven Wrapperklassen konvertiert, z. B. Ganzzahl, Long, Double,

String usw. Ein Parameterwert kann auch eine Karte, ein Array oder eine Auflistung anderer Werte sein (Plugins müssen den Laufzeittyp dieser Werte überprüfen, bevor sie verwendet werden).

`additionalFeatures | securityPolicy` - Diese Einstellung muss auf 'true' gesetzt werden, wenn Inhalte innerhalb des jeweiligen Systems unter Verwendung der Sicherheitsrichtlinien von Unica geschützt werden.

`additionalFeatures | content | paginatedSearch` - Dieses Funktionsflag gibt an, ob Repository-Suchergebnisse für Seiteninhalte unterstützt werden oder nicht. Die Benutzererfahrung wird entsprechend geändert, um das Inhaltssuchergebnis anzuzeigen.

`additionalFeatures | content | paginatedList` - Dieses Funktionsflag gibt an, ob das Inhalts-Repository die Auflistung von paginierten Inhalten unterstützt oder nicht. Die Benutzererfahrung wird für das Anzeigen von Inhaltslisten entsprechend geändert.

`additionalFeatures | content | anonymousContent` - Mit diesem Funktionsflag wird angegeben, ob öffentlich zugängliche Inhalte vom Inhalts-Repository erwartet werden sollen oder nicht. Wenn es auf true gesetzt ist, muss das Plugin für jeden Inhalt eine öffentlich zugängliche URL zurückgeben. Wenn Inhalte nicht über die HTTP(S)-URL öffentlich zugänglich gemacht werden können, muss der Plugin-Entwickler diese Flag auf false setzen. In diesem Fall können Benutzer die aus dem Repository abgerufenen Inhalte nicht sehen oder herunterladen. Wenn das Zielsystem für den Inhalt keine anonym zugänglichen URL, müssen Sie den Service ausführen, um das Herunterladen geschützter `resource-loader` Inhalte zu ermöglichen.

Standardservices

Die folgende Tabelle enthält eine Einführung in die Standarddienste von Unica Content Integration. Daher sollte keiner der hier aufgeführten Dienstnamen für eine benutzerdefinierte Dienstimplementierung verwendet werden. Inhaltsintegration SDK bietet Standardschnittstellen und -typen zur Implementierung dieser Standarddienste. Diese Schnittstellen und Typen werden in den nachfolgenden Abschnitten näher beschrieben.

Table 1. Standardservices und deren Beschreibung

Name des Standardservice	Beschreibung
`simple-search`	Der Service „Einfache Suche" antwortet auf die Suchanfragen nach Inhalten, die von Content Integration Framework 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.
`list-folders`	Dies ist ein optionaler Dienst. Ordner ist ein allgemeiner Begriff zur Darstellung eines Containerobjekts, das im Zielsystem zur hierarchischen Organisation der Inhalte verwendet wird. Dieser Dienst wird aufgerufen, um die Liste der Ordner und Unterordner zu rendern und die Navigation durch solche hierarchisch organisierten Inhalte zu erleichtern. **Note:** `list-folders` und `list-contents` sind korrelierte Dienste. Die Implementierung für beide Dienste muss vorhanden sein, damit die Inhaltsnavigation ordnungsgemäß funktioniert.
`list-contents`	Dies ist ein optionaler Dienst. Dieser Dienst wird aufgerufen, um die Inhalte auflisten zu können, die zu einem bestimmten Ordner gehören.

Name des Standardservice	Beschreibung
	Note: `list-folders` und `list-contents` sind korrelierte Dienste. Die Implementierung für beide Dienste muss vorhanden sein, damit die Inhaltsnavigation ordnungsgemäß funktioniert.
`get-content-details`	Die Implementierung dieses Dienstes ist beim Abrufen der Details eines einzelnen Inhalts hilfreich. Auf Inhalte, die mit `simple-search` und `list-contents services` eingeholt wurden, wird in anderen Unica-Produkten weiter verwiesen. Benutzer möchten zu einem späteren Zeitpunkt möglicherweise die Details zu bereits referenzierten Inhalten anzeigen. Daher empfehlen wir die Implementierung dieses Dienstes, damit Benutzer die Inhaltsdetails nach Bedarf sehen können.
`get-object-schema`	Dies ist ein optionaler Dienst. Die Implementierung dieses Dienstes ist nützlich, damit Benutzer des Centralized Offer Management Inhaltsattribute Angebotsattributen zuordnen können. Und anschließend Werte für zugeordnete Angebotsattribute aus den entsprechenden Inhaltsattributen abzuleiten, indem der gewünschte Inhalt aus der Inhaltsauswahl ausgewählt wird. Wenn dieser Dienst implementiert ist, erleichtert er die Verwendung anderer Inhaltsattribute zusätzlich zur Inhalts-URL für die Angebotserstellung.

Name des Standardservice	Beschreibung
`resource-loader`	Dieser Dienst ist nützlich, wenn ein direkter Download des Inhalts vom Zielsystem nicht möglich ist. Dieser Service ist nicht obligatorisch und sollte nur implementiert werden, wenn folgende Hindernisse auftreten: • **Falls kein direkter Web-Link vorhanden ist, um den Inhalt herunterzuladen** Von den `simple-search`- und `list-contents`-Diensten zurückgegebene Inhalte müssen eine absolute URL zum jeweiligen Inhalt enthalten, damit der Content Integration-Client ihn direkt über das Web herunterladen kann. Wenn kein solcher direkter Weblink zum Inhalt vorhanden ist, muss der `resource-loader`-Dienst implementiert werden, indem die vom Content Integration Framework bereitgestellte Standardimplementierung überschrieben wird, damit der Content Integration-Client ihn direkt über das Web herunterladen kann. Wenn der Inhalt beispielsweise in einer Datenbanktabelle verwaltet wird, rufen die `simple-search`- und `list-contents`-Dienste Datensätze aus der Datenbank ab. Da die Elemente aus der Datenbank geladen werden, gibt es möglicherweise keine URL, die direkt auf jeden Datensatz verweist. In diesem

Name des Standardservice	Beschreibung
	Fall kann der `resource-loader`-Dienst die Inhaltskennung verwenden, um die entsprechenden Daten zu finden und bereitzustellen, wenn der Download von Inhalten angefordert wird. Alle Anforderungen zum Herunterladen von Inhalten werden über das Content Integration Framework ausgeführt, das die Download-Aufgabe an den `resource-loader`-Dienst delegiert, indem es ihm die Inhalts-URL und seine Kennung bereitstellt. • **Wenn Web-Links zu den Inhalten geschützt sind** Bestimmte Systeme bieten möglicherweise keinen anonymen Zugriff auf die Inhalte, obwohl direkte Weblinks verfügbar sind. In solchen Fällen wird der Zugriff in der Regel erst nach der Bereitstellung der erforderlichen Authentifizierung bereitgestellt. Standardmäßig registriert Content Integration Framework eine standardmäßige Implementierung des `resource-loader`-Dienstes für jedes Plug-in. Diese Standardimplementierung verwendet die URL für den tatsächlichen Inhalt, um den Inhalt vom Remote-System herunterzuladen, indem entsprechende Authentifizierungsdetails bereitgestellt werden, die den

Name des Standardservice	Beschreibung
	Konfigurationen in Unica Platform unterliegen. (Weitere Informationen zu System-Onboarding-Konfigurationen finden Sie im Unica Content Integration-Administratorhandbuch). Alternativ können Plugins die Standardimplementierung des Ressourcenladers überschreiben, um das Verhalten beim Herunterladen von Inhalten zu ändern (mithilfe der Inhalts-URL oder der Inhalts-ID). Wenn der `resource-loader`-Dienst mithilfe des RESTful-Ansatzes überschrieben wird, kümmert sich Content Integration Framework weiterhin um die Bereitstellung von Authentifizierungsdetails basierend auf der Plattformkonfiguration. **Note:** Inhalte müssen anonym zugänglich gemacht werden, wenn erwartet wird, dass sie von einem externen Publikum gesehen/abgerufen werden. In diesem Fall wird die Verwendung des Ressourcenladerdienstes in Produktionssystemen nicht empfohlen. Die Verwendung des `resource-loader`-Dienstes kann jederzeit deaktiviert werden, indem die Eigenschaft Anonymer Inhalt in der

Name des Standardservice	Beschreibung
	Plattformkonfiguration auf `Ja` gesetzt wird. Ebenso kann es aktiviert werden, indem dieselbe Eigenschaft auf `Nein` gesetzt wird.
`list-content-categories`	Inhalte können durch ihre natürliche Klassifizierung logisch kategorisiert werden. Beispielsweise können digitale Inhalte in Bilder, Dokumente, Multimedia (Audios und Videos), Archive usw. eingeteilt werden. In ähnlicher Weise können E-Commerce-Produkte in mehrere große Kategorien eingeteilt werden, wie z. B. Elektronik, Gesundheitswesen, Bücher, Möbel usw. Content Integration Framework ermöglicht die folgenden Möglichkeiten zur Übertragung einer solchen Inhaltskategorisierung, um die Suche von Inhalten innerhalb bestimmter Kategorien zu erleichtern. • **`supportedContentTypes` Dienstparameter** Ein Standardparameter für Service-Level, kann verwendet werden, um statisch ein Wörterverzeichnis der unterstützten Inhaltstypen `supportedContentTypes` unter Servicedeklaration `simple-search` zu liefern. • **`getSupportedContentTypes()`Methode in der Implementierung des Suchdienstes**

Name des Standardservice	Beschreibung
	Die `getSupportedContentTypes()`-Methode kann überschrieben werden, um dynamisch eine Karte der unterstützten Inhaltstypen zu generieren, wobei der Schlüssel als Kategoriekennung und der Wert für die auf der Benutzeroberfläche angezeigte Bezeichnung dient. Diese Methode wird während des Anwendungsstarts ausgeführt. Daher kann mit den Funktionen von Content Integration Framework kein Remote-API-Aufruf durchgeführt werden, da sich die Anwendung beim Aufrufen dieser Methode möglicherweise in einem teilweise initialisierten Zustand befindet. • **`list-content-categories` Service** Optional kann ein Dienst list-content-categories implementiert werden, um die Einschränkung der `getSupportedContentTypes()`-Methode zu beheben. Es ermöglicht Remote-API-Aufrufe, um die Inhaltskategorien noch dynamischer abzurufen. Wenn implementiert, überschreibt dieser Dienst die zuvor genannten Ansätze. Content Integration Framework ruft diesen Dienst immer dann auf, wenn ein Popup für die Inhaltssuche gerendert wird.
`get-cognitive-analysis`	Dies ist ein optionaler Dienst. Wenn implementiert, wird es verwendet, um

Name des Standardservice	Beschreibung
	kognitive Details abzurufen, die dem angegebenen Bild zugeordnet sind, vorbehaltlich der Konfiguration "Bevorzugter kognitiver Dienstanbieter" in Unica Platform.

Serviceimplementierungen

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

Der Content Integration Framework bietet ein SDK zur Rationalisierung der Serviceimplementierung und erleichtert die schnelle Entwicklung von Plug-ins. Das Content Integration 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 Content Integration SDK vorgestellt. Weitere Informationen finden Sie im Abschnitt Plug-in-Entwicklung SDK (on page 27).

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 Content Integration 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

Typenparameter für die `ExecutionContext`-Klasse stellt die Art der Eingabe dar, die für den entsprechenden Service für seinen Aufruf erforderlich ist.

- **HttpRequest buildRequest(ExecutionContext<RQ> executionContext)**
  Dies ist eine obligatorische Methode. Sie gibt ein Objekt vom Typ `com.hcl.unica.cms.model.request.HttpRequest` zurück. Die `HttpRequest`-Klasse stellt Builder API bereit, um das Objekt mit den anwendbaren Details zu erstellen. Dieses Objekt enthält alle erforderlichen Details für die Erstellung einer HTTP, z. B. Endpunkt URL, http, HTTP Header und HTTP-Anfragetext. Die `HttpRequest` Builder-API akzeptiert die folgenden Argumente:
  - **Zeichenfolge endpointUrl**

    Eine absolute URL zur Ziel-API.
  - **HttpMethod httpMethod**

    HTTP-Methode für die Erstellung von API-Aufrufen. Es muss sich um einen der Werte von `com.hcl.unica.system.integration.service.HttpMethod` enum handeln.
  - **Optional<Map<Zeichenfolge, Objekt>> Header**

    Eine optionale Map von HTTP-Headern. Sie kann sowohl Standard- als auch benutzerdefinierte HTTP Header enthalten. Headernamen müssen in Form von Map-Schlüsseln angegeben werden, und Headerwerte müssen als entsprechende Werte in der Map angegeben werden. In Ermangelung dieser dieses optionalen Wertes werden keine benutzerdefinierten Header zusammen mit der ausgehenden HTTP-Anfrage gesendet.

    Note: Obwohl die Header-Map Werte vom Typ Objekt (oder seiner Subtypen) akzeptiert, werden ab der aktuellen Implementierung von Content Integration Framework 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.
    ```

- **Optional<?> Nutzdaten**

  Wenn der Zielservice irgendeinen Anfragekörper erwartet, dann kann dieses Argument mit dem gewünschten HTTP-Anfragekörper geliefert werden. Es kann sich um ein beliebtiges Objekt handelt, solange der entsprechende `Content-Type`-Header in der Header-Map angegeben ist. In Ermangelung dieses Arguments wird ein leerer Anfragekörper zusammen mit der ausgehenden HTTP-Anfrage gesendet.

  Note: Jackson- und JAXB-Unterstützung: Die Objektserialisierung mit Jackson und JAXB wird von Content Integration Framework vollständig unterstützt. So kann ein entsprechend dekoriertes Objekt mit Jackson- oder JAXB-Annotationen als Nutzdaten der Anfrage festgelegt werden. In diesem Fall muss der entsprechende `Content-Type`-Header in der Header-Map angegeben werden. Die Serialisierung des gelieferten Objekts in den Anfragekörper wird ebenfalls von Content Integration Frameworkselbst vorgenommen, daher ist keine explizite Serialisierung erforderlich.

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

  Diese optionale Methode wandelt die HTTP-Antwort in ein gewünschtes Format um. Das erste Argument, `com.hcl.unica.system.model.response.HttpResponse`, für diese Methode stellt die Antwort dar, die vom Zielsystem empfangen wurde. Der generische Typenparameter der `HttpResponse`-Klasse stellt den Typ des Antwortkörpers oder der Antwortnutzdaten dar, der von der fernen API erwartet wird. Die Antwortnutzdaten können beliebigen Typs sein, z. B. eine Zeichenkette, die den gesamten vom Service empfangenen Text enthält, ein Byte-Array, das den Antwortkörper enthält, oder ein deserialisierter POJO, der die Antwort JSON/XML darstellt. Zusätzlich zu den Antwortnutzdaten kann das `HttpResponse`-Objekt zum Abrufen von Antwortheader, Statuscode und Cookies eingesetzt werden.

  Note: Jackson- und JAXB-Unterstützung: Die Objektdeserialisierung mit Jackson und JAXB wird von Content Integration Framework 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 Content Integration Framework

durchgeführt, daher ist bei der Antwortumwandlung innerhalb dieser Methode keine explizite Deserialisierung erforderlich.

In Ermangelung dieser Implementierung wird von Content Integration Framework keine implizite Transformation durchgeführt.

Zusätzlich zu diesen Methoden gibt es noch eine weitere Methode, die `getServiceInterface` von `com.hcl.unica.system.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.

Content Integration Framework 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 Content Integration Framework 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`

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 Content Integration Framework 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 Typenparameter für die `ExecutionContext`-Klasse stellt die Art der Eingabe dar, die für den entsprechenden Service für seinen Aufruf erforderlich ist.

- **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 `FunctionalService`-Schnittstelle verwendeten Typ.

Fehlerbehandlung

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

Common methods

Im Folgenden sind die gängigen Verfahren aufgeführt, die sowohl für RESTful- als auch für funktionale Services anwendbar sind. Diese Methode werden von der `com.hcl.unica.system.integration.service.AbstractService`-Schnittstelle übernommen.

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

  Ihre Implementierung ist jedoch eher für den Serviceaufruf als für die Implementierung des Service relevant. Weitere Informationen finden Sie in Plug-in-Entwicklung SDK (on page 27).
- **void init(SystemConfig systemConfig, ServiceConfig serviceConfig)**

  Überschreiben Sie diese optionale Methode, um eine einmalige Initialisierung (nach der Installation des Serviceobjekts) durchzuführen, bevor Sie eine beliebige Anforderungen bedienen. Verwenden Sie das Objekt ' systemConfig ' und das ServiceConfig-Objekt, das an diese Methode übergeben wird, um System- und servicespezifische Details zu erhalten bzw. um erforderliche Initialisierungen zu ermöglichen, wie z. B. eine Datenbankverbindung zu erhalten, ein Datei-Handle

zu öffnen usw. Für jede einzelne Systemkonfiguration in Unica Platform wird ein separates Objekt ihrer Serviceklasse erstellt. Wenn also dasselbe Zielsystem für zwei verschiedene Partitionen in Unica Centralized Offer Management konfiguriert ist, werden für jede Partition zwei verschiedene Objekt ihrer Serviceklasse erstellt. Auch wenn dasselbe Zielsystem für ein anderes Unica-Produkt konfiguriert ist, ist ein separates Objekt für diese Konfiguration vorhanden. Das `com.hcl.unica.system.integration.config.SystemConfig`-Objekt verkettet alle Systemkonfigurationen, die im Abschnitt Unica Platform Konfiguration vorgenommen werden, während das `com.hcl.unica.system.integration.config.ServiceConfig`-Objekt alle für den entsprechenden Dienst vorgenommenen Konfigurationen in den `<ASSET_PICKER_HOME>/conf/plugin-services.yml`- und `<ASSET_PICKER_HOME>/conf/custom-plugin-services.yml`-Dateien enthält . Diese Objekte können auch in allen zuvor besprochenen Methoden über `ExecutionContext` aufgerufen werden.

Note: Content Integration Framework bietet keine spezielle End-of-Lifecycle-Methode für Dienste zur Bereinigung der Dinge, die innerhalb Methode ' init ' initialisiert wurden. Wir empfehlen Ihnen, den standardmäßigen Java-Ansatz zu verwenden, indem Sie gegebenenfalls die Methode ' Finalize ' umsetzen.

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 Content Integration Framework

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 Content Integration Framework
- Wenn dies erforderlich ist, kann die explizite Implementierung fehlender inaktiverer Funktionen die Serviceimplementierungen sehr verbogen.

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 Content Integration Framework 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.

Chapter 3. Plug-in-Entwicklung SDK

Dieses Thema liefert Informationen über die verschiedenen Klassen, Schnittstellen und Aufzählungen aus demContent IntegrationSDK mithilfe von entsprechenden logischen Einheiten in `asset-integration-starter`-, `-aem-integration` und `wcm-integration`-Referenzprojekten, die als Teil des Development Kits zusammen mit der Content Integration-Funktion eingebettet sind.

Content Integration SDK für Plug-in-Entwicklung befindet sich im `<ASSET_PICKER_HOME>/dev-kits/sdk/`-Verzeichnis auf Anwendungsserver. Die folgenden JAR-Dateien befinden sich im `sdk`-Verzeichnis:

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

Diese JARs enthalten alle SDL-Klassen, Schnittstellen & enums, die in diesem Abschnitt diskutiert werden. Schauen Sie sich die relevanten Klassen aus diesen JARs an, wenn Sie auf das jeweilige Thema in diesem Handbuch stoßen.

Generische Typparameter

Generische Typparameter werden für die Implementierung von Serviceschnittstellen verwendet. Weitere Informationen zu Serviceschnittstellen finden Sie unter Serviceimplementierungen (on page 20).

Ein in einem Plug-in befindlicher Service ist lediglich eine Programmiereinheit, die einige Inputs entgegennimmt und den erwarteten Output zurückgibt. Ähnlich fragt die REST-API, die von unserem Service umhüllt wird, nach der erforderlichen Eingabe (Anforderungstext, Header, Cookies und Abfrageparameter) und erzeugt die gewünschte Antwort (Antworttext, Header und Cookies). Es erfordert bestimmte generische Notationen für die während des logischen End-to-End-Ablaufs ausgetauschten In- und Outputs.

Content Integration Framework verwendet den Parameter Typ RQ, um den Typ der Eingabe zu kennzeichnen, die dem Service bei seinem Aufruf zur Verfügung gestellt wird. Hier wird der Parameter vom Typ RS verwendet, um entweder den Typ des Objekts zu bezeichnen, das vom Funktionsservice zurückgegeben wurde, oder den Typ des Antwortkörpers, der von der entfernten REST-API zurückgegeben wird, die mit dem RESTful-Ansatz aufgerufen wurde. Der Zweck von RS kann sich je nach Verwendungszweck ändern, aber er gibt immer den Rückgabewert von "etwas" an.

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

- **RS**

  Dieser Parametertyp 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 Typargument an die `HttpResponse`Klasse geliefert wird, was dem Typparameter RS der `RestService`-Schnittstelle entspricht.

Note: 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 `FunctionalService` 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. Die `getServiceInterface`-Methode muss das Klassenobjekt der abgeleiteten (untergeordneten Schnittstelle) von `ServiceGateway`zurückgeben . Die Definition von

`com.hcl.unica.system.integration.service.gateway.ServiceGateway` sieht wie folgt aus:

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

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.

Note:

- 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.
- Durch den Verweis auf die `ServiceGateway`-Instanz eines auf diese Weise implementierten Service kann die Methode "Execution" (RQ-Anfrage) aufgerufen werden, um den Service auszuführen. Beachten Sie, dass die Ausführungsmethode das `ServiceExcecutionException` auslösen kann, wenn bei der Serviceausführung etwas schief geht. Details zur Handhabung von Serviceaufrufen und Ausnahmen werden in den folgenden Themen behandelt.

Serviceaufruf

Das `asset-integration-starter`-Projekt enthält eine `com.example.service.client.CustomServiceClient`-Klasse zum illustrieren des Serviceaufrufs.

Die `CustomServiceClient`-Klasse erhält einen Verweis auf das `SystemGateway`-Objekt für das System, das durch eine Kennung *Foo* durch Aufrufen der `SystemGatewayFactory.getSystemGateway`-Methode mit `Foo` als Argument dargestellt wird. `SystemGatewayFactory.getSystemGateway`-Methode gibt somit jedem Zielsystem einen Handle, indem sie den *systemId* angibt. Sobald der Handle in Bezug auf das `SystemGateway`-Objekt abgerufen wurde, kann er zum Aufrufen von Service auf dem jeweiligen Zielsystem verwendet werden. Im folgenden finden Sie ein entsprechendes Codeausschnitt aus der `CustomServiceClient`-Klasse:

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

SystemGateway

Die `com.hcl.unica.system.integration.service.gateway.SystemGateway` stellt eine überladene Methode `executeService` zur Verfügung, um einen beliebigen Dienst auf dem Zielsystem auszuführen. Eine Version dieser Methode bietet die Möglichkeit, jeden in Dienstdeklarationsdateien (`<ASSET_PICKER_HOME>/conf/custom-plugin-services.yml` und `<ASSET_PICKER_HOME>/conf/plugin-services.yml`) deklarierten Dienst für das jeweilige System auszuführen. Die andere Version bietet die Möglichkeit, einen Ad-hoc-HTTP-Aufruf auf dem Zielsystem auszuführen, ohne einen expliziten Dienst für diesen in der Dienstdeklarationsdatei zu deklarieren. Im folgenden sind die beiden Versionen der `executeService`-Methode mit ihren Signaturen zu sehen:

- **<RQ, RS> RS executeService(String serviceName, RQ serviceInput, Class<? erweitert ServiceGateway<RQ, RS>> gatewayClass) löst ServiceExecutionException aus**
  Dies ist eine generische Methode, die mit den Typparametern RQ & RS arbeitet. Die Bedeutung von RQ & RS ist identisch mit der zuvor erwähnten. Diese Methode hilft bei der Ausführung eines bereits deklarierten Dienstes. Die `invocationDemo`-Methode in

der `CustomServiceClient`-Klasse demonstriert die Verwendung dieser Methode. Sie akzeptiert die folgenden Argumente:

- **Zeichenkette serviceName**

  Dies muss der Name des Service sein, der ausgeführt werden soll. Der Name des Dienstes muss genau mit der entsprechenden Deklaration in der Dienstdeklarationsdatei übereinstimmen.
- **RQ serviceInput**

  Dies ist eine Eingabe für den Service, der ausgeführt wird. Der Typparameter des Typs RQ stellt den Typ der Eingabe dar, die für den aufgerufenen Service erforderlich ist
- **Class<? extends ServiceGateway<RQ, RS>> gatewayClass**

  Er muss mit dem Rückgabewert der `getServiceInterface`-Methode in der entsprechenden Service-Implementierung übereinstimmen. Er hilft Content Integration Framework bei der Ermittlung der richtigen Eingabe für den Service, der ausgeführt wird, und gibt die gewünschte Ausgabe zurück. Die Parameter RQ und RS, die für das `gatewayClass`-Argument verwendet werden, stellen den Typ der Eingabe dar, die für den Serviceaufruf und den Antworttyp bereitgestellt wird, der vom Service beim Abschluss zurückgegeben wurde.

Bei erfolgreichem Abschluss gibt diese Methode das Objekt des Typs zurück, das durch den Typparameter RS dargestellt wird. Daher regelt das dritte Argument für die `executeService`-Methode, `gatewayClass`, den Typ der Eingabe, der in den Service fließt, und den Wertentyp, den der Service zurückgibt.

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

  Dabei handelt es sich außerdem um eine generische Methode, bei der der Typparameter T den Typ der Antwort darstellt, die vom entfernten HTTP-Aufruf erwartet wird. Es hilft Ihnen dabei, eine schreibübergreifende HTTP auf das Zielsystem zu erstellen, ohne einen expliziten Service dafür in der Dienstdeklarations-Datei zu deklarieren. Die `adHocInvocationDemo`-Methode in der `CustomServiceClient`-Klasse demonstriert die Verwendung dieser Methode. Es werden die folgenden aufgelisteten Argumente akzeptiert:

- **HttpRequest-Anforderung**

  Dies muss ein Objekt der `com.hcl.unica.system.model.request.HttpRequest`-Klasse sein. `HttpRequest` stellt eine Builderschnittstelle für das Konstruieren des Objekts mit den erforderlichen Details bereit. Dieses Objekt enthält im Wesentlichen die Details, die für die Erstellung einer HTTP erforderlich sind, wie z. B. absolute URL, HTTP-Anforderungsmethode, HTTP- Anforderungsheader & HTTP-Anforderungstext oder HTTP-Anforderungsnutzdaten
- **Klasse<T> expectedResponse**

  Hier muss der Antworttyp angegeben werden, der von der fernen URL erwartet wird. Es können auch die Typen Jackson und JAXB eingesetzt werden. Die Deserialisierung von JSON/XML wird in diesem Fall automatisch erfolgen.

Nach erfolgreichem Abschluss gibt diese Methode das Objekt `com.hcl.unica.system.model.response.HttpResponse` zurück, wobei das Antwortobjekt aus dem fernen Aufruf gekapselt wird. Der Antworttyp, der von `HttpResponse` eingekapselt wird, ist mit dem `expectedResponse`-Argument für die `executeService`-Methode identisch. Das `HttpResponse`-Objekt ermöglicht den Zugriff auf die HTTP- Antwortstatuscode, Antwortheader und Antwortcookies , zusätzlich zu den Antwortnutzdaten.

Beide Versionen der `executeService`-Methode können die `com.hcl.unica.system.integration.exception.ServiceExecutionException` oder eine der zugehörigen Subtypen auswerfen, falls bei Serviceausführung etwas schief läuft. Das Objekt dieser Ausnahmebedingung kann für die unmittelbare Ursache für den Serviceausführung konsultiert werden. Auch wenn der aufgerufene Service einen REST/ HTTP-Service darstellt (z. B. Ad-hoc-Serviceaufrufe sind immer HTTP-Aufrufe) und der Fehler aus HTTP-Interaktion auftritt, kann auch ein optionales `HttpResponse`-Objekt aus der Ausnahme abgerufen werden. In solchen Fällen wird die `HttpServiceExecutionException`-Methode durch die `executeService`-Methoden ausgelöst. Die Anwesenheit von `HttpResponse` hängt davon ab, ob die HTTP-Interaktion eingetreten ist oder nicht. Die `HttpServiceExecutionException` kann aufgrund einer Ausnahmebedingung in einer beliebigen Logik empfangen werden, die vor dem tatsächlichen http aufgerufen wurde, z `buildRequest` . B. der Methode in einem deklarierten Service.

Die `executeService`-Methode kann auch einen `SystemNotFoundException` auslösen, wenn das Plug-in für das angegebene Zielsystem nicht vorhanden ist oder das entsprechende System nicht in Unica Platform integriert ist. Ebenso kann eine ServiceNotFoundException ausgelöst werden, wenn der angegebene Dienst entweder nicht in der Dienstdeklarationsdatei deklariert oder vom Plugin nicht implementiert ist.

Note:

- 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 Klasse`com.example.service.functional.CustomService` verwendet wird. Es handelt sich um denselben Ausgabetyp, der für die Definition der `CustomServiceGateway`-Schnittstelle verwendet wird, deren Klassenobjekt in beiden Versionen der `getServiceInterface`-Implementierungen von der `CustomService`-Methode zurückgegeben wird.
- Die Klasse `com.example.service.rest.CustomService` und die Klasse `com.example.service.functional.CustomService` stellen denselben Service dar, der mit zwei verschiedenen Ansätzen implementiert wurde. Die Service-Metainformationsdateien im `asset-integration-starter`-Projekt, nämlich die `META-INF/rest-content-services.yml` und die `META-INF/functional-content-services.yml`, 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 Content Integration Frameworknur 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

Aus der Perspektive der Serviceimplementierung, enthalten die `ExecutionContext`- und `SystemConfig`- Objekte, die an verschiedene Rückrufverfahren übergeben werden, die Clientanwendung und partitionsspezifische Informationen. Aus der Perspektive des Serviceaufrufs werden die mit der `executeService`-Methode ausgeführten Services aus der

`SystemGateway`-Klasse auf dem System ausgeführt, das für die richtige Clientanwendung und die Partition des Benutzers, der auf Unica Content Integration zugreift, konfiguriert ist. Daher müssen weder die Implementierung noch der Aufrufer explizit mit der Partitionierung und anderen kontextbezogenen Details arbeiten. Content Integration Framework verarbeitet sie automatisch.

Ausführungskontext

Nahezu jede Methode in einem Serviceimplementierungsvertrag erhält eine Instanz der `com.hcl.unica.system.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 die Eingabe oder Anforderung zu erhalten, die an den Service übergeben wird, wenn Sie mit der in `executeService` diskutierten Serviceaufruf (on page 31)-Methode 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 eine Map zurück, die zum Speichern und Abrufen von benutzerdefinierten Attributen während der Ausführung genutzt werden kann. Es ist hilfreich, Ausführung spezifischen, vorübergehenden Informationen über mehrere Callbacks hinweg zu transportieren. Wenn beispielsweise die Implementierung der `buildRequest` Methode von der `RestService`-Schnittstelle oder `HttpService`-Schnittstelle aus einige Informationen mit der `transformResponse`-Methode teilen muss, kann Sie diese mit dieser Attributs Map teilen.

  Es ist wichtig festzustellen, dass von Content Integration Framework für jeden einzelnen Serviceaufruf eine separate Instanz von `ExecutionContext`erstellt wird.

Kontextattribute können daher nicht über mehrere Serviceausführungen hinweg gemeinsam genutzt werden. Ihr Geltungsbereich ist auf einzelne Serviceausführung beschränkt.

- **ServiceConfig getServiceConfig()**

  Diese Methode gibt eine Instanz einer `com.hcl.unica.system.integration.config.ServiceConfig`-Klasse zurück. `ServiceConfig`-Objekt enthält die Konfigurationen, die in der Dienstdeklarations-Datei für den jeweiligen Service vorgenommen wurden.

- **SystemConfig getSystemConfig()**

  Diese Methode gibt eine Instanz der `com.hcl.unica.system.integration.config.SystemConfig` Klasse zurück. `SystemConfig`-Objekt enthält alle Konfigurationen, die in Unica Platform für das Zielsystem vorgenommen wurden. Im Falle von Konfigurationen mit mehreren Partitionen wird dieses Objekt von Content Integration Framework entsprechend gefüllt, um die partitionsspezifische Konfiguration für die betreffende Client-Anwendung aufzunehmen. Für Informationen zu den verschiedenen Systemkonfigurationseinstellungen in Unica Platform, siehe Unica Content IntegrationAdministratorhandbuch.

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

  Diese Methode kann für die Festlegung von Attributen in `ExecutionContext` verwendet werden, die dann in anderen Bereichen der Serviceimplementierung abgerufen werden können. Dies ist für die gemeinsame Nutzung benutzerdefiniertes Kontextinformationen während des Serviceausführung hilfreich. Der Umfang der im Ausführungskontext gespeicherten Attribute ist nur auf den aktuellen Ausführungsfluss beschränkt. Attribute können nicht von mehreren Ausführungsabläufen desselben Dienstes gemeinsam genutzt werden.

  - **Locale getUserLocale ()**

    Diese Methode kann verwendet werden, um das Locale angemeldeter Benutzer zu erhalten.

Benutzerdatenquelle

Unica Platform verwendet Benutzerdatenquellen zum Speichern vertraulicher Informationen wie API-Anmeldeinformationen, Sicherheitstoken, Datenbankbenutzeranmeldeinformationen usw. Plug-ins müssen solche Konfigurationsdetails häufig speichern. Content Integration bietet die entsprechende Konfiguration, um den Namen der Benutzerdatenquelle und den zugehörigen Unica-Benutzer anzugeben, während Systeme mit der Unica Platform-Konfiguration verwendet werden.

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

```
executionContext.getSystemConfig().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 `SystemConfig`-Objekt aufgerufen wird, ein Unica Token-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.

Das Content Integration Framework nutzt dieses Design und definiert bestimmte Standarddienstklassen für Dienste Einfache Suche (`simple-search`), Inhaltskategorienliste (`list-content-categories`), Listenordner (`list-folders`), Listeninhalte (`list-contents`), Inhaltsdetails aufrufen (`get-content-details`), Objektschema aufrufen (`get-object-schema`) und Kognitive Analyse abrufen (`get-cognitive-analysis`). Die im Rahmen des Content Integration SDK bereitgestellte Datei standard-integrations.jar bietet spezielle

Versionen von `RestService` und `FunctionalService` für jeden dieser Standarddienste, um deren Implementierung mithilfe des RESTful- oder Functional-Ansatzes zu erleichtern.

Anrufung von Standardservices

Sobald das Content Integration Framework in der Dienstdeklarationsdatei deklariert und entweder mit dem RESTful- oder dem funktionalen Ansatz implementiert wurde, ruft es die Standarddienste in folgenden Szenarien auf:

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

  Immer wenn Content Integration Framework von seiner Client-Anwendung gegen das Zielsystem Inhalte oder Asset-Suchanfragen erhält, ruft er den für das jeweilige System implementierten `simple-search`-Service auf. Content Integration Frameworkliefert den notwendigen Input für den `simple-search`-Service nach dem Aufruf. 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 Dienstdeklarations-Datei 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.

  Das von diesem Service erzeugte Suchergebnis kann paginiert oder unpaginiert sein. Das Vorhandensein oder Nichtvorhandensein von Unterstützung für paginierte Ergebnisse sollte mithilfe der Eigenschaft paginnedSearch im Abschnitt Systeme in der Dienstdeklarationsdatei, wie im Thema Servicedeklarationsdatei (on page 8) erläutert, eindeutig angegeben werden.
- **Ressourcenlader (`resource-loader`)**

  Der `resource-loader`-Service wird von Content Integration Framework nur dann ausgeführt, wenn ein indirekter (oder authentifizierter) Zugriff auf das Suchobjekt auf dem Zielsystem erforderlich ist. Die Konfiguration kann in Unica Platform erfolgen, um anzugeben, ob auf Inhalte direkt (anonym) vom Zielsystem aus zugegriffen werden kann oder nicht. Weitere Informationen zu Systemkonfigurationen finden

Sie im Unica Content Integration Administrationsleitfaden. Content Integration Framework stellt einen `resource-loader`-Standardservice für jedes System bereit. Der `resource-loader`-Standardservice lädt die Webressourcen einfach aus dem Zielsystem, indem er die erforderlichen Autorisierungdetails bereitstellt, sofern zutreffend. Plug-Ins können den `resource-loader`-Standardservice überschreiben und ihre eigene Implementierung einschließen, indem sie die gebrauchsfertige Implementierung erweitern. Das Herunterladen von Inhalten und die Wiedergabe von Inhalten können fehlschlagen, wenn die erforderliche überschriebene `resource-loader`-Implementierung fehlt

- **Inhaltskategorien auflisten (`list-content-categories`)**

  Falls dieser Dienst implementiert ist, wird er zum Abrufen der Liste der unterstützten Inhaltskategorien aufgerufen, die schließlich zum Auffüllen der Dropdown-Liste des Inhaltstyps auf der Benutzeroberfläche der Inhaltsauswahl verwendet werden. Diese Kategorien werden verwendet, um die Inhaltssuche innerhalb einer bestimmten Kategorie einzugrenzen. In zukünftigen Versionen von Unica Content Integration kann es andere Anwendungsfälle geben, die sich auf diese Kategorien beziehen.

  Dies ist ein optionaler Dienst, und das Fehlen seiner Implementierung hat keinen Einfluss auf die Suchbarkeit von Inhalten in Content Picker. Andere Alternativen werden stattdessen verwendet, um die Liste der unterstützten Inhaltskategorien zu generieren, wenn dieser Dienst nicht vorhanden ist. Dies ist der `supportedContentTypes`-Standardparameter für den `simple-search`-Dienst in der Dienstdeklarationsdatei oder die `getSupportedContentTypes()`-Methode in der `simple-search`-Dienstimplementierung.
- **Ordner auflisten (`list-folders`)**

  Dieser Dienst wird verwendet, um die Inhaltsnavigation zusammen mit dem `list-contents`-Dienst zu erleichtern. Zusätzlich zur Inhaltssuche können Inhalte auch durch die Hierarchie der Ordner (oder ein anderes ähnliches Konzept im jeweiligen System) gefunden werden. Wenn dieser Dienst implementiert ist, wird erwartet, dass er Ordner auf oberster/Stammebene sowie Unterordner eines bestimmten übergeordneten Ordners bereitstellt, wenn dies während der Inhaltsnavigation angefordert wird.
  Bei einer einzelnen Ausführung wird nur eine Ebene der Ordnerliste erwartet. Es muss keine gesamte Ordnerhierarchie angegeben werden. Wenn dieser Dienst

implementiert ist, muss der Listeninhalts-Dienst unbedingt implementiert werden, um die Inhaltsnavigationsfunktion zu aktivieren.

Dies ist ein optionaler Dienst, und das Fehlen seiner Implementierung hat keinen Einfluss auf die Suchbarkeit von Inhalten in Content Picker. Die Inhaltsnavigation ist jedoch in der Content Picker-Benutzerschnittstelle in deaktiviert, wenn dieser Service nicht implementiert ist.

- **Listeninhalt (`list-contents`)**

  Dieser Dienst wird verwendet, um die Inhaltsnavigation zusammen mit dem `list-folders`-Dienst zu erleichtern. Wenn dieser Dienst implementiert ist, wird erwartet, dass er die Liste der Inhalte zur Verfügung stellt, die zu einem bestimmten Ordner gehören. Die Liste kann paginiert oder unpaginiert sein. Das Vorhandensein oder Nichtvorhandensein einer Unterstützung für eine paginierte Liste sollte anhand der `paginatedList`-Eigenschaft im Abschnitt Systeme in der Dienstdeklarationsdatei, wie im Servicedeklarationsdatei (on page 8)-Thema erläutert, eindeutig angegeben werden.

  Wenn dieser Dienst implementiert ist, muss der `list-folders`-Dienst unbedingt implementiert werden, um die Inhaltsnavigationsfunktion zu aktivieren.

  Dies ist ein optionaler Dienst, und das Fehlen seiner Implementierung hat keinen Einfluss auf die Suchbarkeit von Inhalten in Content Picker. Die Inhaltsnavigation ist jedoch in der Content Picker-Benutzerschnittstelle in deaktiviert, wenn dieser Service nicht implementiert ist.

- **Inhaltsdetails abrufen (`get-content-details`)**

  Alle mit dem `simple-search`-Dienst durchsuchten oder mit dem `list-contents`-Dienst aufgelisteten Inhalte können ausgewählt und für verschiedene Anwendungsfälle in Unica-Anwendungen verwendet werden. In solchen Anwendungsfällen können zu einem späteren Zeitpunkt Details zu bereits ausgewähltem Inhalt verlangt werden. Ein Beispiel hierfür ist die Funktion Inhaltsvorschau in Centralized Offer Management, in der Details zu bereits verlinkten Inhalten mit Angebotsattribut angezeigt werden. Wenn Unica -Anwendungen Details zu einem einzelnen Inhalt benötigen, wird der Service get-content-details durch sie Bereitstellung der eindeutigen ID des erforderlichen Inhalts aufgerufen.

Dies ist ein optionaler Dienst, und das Fehlen seiner Implementierung hat keinen Einfluss auf die Suchbarkeit von Inhalten in Content Picker. Nachfolgende Benutzeranforderungen zum Abrufen von Details eines Inhalts werden jedoch nicht bedient, wenn dieser Dienst nicht implementiert ist.

- **Objektschema abrufen (`get-object-schema`)**

  Dieser Dienst wird von Unica-Anwendungen aufgerufen, um die Details verschiedener Attribute im Inhalt anzuzeigen. Von diesem Dienst wird das gesamte Master-Schema aller Inhalte erwartet, das die Details zu jedem Inhaltsattribut enthalten sollte, z. B. den Typ und das Format des darin enthaltenen Werts sowie eine eindeutige Kennung, um dieses Attribut für das angegebene System eindeutig zu identifizieren. Ab der aktuellen Version von Unica Content Integration und Unica Centralized Offer Management werden diese Informationen verwendet, um Inhaltsattribute Angebotsattributen zuzuordnen und anschließend Angebotsattributwerte automatisch auszufüllen, indem der Inhalt aus der Inhaltsauswahl ausgewählt wird. Weitere Informationen zu dieser Funktion finden Sie im Unica Centralized Offer Management-Benutzerhandbuch.

  Dies ist ein optionaler Dienst, und das Fehlen seiner Implementierung hat keinen Einfluss auf die Suchbarkeit von Inhalten in Content Picker. Die Funktion Content Integration in der Centralized Offer Management ist jedoch für das jeweilige System nicht mehr verfügbar, wenn dieser Dienst nicht implementiert ist.

- **kognitive Analyse abfrufen (`get-cognitive-analysis`)**

  Dieser Dienst wird aufgerufen, um eine kognitive Analyse eines Bildes zu versuchen und die kognitiven Details entsprechend abzurufen. Er wird nur aufgerufen, wenn das jeweilige System in der Plattformkonfiguration als bevorzugter kognitiver Dienstanbieter konfiguriert ist. Weitere Informationen finden Sie im Unica Content Integration-Installations- und Konfigurationshandbuch.

  Dies ist ein optionaler Dienst, dessen Implementierung keine Auswirkung auf die Suche nach Inhalten oder auf andere Funktionen in Content Picker hat. Die Funktion zur kognitiven Kennzeichnung ist jedoch in der zentralen Angebotsverwaltung deaktiviert, wenn dieser Dienst nicht verfügbar ist.

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 *(on page 42)*
- Ableitungen von HttpService *(on page 52)*
- Ableitungen von Funktionsservice *(on page 55)*
- AbstractEntity *(on page 67)*
- Vorzeigbar *(on page 67)*

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.system.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.system.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 Content Integration 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 Content Integration SDK und enthält das Suchergebnis und die zugehörigen Paginierungsdetails.

Wenn das Wenn das Plug-in seine `simple-search`-Implementierung nicht vom `com.hcl.unica.system.integration.service.search.RestSearchService`-Service ableitet, um von der Content Integration Framework als `simple-search`-Service erkannt zu werden (das funktionale Pendant, das später diskutiert wird, ist auch eine gültige Option für die `simple-search`-Services, die mit dem funktionalen Ansatz implementiert wurden).

`RestSearchService` wird von der abstrakten Klasse `com.hcl.unica.system.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.

Die Einhaltung des Vertrags der `Presentable`-Schnittstelle beim Auffüllen des Inhaltsverzeichnisses in `ContentPage` ist ein wesentlicher Bestandteil dieser Dienstimplementierung. Die `Presentable`-Schnittstelle wird im folgenden Abschnitt ausführlicher behandelt.

- `com.hcl.unica.system.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.system.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 namens `getSupportedContentTypes` eingeführt Jede `simple-search`-Implementierung kann diese Methode optional 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 Methode sieht wie folgt aus:

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

Die Implementierung dieser Methode liefert ein `Map<String, String>`, das die unterstützten Kategorien von Inhalten darstellt, die im Zielsystem durchsucht werden können. Mit den Werten in dieser Map ist keine spezifische Semantik verbunden. Hierbei kann es sich um jedes aussagekräftige Schlüssel-Werte-Paar handeln. Es wirkt während des Suchvorgangs als Filter für die Client-Anwendung. Ab der aktuellen Implementierung von Unica Content Integration wird diese Map verwendet, um Einträge in einer Dropdown-Liste zu füllen, wobei die Schlüssel der Map zu Werten der Option werden und die Werte der Karte Anzeigebeschriftungen für die Option werden. Daher können Schlüssel interne Namen oder Identifikatoren tragen, und Werte sollten lesbare und aussagekräftige Texte sein. Wenn der Benutzer einen bestimmten Inhaltstyp durchsuchen muss, kann er eine oder mehrere Möglichkeiten aus den unterstützten Typen auswählen. In diesem Fall erhält der `simple-search`-Service eine Reihe von Schlüsseln, die den vom Benutzer ausgewählten Werten entsprechen. Der von der Clientanwendung empfangene Schlüssel kann vom `ExecutionContext`-Objekt abgerufen werden, indem er durch die `getRequest`-Methode navigiert wird und anschließend `getTypes()`aufruft. Die `simple-search`-Implementierung behandelt diese Gruppe von Schlüsseln entsprechend der Programmierschnittstelle des Zielsystems und filtert die Suchbegriffe nach diesen Werten.

Standardserviceparameter - supportedContentTypes

Das Überschreiben der `getSupportedContentTypes`-Methode wird nur empfohlen, wenn die Karte dynamisch generiert werden muss. Content Integration Framework bietet einen alternativen Ansatz zum statischen Definieren dieser Zuordnung mithilfe eines Standarddienstparameters namens `supportedContentTypes`, der unter dem `params`-Element in der Dienstdeklarationsdatei konfiguriert ist. Lesen Sie beispielsweise die `simple-search`-Dienstdeklaration für AEM und WCM in der `<ASSET_PICKER_HOME>/conf/plugin-services.yml`-Datei.

Inhaltskategorien auflisten (`list-content-categories`)

Im Folgenden sind die für den `list-content-categories`-Service verfügbaren speziellen Schnittstellen und Klassen aufgeführt:

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

  `com.example.service.rest`. Die ExampleContentCategoryListingService-Klasse im Asset-Integration-Starter-Projekt ist ein Schnellstarter für den RESTful-Dienst für Listeninhaltskategorien. Die Klasse ExampleContentCategoryListingService erweitert die Klasse RestContentCategoriesListService.

  Die `RestContentCategoriesListService`-Klasse verfügt über einen Typparameter RS, der die Art der von der entfernten REST-API empfangenen Antwort (Post-Deserialisierung) darstellt. In diesem Fall wird er beispielsweise als `List<ContentCategoryDetails>` angegeben.

  `RestContentCategoriesListService`-Klasse implementiert die `RestService`-Schnittstelle und definiert die `com.hcl.unica.system.model.request.content.categories` .ContentCategoryListRequest-Klasse als Typargument RQ für RestService. Somit wird das Objekt von ContentCategoryListRequest zum Input in alle Dienste für Listeninhaltskategorien (derselbe Input wird auch für das funktionale Gegenstück für Listeninhaltskategorien verwendet).

Die RestContentCategoriesListService-Klasse definiert nicht nur den Input-Typ für den Dienst für Listeninhaltskategorien, sondern überschreibt auch die transformResponse-Methode und schreibt vor, dass der Rückgabewert dieser Methode ein Objekt vom Typ `List<ContentCategory>` ist. Die Klasse `ContentCategory` ist Teil des Content Integration SDK.

Das Plugin muss die Implementierung des Dienstes für Listeninhaltskategorien von com.hcl.unica.system.integration.service.content.categories.list erweitern. RestContentCategoriesListService-Klasse, die vom Content Integration Framework als gültiger Dienst für Listeninhaltskategorien erkannt werden soll (das später erläuterte funktionale Gegenstück ist ebenfalls eine gültige Auswahl, von der aus erweitert werden kann).

`RestContentCategoriesListService` Erweitert von

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

.

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

  Dies ist eine allgemeine Basisklasse für `RESTful` und funktionale Implementierungen von `list-content-categories`-Diensten. Die hier behandelten Details gelten also auch für die funktionale Version von `list-content-categories`.

  Diese Klasse definiert die `com.hcl.unica.system.integration.service.gateway.ContentCategoriesListServiceGateway`-Schnittstelle als Servicegateway für den `list-content-categories`-Service. Diese Schnittstelle erstreckt sich von `com.hcl.unica.system.integration.service.gateway.ServiceGateway`-Schnittstelle und beauftragt die `ContentCategoryListRequest & List<ContentCategory>`-Objekte, die Eingabe- und Ausgabetypen für den `list-content-categories`-Dienst zu sein.

Ordner auflisten (`list-folders`)

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

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

  Die `com.aem.service.AemFolderListService`-Klasse in `aem-integration`-Projekt ist eine Referenzimplementierung für den RESTful-Listenordnerservice. `AemFolderListService`-Klasse erweitert die `RestFolderListService`-Klasse.

  Die `RestFolderListService`-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 `aem-integration`-Projekts definiert ist.

  `RestFolderListService`-Klasse implementiert die `RestService`-Schnittstelle und definiert die `com.hcl.unica.system.model.request.folder.list.FolderListRequest`-Klasse als das Typargument RQ für `RestService`. Somit wird das Objekt von `FolderListRequest` zum Input für alle `list-folders`-Dienste (derselbe Input wird auch für das funktionale Gegenstück von `list-folders` verwendet).

  Zusätzlich zum Definieren des Eingabetyps für den `list-folders`-Dienst überschreibt die `RestFolderListService`-Klasse auch die `transformResponse`-Methode und schreibt vor, dass der Rückgabewert dieser Methode ein Objekt vom Typ `List<Folder>` ist. Ordner ist ein im Content Integration SDK definierter Standardtyp.

  Das Plugin muss die Implementierung des `list-folders`-Dienstes von der `com.hcl.unica.system.integration.service.folder.list.RestFolderListService`-Klasse aus erweitern, um vom Content Integration Framework als gültiger Dienst für Listenordner erkannt zu werden (das später erläuterte funktionale Gegenstück ist auch eine gültige Auswahl für die Erweiterung).

  `RestFolderListService` wird von der Klasse `com.hcl.unica.system.integration.service.folder.list.AbstractFolderListService` abgeleitet.
- `com.hcl.unica.system.integration.service.folder.list.AbstractFolderListService`

Dies ist eine allgemeine Basisklasse für RESTful und funktionale Implementierungen von `list-folders`-Diensten. Die hier behandelten Details gelten also auch für die funktionale Version von `list-folders`.

Diese Klasse definiert die `com.hcl.unica.system.integration.service.gateway.FolderListServiceGateway`-Schnittstelle als Servicegateway für den `list-folders`-Service. Diese Schnittstelle erstreckt sich von der `com.hcl.unica.system.integration.service.gateway.ServiceGateway`-Schnittstelle und legt fest, dass die `FolderListRequest`- und `List<Folder>`-Objekte die Input- und Output-Typen für den `list-folders`-Dienst sind.

Listeninhalt (`list-contents`)

Im Folgenden sind die speziellen Schnittstellen und Klassen aufgeführt, die für den Dienst Listeninhalte verfügbar sind:

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

  Das `com.aem.service.AemContentListServiceclass` im `aem-integration`-Projekt ist eine Referenzimplementierung für den RESTful-`list-contents`-Dienst. Die `AemContentListServiceclass`-Klasse erstreckt sich von der `RestContentListService`-Klasse.

  Die `RestContentListService`-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 `aem-integration`-Projekts definiert ist.

  `RestContentListService`-Klasse implementiert die `RestService`-Schnittstelle und definiert die `com.hcl.unica.system.model.request.content.list.ContentListRequest`-Klasse als das Typargument **RQ** für `RestService`. Somit wird das Objekt von `ContentListRequest` zum Input für alle `list-contents`-Dienste (derselbe Input wird auch für das funktionale Gegenstück von `list-contents` verwendet).

Zusätzlich zum Definieren des Eingabetyps für den `list-contents`-Dienst überschreibt die `RestContentListService`-Klasse auch die `transformResponse`-Methode und schreibt vor, dass der Rückgabewert dieser Methode ein Objekt vom Typ `ContentPage` ist. Dieser Rückgabetyp ist der gleiche wie der für den `simple-search`-Dienst verwendete. `ContentPage` ist ein Standardtyp, der im Content Integration SDK definiert ist.

Das Plugin muss die Implementierung des `list-contents`-Dienstes von der `com.hcl.unica.system.integration.service.content.list.RestContentListService`-Klasse aus erweitern, um vom Content Integration Framework als gültiger `list-contents`-Dienst erkannt zu werden (das später erläuterte funktionale Gegenstück ist auch eine gültige Auswahl für die Erweiterung).

`RestContentListService` wird von der Klasse `com.hcl.unica.system.integration.service.content.list.AbstractContentListService` abgeleitet.

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

  Dies ist eine allgemeine Basisklasse für RESTful und funktionale Implementierungen von `list-contents`-Diensten. Die hier behandelten Details gelten also auch für die funktionale Version von `list-contents`.

  Diese Klasse definiert die `com.hcl.unica.system.integration.service.gateway.ContentListServiceGateway`-Schnittstelle als Servicegateway für den `list-contents`-Service. Diese Schnittstelle erstreckt sich von der `com.hcl.unica.system.integration.service.gateway.ServiceGateway`-Schnittstelle und legt fest, dass die `ContentListRequest`- und `ContentPage`-Objekte die Input- und Output-Typen für den `list-contents`-Dienst sind.

Inhaltsdetails abrufen (`get-content-details`)

Im Folgenden sind die speziellen Schnittstellen und Klassen aufgeführt, die für den Dienst Inhaltsdetails beschaffen verfügbar sind:

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

Die `com.aem.service.AemObjectDetailsService`-Klasse im `aem-integration`-Projekt ist eine Referenzimplementierung für den RESTful-`get-content-details`-Dienst. Die `AemObjectDetailsService`-Klasse erstreckt sich von der `RestContentDetailsService`-Klasse.

Die `RestContentDetailsService`-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 `aem-integration`-Projekts definiert ist.

`RestContentDetailsService`-Klasse implementiert die `RestService`-Schnittstelle und definiert die `com.hcl.unica.system.model.request.content.details.ContentDetailsRequest`-Klasse als das Typargument **RQ** für `RestService`. Somit wird das Objekt von `ContentDetailsRequest` zum Input für alle `get-content-details`-Dienste (derselbe Input wird auch für das funktionale Gegenstück von `get-content-details` verwendet).

Zusätzlich zum Definieren des Eingabetyps für den `get-content-details`-Dienst überschreibt die `RestContentDetailsService`-Klasse auch die `transformResponse`-Methode und schreibt vor, dass der Rückgabewert dieser Methode ein Objekt vom Typ Presentable ist.

Das Plugin muss die Implementierung des `get-content-details`-Dienstes von der `com.hcl.unica.system.integration.service.content.details.RestContentDetailsService`-Klasse aus erweitern, um vom Content Integration Framework als gültiger `get-content-details`-Dienst erkannt zu werden (das später erläuterte funktionale Gegenstück ist auch eine gültige Auswahl für die Erweiterung).

`RestContentDetailsService` wird von der Klasse `com.hcl.unica.system.integration.service.content.details.AbstractContentDetailsService` abgeleitet.

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

  Dies ist eine allgemeine Basisklasse für RESTful und funktionale Implementierungen von `get-content-details`-Diensten. Die hier behandelten Details gelten also auch für die funktionale Version von `get-content-details`.

Diese Klasse definiert die `com.hcl.unica.system.integration.service.gateway.ContentDetailsServiceGateway`-Schnittstelle als Service-Gateway für den `get-content-details`-Service. `ServiceGateways` sind die Mittel, um Eingabe- und Ausgabetypen des Dienstes programmgesteuert zu definieren und den Aufruf der Dienste zu erleichtern. Ein genauerer Blick auf diese Schnittstelle lässt erkennen, dass der `get-content-details`-Dienst das `ContentDetailsRequest`-Objekt annimmt und ein Presentable-Objekt zurückgibt.

Kognitive Analyse abrufen (get-cognitive-analysis)

Im Folgenden sind die für den Dienst Kognitive Analyse abrufen verfügbaren speziellen Schnittstellen und Klassen aufgeführt:

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

  Das `com.example.service.rest.ExampleCognitiveAnalysisService` im `asset-integration-starter`-Projekt ist eine Referenzimplementierung für den RESTful-`get-cognitive-analysis`-Dienst. Die `ExampleCognitiveAnalysisService`-Klasse erstreckt sich von der `RestCognitiveAnalysisService`-Klasse.

  Die `RestCognitiveAnalysisService`-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 `CognitiveDetails`-Klasse, die innerhalb des `asset-integration-starter`-Projekts definiert ist.

  `RestCognitiveAnalysisService`-Klasse implementiert die `RestService`-Schnittstelle und definiert die `com.hcl.unica.system.model.request.cognitive.analysis.CognitiveAnalysisRequest`-Klasse als das Typargument **RQ** für `RestService`. Somit wird das Objekt von `CognitiveAnalysisRequest` zum Input für alle `get-cognitive-analysis`-Dienste (derselbe Input wird auch für das funktionale Gegenstück verwendet).

  Zusätzlich zum Definieren des Eingabetyps für den `get-cognitive-analysis`-Dienst überschreibt die `RestCognitiveAnalysisService`-Klasse auch die `transformResponse`-Methode und schreibt vor, dass der Rückgabewert dieser Methode ein Objekt vom Typ

`com.hcl.unica.system.model.response.cognitive.analysis.CognitiveAnalysis` ist. `CognitiveAnalysis` ist ein Standardtyp, der im Content Integration SDK definiert ist.

Das Plugin muss die Implementierung des `get-cognitive-analysis`-Dienstes von der `com.hcl.unica.system.integration.service.cognitive.analysis.RestCognitiveAnalysisServi` Klasse aus erweitern, um vom Content Integration Framework als gültiger `get-cognitive-analysis`-Dienst erkannt zu werden (das später erläuterte funktionale Gegenstück ist auch eine gültige Auswahl für die Erweiterung).

`RestCognitiveAnalysisService` wird von der Klasse `com.hcl.unica.system.integration.service.cognitive.analysis.AbstractCognitiveAnalysisS` abgeleitet.

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

  Dies ist eine allgemeine Basisklasse für RESTful und funktionale Implementierungen von `get-cognitive-analysis`-Diensten. Die hier behandelten Details gelten also auch für die funktionale Version von `get-cognitive-analysis`.

  Diese Klasse definiert die `com.hcl.unica.system.integration.service.gateway.CognitiveAnalysisServiceGateway`-Schnittstelle als Servicegateway für den `get-cognitive-analysis`-Service. Diese Schnittstelle erstreckt sich von der `com.hcl.unica.system.integration.service.gateway.ServiceGateway`-Schnittstelle und legt fest, dass die `CognitiveAnalysisRequest`- und `CognitiveAnalysis`-Objekte die Input- und Output-Typen für den `get-cognitive-analysis`-Dienst sind.

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.system.integration.service.resourceloader.DefaultWebResourceLoaderServic`

  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.system.integration.service.resourceloader
  .DefaultWebResourceLoaderService
  ```

  `DefaultWebResourceLoaderService`-Klasse ist die Standardimplementierung des vom Content Integration SDK bereitgestellten `resource-loader`-Service. Wenn das Plug-in keinen eigenen `resource-loader`-Service implementiert, greift Content Integration Framework auf diese Standardimplementierung zurück. Die vom Content Integration 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. Es ist nicht erforderlich, die `resource-loader`-Implementierung von `DefaultWebResourceLoaderService` zu erweitern, wenn das Plug-in einen ganz anderen Ansatz zum Laden von Inhalten verwenden muss, z. B. das Lesen von Dateisystem, Datenbank, FTP-Server usw. In einem solchen Fall muss der Wert von entweder `HttpWebResourceLoaderService` für HTTP basierte Vorgehensweise oder `WebResourceLoaderService` für eine funktionale Vorgehensweise abgeleitet werden.

- `com.hcl.unica.system.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.system.model.request.resourceloader.ResourceRequest` bzw. `byte[]`empfangen wird. Die `ResourceRequest`-Klasse enthält die Ressourcen-URL und den Systembezeichner. In ähnlicher Weise arbeitet `resource-loader` mit einem Bytearray, wenn der Inhalt von einer entfernten HTTP-URL erfolgreich gelesen wurde.

Wenn das Wenn das Plug-in seine `resource-loader`-Implementierung nicht von der `DefaultWebResourceLoaderService`-Klasse ableitet, muss es sich zumindest von der `com.hcl.unica.system.integration.service.resourceloader.HttpWebResourceLoaderService`-Klasse ableiten, um von der Content Integration Frameworkals `resource-loader`-Service erkannt zu werden (das funktionale Pendant, das später diskutiert wird, ist auch eine gültige Option für die `resource-loader`-Services, die mit dem funktionalen Ansatz implementiert wurden).

- `com.hcl.unica.system.integration.service.resourceloader.AbstractWebResourceLoaderServi`
  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.system.integration.service.gateway
.ResourceLoaderServiceGateway
```

Informationen über die Rolle von Servicegateways beim Serviceaufruf finden Sie unter Serviceaufruf (on page 31). `ResourceLoaderServiceGatewayy`-Schnittstelle definiert `ResourceRequest` und `HttpResponse<?>` als Input- und Outputtypen für den `resource-loader`-Service. `HttpResponse` ist eine Schnittstelle, die von der `WebResource`-Klasse implementiert wird. Darin werden die HTTP-Antwortheader, der Textkörper oder die Nutzdaten sowie die Cookies, die von der entfernten URL empfangen werden, eingekapselt. Auch wenn der angepasste `resource-loader`-Service den Inhalt nicht über das Web abrufen kann, muss er das Objekt der `WebResource` (oder anderen Implementierung von `HttpResponse`), die mit den entsprechenden Details gefüllt ist, zurückgeben. Wenn Sie das `WebResource` nicht entsprechend ausfüllen, kann dies zu Problemen beim Laden von Inhalten für Clientanwendungen führen. Die `WebResource` stellt eine Builder-API zur Verfügung, um ein Objekt mit erforderlichen Details zu erstellen Das wichtigste ist es, den `Content-Type`-Header zu füllen, damit die Clientanwendung die Nutzdaten entsprechend bearbeiten kann. Ebenso muss der `Content-Disposition`-Header entsprechend ausgefüllt werden, der den Dateinamen enthält, der dem Inhalt zugeordnet ist.

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.system.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. Es wird von der `com.hcl.unica.system.integration.service.search.SearchService` Klasse abgeleitet.

  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 aus der `com.hcl.unica.system.integration.service.search.SearchService`-Klasse ableiten, um von der `simple-search` als Content Integration Framework-Service erkannt zu werden (das im vorigen Abschnitt beschriebene RESTful-Pendant ist ebenfalls eine gültige Wahl für die `simple-search`-Services, die mit dem RESTful-Ansatz implementiert wurden).

  Das `SearchService` wird von der folgenden `com.hcl.unica.system.integration.service.search.AbstractSearchService` abstrakten Klasse abgeleitet: Es wird eine weitere Methode mit dem Namen

`getSupportedContentTypes` eingeführt. Weitere Informationen zur Methode finden Sie unter Ableitungen von RestService (on page 42) .

Ressourcenlader (`resource-loader`)

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

- `com.hcl.unica.system.integration.service.resourceloader.WebResourceLoaderService`
  Die `com.example.service.functional.ResourceLoaderService`-Klasse im `asset-integration-starter`-Projekt ist eine Schnellstartimplementierung für `resource-loader` Funktionsservice. Es wird von der folgenden Klasse abgeleitet:

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

  Die `WebResourceLoaderService`-Klasse implementiert die `FunctionalService`-Schnittstelle und definiert die `ResourceRequest` und `HttpResponse`-Typen als die Typargumente RQ & RS für den `FunctionalService`. Somit wird das Objekt des `ResourceRequest` zu einem Input für alle `resource-loader`-Services, und das `HttpResponse` wird nach Abschluss des Services als Output erwartet (die gleichen Input- und Outputtypen werden für das RESTful-Pendant des Ressourcen-Laders verwendet). Weitere Informationen zu `ResourceRequest`- und `HttpResponse`-Typen finden Sie unter Ableitungen von RestService (on page 42).

  Das Plug-in muss seine `resource-loader`-Implementierung aus der `com.hcl.unica.system.integration.service.resourceloader.WebResourceLoaderService`-Klasse ableiten, um von der `resource-loader` als Content Integration Framework-Service erkannt zu werden (das im vorigen Abschnitt beschriebene HTTP-Pendant ist ebenfalls eine gültige Wahl für die `resource-loader`-Services, die mit dem HTTP-Ansatz implementiert wurden).

  Der WebResourceLoaderService erstreckt sich über die folgende Klasse:

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

Weitere Informationen zu dieser Klasse finden Sie in der Ableitungen von RestService (on page 42).

Inhaltskategorien auflisten (`list-content-categories`)

Im Folgenden sind die speziellen Schnittstellen und Klassen aufgeführt, die für den Dienst Listeninhaltskategorien verfügbar sind:

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

  Das Plugin kann alternativ den funktionalen Ansatz zur Implementierung des `list-content-categories`-Dienstes wählen, indem die Implementierung von der `ContentCategoriesListService`-Klasse erweitert wird. Die `ContentCategoriesListService`-Klasse implementiert die `FunctionalService`-Schnittstelle und beauftragt die `ContentCategoryListRequest` und `List<ContentCategory>`-Klassen als die Typargumente **RQ** und **RS** für den `FunctionalService`. Dadurch wird das Objekt des `ContentCategoryListRequest` zu einem Input für alle `list-content-categories`-Dienste und das Objekt vom Typ `List<ContentCategory>` wird nach Abschluss des Dienstes als Output erwartet.
- Das Plugin muss seine Implementierung von Listeninhaltskategorien aus der `com.hcl.unica.system.integration.service.content.categories.list.ContentCategoriesLists` Klasse ableiten, um vom Content Integration Framework als gültiger `list-content-categories`-Dienst erkannt zu werden (das im vorherigen Abschnitt beschriebene RESTful-Gegenstück ist ebenfalls eine gültige Option, um es zu erweitern).

  `ContentCategoriesListService` wird von der Klasse `AbstractContentCategoriesListService` abgeleitet. Details zur Klasse `AbstractContentCategoriesListService` werden im Thema Ableitungen von RestService (on page 42) behandelt.

Ordner auflisten (`list-folders`)

Im Folgenden sind die für den Dienst Ordner auflisten verfügbaren speziellen Schnittstellen und Klassen aufgeführt:

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

Das Plugin kann alternativ den funktionalen Ansatz zur Implementierung des `list-folders`-Dienstes wählen, indem die Implementierung von der `FolderListService`-Klasse erweitert wird. Die `FolderListService`-Klasse implementiert die `FunctionalService`-Schnittstelle und beauftragt die `FolderListRequest` und `List<Folder>`-Klassen als die Typargumente **RQ** und **RS** für den `FunctionalService`. Dadurch wird das Objekt des `FolderListRequest` zu einem Input für alle `list-folders`-Dienste und das Objekt vom Typ `List<Folder>` wird nach Abschluss des Dienstes als Output erwartet.

- Das Plugin muss seine Implementierung von Listenordnern aus der `com.hcl.unica.system.integration.service.folder.list.FolderListService`-Klasse ableiten, um vom Content Integration Framework als gültiger `list-folders`-Dienst erkannt zu werden (das im vorherigen Abschnitt beschriebene RESTful-Gegenstück ist ebenfalls eine gültige Option, um es zu erweitern).

  `FolderListService` wird von der Klasse `AbstractFolderListService` abgeleitet. Details zur Klasse `AbstractFolderListService` werden im Thema Ableitungen von RestService (on page 42) behandelt.

Listeninhalt (`list-contents`)

Im Folgenden sind die speziellen Schnittstellen und Klassen aufgeführt, die für den Dienst Listeninhalte verfügbar sind:

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

  Das Plugin kann alternativ den funktionalen Ansatz zur Implementierung des `list-contents`-Dienstes wählen, indem die Implementierung von der `ContentListService`-Klasse erweitert wird. Die `ContentListService`-Klasse implementiert die `FunctionalService`-Schnittstelle und beauftragt die `ContentListRequest` und `ContentPage`-Klassen als die Typargumente **RQ** und **RS** für den `FunctionalService`. Dadurch wird das Objekt des `ContentListRequest` zu einem Input für alle `list-contents`-Dienste und das Objekt vom Typ `ContentPage` wird nach Abschluss des Dienstes als Output erwartet.
- Das Plugin muss seine Implementierung von Listeninhalten aus der `com.hcl.unica.system.integration.service.content.list.ContentListService`-

Klasse ableiten, um vom Content Integration Framework als gültiger -Dienst erkannt zu werden (das im vorherigen Abschnitt beschriebene RESTful-Gegenstück ist ebenfalls eine gültige Option, um es zu erweitern).

`ContentListService` wird von der Klasse `AbstractContentListService` abgeleitet. Details zur Klasse `AbstractContentListService` werden im Thema Ableitungen von RestService (on page 42) behandelt.

Inhaltsdetails abrufen (`get-content-details`)

Im Folgenden sind die speziellen Schnittstellen und Klassen aufgeführt, die für den Dienst Inhaltsdetails beschaffen verfügbar sind:

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

  Das Plugin kann alternativ den funktionalen Ansatz zur Implementierung des `get-content-details`-Dienstes wählen, indem die Implementierung von der `ContentDetailsService`-Klasse erweitert wird.

  Die `ContentDetailsService`-Klasse implementiert die `FunctionalService`-Schnittstelle und beauftragt die `ContentDetailsRequest`- und Presentable-Klassen als die Typargumente **RQ** und **RS** für den `FunctionalService`. Dadurch wird das Objekt des `ContentDetailsRequest` zu einem Input des `get-content-details`-Dienstes und das Objekt vom Typ Presentable wird nach Abschluss des Dienstes als Ausgabe erwartet.

  Das Plugin muss seine `get-content-details`-Implementierung aus der `com.hcl.unica.system.integration.service.content.details.ContentDetailsService`-Klasse ableiten, um vom Content Integration Framework als gültiger `get-content-details`-Dienst erkannt zu werden (das im vorherigen Abschnitt beschriebene RESTful-Gegenstück ist ebenfalls eine gültige Option, um es zu erweitern).

  `ContentDetailsService` wird von der Klasse `AbstractContentDetailsService` abgeleitet. Details zur Klasse `AbstractContentDetailsService` werden im Thema Ableitungen von RestService (on page 42) behandelt.

Objektschema abrufen (`get-object-schema`)

`get-object-schema` -Dienst wird verwendet, um das Hauptschema des Domänenobjekts oder der Domänenentität zu generieren, das bzw. die vom jeweiligen System zur Darstellung des Inhalts verwendet wird. Das Master-Schema in einfachster Form besteht nur aus hierarchischen Metadaten jedes zuordenbaren Inhaltsattributs. Es wird erwartet, dass die Attributhierarchie und die Metadaten mit der JSON-Darstellung des Domäne übereinstimmen. Attributmetadaten umfassen hauptsächlich den Datentyp des Attributs, das Format des im Attribut gespeicherten Werts, die eindeutige ID des Attributs und den Anzeigetitel oder die Anzeigebezeichnung für das Attribut.

Im Folgenden sind die für den `get-object-schema`-Service verfügbaren speziellen Schnittstellen und Klassen aufgeführt:

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

  Die `ObjectSchemaProviderService`-Klasse implementiert die `FunctionalService`-Schnittstelle und beauftragt die `com.hcl.unica.system.model.ObjectSchemaRequest` und `com.hcl.unica.system.model.json.schema.ObjectSchema`-Klassen als die Typargumente **RQ** und **RS** für den `FunctionalService`. Dadurch wird das Objekt des `ObjectSchemaRequest` zu einem Input für alle `get-object-schema`-Dienste und das Objekt vom Typ `ObjectSchema` wird nach Abschluss des Dienstes als Output erwartet. Das Plugin sollte das `ObjectSchema` jedoch nicht selbst erstellen. Es sollte nur die folgende abstrakte Methode aus der `ObjectSchemaProviderService`-Klasse überschreiben und implementieren.

  ObjectProfile getObjectProfile(ObjectSchemaRequest objectSchemaRequest)

  Die `getObjectProfile()` Methode akzeptiert `ObjectSchemaRequest` und gibt `ObjectProfile` zurück. (Diese Typen werden in nachfolgenden Abschnitten behandelt.)

  Das Plugin muss `get-object-schema`-Implementierung aus der `com.hcl.unica.system.integration.service.object.schema.ObjectSchemaProviderService`-Klasse ableiten, um vom Content Integration Framework als gültiger `get-object-schema`-Dienst erkannt zu werden. Es gibt kein RESTful-Gegenstück zu dieser Standard-Superklasse, da die Objektschemagenerierung keine HTTP-Interaktion enthält. Plugins

können benutzerdefinierte RESTful-Dienste implementieren und bei Bedarf intern aus dem `get-object-schema`-Dienst heraus aufrufen.

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

  Das Objekt dieser Klasse wird als Eingabe für den `get-object-schema`-Dienst bereitgestellt. Die wichtigste Methode dieser Klasse ist `getObjectIdentity()`, das ein Objekt vom Typ `com.hcl.unica.system.model.ObjectIdentity` zurückgibt, das die Details des Inhalts enthält, den der Benutzer ausgewählt hat, um das Masterschema anzufordern. Dazu gehören `applicationId` (die System-ID), `objectType` (Inhaltstyp-/Kategorie-ID) und `objectId` (eindeutige Kennung des ausgewählten Inhalts). Unabhängig von der Kategorie und/oder dem Inhalt, die der Benutzer beim Einrichten der Inhaltszuordnung ausgewählt hat, muss das generierte Schema Attribute aller Inhaltsarten enthalten, die vom jeweiligen System unterstützt werden. Mit anderen Worten, für die Zuordnung aller vom angegebenen System bereitgestellten Inhaltstypen wird nur ein Masterschema verwendet.

  Die `getEnrichmentObjectJson()`-Methode in der `ObjectSchemaRequest`-Klasse kann ab dem aktuellen Release ignoriert werden.

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

  Dies ist ein Rückgabetyp der `getObjectProfile()`-Methode im `get-object-schema`-Dienst. Sie enthält den Java-Typ, der der Domäne / dem Objekt für das jeweilige System entspricht. Content Integration Framework verwendet diesen Java-Typ, um das Schema für öffentliche und nicht öffentliche, nicht statische Klasseneigenschaften (einschließlich Enums & Optionals) zu generieren. Mit der `@MappableAttribute`-Annotation kann jede einzelne Klasseneigenschaft konfiguriert werden, um das von Content Integration Framework generierte Schema zu steuern. Lesen Sie das `com.aem.model.response.simplesearch.SimpleSearchItem`-Domänenobjekt im `aem-integration reference`-Projekt, um eine Vorstellung davon zu erhalten, wie diese Anmerkung verwendet wird. Weitere Details zu `@MappableAttribute` finden Sie im nächsten Abschnitt. `ObjectProfile` kann optional eine Instanz von `com.hcl.unica.system.integration.service.object.schema.ObjectSchemaEnricher` einschließen, um Attribute dynamisch zu dem so generierten Schema hinzuzufügen, zu ändern oder zu entfernen. Im nächsten Abschnitt wird `ObjectSchemaEnricher` detailliert erläutert.

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

  `ObjectSchemaEnricher` ist eine abstrakte Klasse. Plug-in sollte es erweitern, um die gewünschte Implementierung zu haben. Der Typparameter für die `ObjectSchemaEnricher`-Klasse stellt den Java-Typ mit den zusätzlichen Details dar, die zur Bereicherung des statisch generierten Objektschemas erforderlich sind. Diese zusätzlichen Details werden möglicherweise von den Clientanwendungen der Unica Content Integration bereitgestellt. Ab dem aktuellen Release werden keine zusätzlichen Details bereitgestellt, daher sollte sie bei der Implementierung der Schemaerweiterung auf Ungültig gesetzt werden. `ObjectSchemaEnricher` deklariert nur eine abstrakte Methode, die vom Plug-in implementiert werden sollte:

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

  Das erste Argument für diese Methode ist eine Instanz der `com.hcl.unica.system.model.json.schema.ObjectSchema`-Klasse. Es enthält das automatisch generierte Domänenobjektschema, das vom in `ObjectProfile` angegebenen Java-Typ abgeleitet ist. Im Kern ist `ObjectSchema` nur ein `Map<String, AttributeSchema>`, wobei Klasseneigenschaftsnamen die Schlüssel dieser Karte bilden und Eigenschaftsmetadaten als Objekt von `AttributeSchema` enden. Wenn sich die Klasseneigenschaft wiederum auf ein anderes Objekt bezieht, hat das entsprechende `AttributeSchema` ein weiteres `Map<String, AttributeSchema>`, das die Attribute dieses Objekttyps enthält, usw.

  Note: Es ist wichtig, zu beachten, dass Attributnamen, die als Schlüssel in der Attributzuordnung verwendet werden, den JSON-Eigenschaften entsprechen, die in der JSON-Darstellung des Domäne enden. Wenn die `@JsonProperty`-Annotation verwendet wird, um den JSON-Eigenschaftsnamen für ein bestimmtes Klassenattribut zu überschreiben, erkennt Content Integration Framework diesen automatisch und verwendet den überschriebenen Eigenschaftsnamen.

`ObjectSchema` und `AttributeSchema` erweitern sich von der abstrakten Klasse `com.hcl.unica.system.model.json.schema.AttributeContainer`. `AttributeContainer` bietet `ObjectSchema`- und `AttributeSchema`-Klassen bequeme Methoden zum Navigieren durch die Attributhierarchie sowie zum Hinzufügen, Ändern und Entfernen von Attributen auf jeder Hierarchieebene, um die Schemaanreicherung zu vereinfachen. Auf Attribute auf jeder Hierarchieebene kann mithilfe ihrer Namen, wie sie in der JSON-Darstellung erscheinen, zugegriffen und bearbeitet werden.

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

  `@MappableAttribute`-Annotation bietet eine Möglichkeit, um zu steuern, wie Content Integration Framework ein Objektschema aus dem entsprechenden Java-Typ generiert. Die Verwendung `@MappableAttribute` von ist nicht obligatorisch. Wird es nicht verwendet, berechnet Content Integration Framework automatisch Metadaten für Eigenschaften. Falls erforderlich, sollte diese Anmerkung über den gewünschten Klasseneigenschaften platziert werden. Die folgenden Annotationsattribute können zur Steuerung der Schemagenerierung verwendet werden:

  - **ausgeblendet** – Setzen Sie diese Eigenschaft auf wahr, um bestimmte Eigenschaften explizit aus dem Objektschema auszuschließen (`@JsonIgnore` wird derzeit vom Content Integration Framework nicht berücksichtigt. Daher muss jede Eigenschaft, die von der JSON-Darstellung mit `@JsonIgnore` ausgeschlossen ist, explizit vom Schema ausgeschlossen werden.
  - **id** – Angabe einer eindeutigen Kennung für die Eigenschaft. Content Integration Framework benötigt für jede zugeordnete Klasseneigenschaft eine eindeutige ID. Wenn `@MappableAttribute` nicht verwendet wird oder keine **ID** angegeben ist, wird eine ID automatisch basierend auf der Position der Eigenschaft innerhalb der Klasse generiert.

    Die automatische Generierung der Attributkennung hängt vom Namen und der hierarchischen Position der Klasseneigenschaft im Domänenobjektdiagramm ab. Wenn der Eigenschaftsname geändert und/oder in der Hierarchie des Objektdiagramms nach oben oder unten verschoben wird, ändert sich die ihm zugeordnete ID. Ein solches Refaktorieren kann das Content Integration Framework beim Lesen der Werte refaktorierter Attribute irreführen und zu

unerwünschten Daten in zugeordneten Inhalten führen (z. B. Angebote in COM). Daher wird empfohlen, eindeutige Attribut-IDs manuell zu zuordnen, um solche unbeabsichtigten Änderungen an Attribut-IDs zu vermeiden, die unabhängig vom Namen und der Position der Klasseneigenschaften konstant bleiben.

- **Titel** – Anzeigetitel/-bezeichnung für die Eigenschaft. Wird dies nicht angegeben, generiert Content Integration Framework eines unter Verwendung des Eigenschaftsnamens.
- **Typ** – Einer der Werte aus `com.hcl.unica.system.model.json.schema.generator.annotations.AttributeType`. Wird dies nicht angegeben, erkennt Content Integration Framework automatisch den entsprechenden Typ.
- **Format** – Einer der Werte aus `com.hcl.unica.system.model.json.schema.generator.annotations.AttributeFormat`. Content Integration Framework kann standardmäßige temporäre Java-Typen (`Date`, `LocalDateTime`, `Instant`) automatisch identifizieren und den Attributtyp auf `DATETIME` setzen. Andere Formate sollten explizit deklariert werden.
- **Implementation** – Sollte für polynische Referenzen verwendet werden, um explizit den Java-Typ zu deklarieren, der für die automatische Schemagenerierung berücksichtigt werden soll.
- **hiddenProperties** - `@MappableAttribute`-Annotation kann auf Klassenebene verwendet werden, um mehrere Eigenschaften an einer einzigen Stelle auszublenden. `hiddenProperties` nimmt ein Array aus Zeichenfolgen, die die Namen der Eigenschaften (direkt und übernommen) enthalten, die aus dem automatisch generierten Schema ausgeschlossen werden sollen. Dies ist besonders nützlich, um Eigenschaften zu ausblenden, die von übergeordneten Klassen von Drittanbietern übernommen wurden.

Zuordnung zwischen Java-Typ und AttributeType

In der folgenden Tabelle wird die Zuordnung zwischen Java-Typ und AttributeType/ AttributeFormat zusammengefasst, die vom Content Integration Framework für die automatische Schemagenerierung verwendet wird:

Java-Typ	AttributeType	AttributeFormat
◦ String ◦ Character ◦ Char ◦ CharSequence ◦ LocalDate ◦ LocalTime ◦ ZonedDateTime ◦ OffsetDateTime ◦ OffsetTime ◦ ZoneId ◦ Calendar ◦ UUID	STRING	
◦ Boolean ◦ boolean	BOOLEAN	
◦ BigInteger ◦ Integer ◦ Int ◦ Long ◦ Long ◦ Short ◦ Short ◦ Byte ◦ byte	INTEGER	
◦ BigDecimal ◦ Number ◦ Double ◦ Double ◦ Float ◦ float	NUMBER	
◦ Date ◦ LocalDateTime	INTEGER	DATETIME

Java-Typ	AttributeType	AttributeFormat
◦ `Instant` Das Content Integration Framework erwartet, dass Datumswerte in UTC-Standardzeit ausgedrückt werden. Temporäre Werte, die in einer anderen Zeitzone ausgedrückt werden, können in weiteren Anwendungsfällen zu ungenauen zeitlichen Berechnungen führen.		

kognitive Analyse abfrufen (`get-cognitive-analysis`)

Im Folgenden sind die für den Dienst Kognitive Analyse abrufen verfügbaren speziellen Schnittstellen und Klassen aufgeführt:

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

  Das Plugin kann alternativ den funktionalen Ansatz zur Implementierung des `get-cognitive-analysis`-Dienstes wählen, indem die Implementierung von der `CognitiveAnalysisService`-Klasse erweitert wird. Die `CognitiveAnalysisService`-Klasse implementiert die `FunctionalService`-Schnittstelle und beauftragt die `CognitiveAnalysisRequest` und `CognitiveAnalysis`-Klassen als die Typargumente **RQ** und **RS** für den `FunctionalService`. Dadurch wird das Objekt des `CognitiveAnalysisRequest` zu einem Input für alle `get-cognitive-analysis`-Dienste und das Objekt vom Typ `CognitiveAnalysis` wird nach Abschluss des Dienstes als Output erwartet.
- Das Plugin muss seine `get-cognitive-analysis`-Implementierung aus der `com.hcl.unica.system.integration.service.cognitive.analysis.CognitiveAnalysisService`-Klasse ableiten, um vom Content Integration Framework als gültiger `get-cognitive-`

`analysis`-Dienst erkannt zu werden (das im vorherigen Abschnitt beschriebene RESTful-Gegenstück ist ebenfalls eine gültige Option, um es zu erweitern).

CognitiveAnalysisService erweiter sich von der `AbstractCognitiveAnalysisService`-Klasse. Details zur Klasse `AbstractCognitiveAnalysisService` werden im Thema Ableitungen von RestService (on page 42) behandelt.

AbstractEntity

Die `com.hcl.unica.system.model.AbstractEntity` Klasse Klasse stellt eine allgemeine Domänenentität dar. Für das jetzige Release enthält diese abstrakte Klasse keine Implementierung.

Für das Content Integration Framework müssen Plugins jedoch ihre Domänenentitäten aus der `com.hcl.unica.system.model.AbstractEntity`-Klasse erweitern. Dies stellt sicher, dass `AbstractEntity` die Basis für den Umgang mit Domänenentitäten innerhalb des Content Integration Framework ist.

Bei den Plugin-Implementierungen muss sich die Klasse, die zur Darstellung eines einzelnen Inhalts verwendet wird, der von den `simple-search`-, `list-contents`- und `get-content-details`-Diensten zurückgegeben wird, von der `AbstractEntity`-Klasse erweitern.

Vorzeigbar

Um einen einzelnen Inhalt rendern zu können, der von den Diensten Einfache Suche, Listeninhalte und Inhalt abrufen zurückgegeben wird, muss die von diesen Diensten verwendete Domänenentitätsklasse die `com.hcl.unica.system.model.presentation.Presentable`-Schnittstelle implementieren und die `getPresentationDetails()`-Methode überschreiben. Das von der `com.hcl.unica.system.model.presentation.Presentable$PresentationDetails`-Methode zurückgegebene `getPresentationDetails()`-Objekt muss sowohl die `TextualPresentation`- als auch die `MultimediaPresentation`-Details enthalten.

`TextualPresentation` enthält die folgenden Angaben:

- **Note:** Die hervorgehobenen Felder sind obligatorisch. Geben Sie, falls verfügbar, Details zu den anderen Feldern an.

- **`heading` – Titel des Inhalts**
- `subheadings` – Liste der Unterüberschriften für den Inhalt
- `summary` – Zusammenfassung oder Beschreibung des Inhalts
- **`name` – Sollte für den mit dem Inhalt verknüpften Dateinamen verwendet werden.**
- `tags` – Tags, die dem Inhalt zugeordnet sind (dies wird von Plug-ins verwendet, um den MIME-Typ oder die Kategorie des Inhalts zu vermitteln)

`MultimediaPresentation` enthält hingegen folgende Angaben:

- **Note:** Die hervorgehobenen Felder sind obligatorisch. Geben Sie, falls verfügbar, Details zu den anderen Feldern an.
- **`id` - Eindeutige ID des Inhalts**
- **`folderId` - Eindeutige ID des jeweiligen Ordnerinhalts**
- **`mimeType` - MIME-Typ des ursprünglichen Inhalts.**
- `size` - Größe des ursprünglichen Inhalts in Byte
- **`resourceUrl` - Absolute URL zum originalen Inhalt**
- `thumbnailUrl` - Absolute URL zum Inhaltsminiaturbild, falls verfügbar
- **`fileName` - Dateiname, der dem ursprünglichen Inhalt zugeordnet ist**
- **`type` – Typ-/Kategorie-ID des Inhalts (muss einer der Werte aus unterstützten Inhaltstypen sein, die mithilfe einer der anwendbaren Alternativen eingerichtet werden, die vom Content Integration Framework bereitgestellt werden)**
- `list of variants` – Jede Variante unterstützt nahezu dieselben Details wie die primären MultimediaPresentation-Details außer thumbnailUrl (es kann nur über eine eigene resourceUrl verfügen), folderId und variants (variant kann keine weiteren Varianten haben)

Builder-API

Fast alle in den vorherigen Abschnitten beschriebenen Standardtypen bieten die Builder-API für die einfache Erstellung von Objekten an.

Zum Beispiel kann `TextualPresentation` mit der folgenden Syntax erstellt werden, anstatt es in Konstruktor- und Setter-Operationen aufzuteilen:

```
TextualPresentation.builder()
```

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

Die Verwendung der Builder-API zum Erstellen von Standardobjekten ist nicht obligatorisch. Es hält jedoch Plugin-Implementierungen sicher sauber, während komplexe Objekte bearbeitet werden.

Standardausnahmen

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

RESTful-Ansatz

Content Integration Framework behandelt Fehlerzustände, die durch Services entstehen, die mit dem RESTful-Ansatz implementiert wurden.

Darüber hinaus leitet Content Integration Framework 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 Content Integration Framework 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 Zielsystem und sind im `com.hcl.unica.system.integration.exception`-Paket enthalten.

- **SystemNotFoundException**

  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.
- **UnreachableSystemException**

  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.
- **SluggishSystemException**

  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.
- **InternalSystemError**

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.

Content Integration Framework schließt die von den Serviceimplementierungen empfangenen Ausnahmebedingungen in `com.hcl.unica.system.integration.exception.ServiceExecutionException` oder deren Subtyp ein. Ausnahmen, die von REST-Services oder HTTP-Services empfangen wurden, werden in `com.hcl.unica.system.integration.exception.HttpServiceExecutionException` eingewickelt während die von funktionalen Services empfangenen Dienste in `com.hcl.unica.system.integration.exception.ServiceExecutionException` eingewickelt sind.

Wie in Serviceaufruf (on page 31) erläutert, `HttpServiceExecutionException` stellt eine Methode zur Verfügung, um ein `Optional<HttpResponse>`-Objekt zu erhalten. Wenn die Serviceausführung vor dem Initiierung eines HTTP-Anrufs fehlschlägt, enthält dieses optionale Objekt keine `HttpResponse` .

Protokollfunktionen (Loggers)

Content Integration Framework 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.

Chapter 4. Entwicklungsumgebung konfigurieren

Instellen Sie die Entwicklungsumgebung in Eclipse IDE, um Ihre Plugins zu schreiben. Verwenden Sie alle Java EE IDE Ihrer Wahl und nehmen Sie die in diesem Abschnitt erwähnten erforderlichen Konfigurationen vor. Sie benötigen bestimmte Artefakte von `<ASSET_PICKER_HOME>` , um die Umgebungskonfiguration abzuschließen. In diesem Abschnitt werden Informationen zum Aufbau und zur Verpackung von Produkten mithilfe von Apache Maven bereitgestellt, um sicherzustellen, dass Apache Maven installiert ist.

Gehen Sie wie folgt vor, um Ihre Entwicklungsumgebung zu konfigurieren:

1. Kopieren Sie das `asset-integration-starter`-Projekt von der `<ASSET_PICKER_HOME>/dev-kits/`-Position aus und platzieren Sie es in Ihrem Entwicklungsarbeitsbereich.
2. Öffnen Sie die Eclipse IDE.
3. Wählen Sie **Datei > Importieren**aus.
   Das Dialogfeld **Auswählen** wird geöffnet.
4. Wählen Sie Ihre **WAR-Datei** aus und klicken Sie auf **Weiter**.
   Das Dialogfeld **War importieren** wird angezeigt.
5. Klicken Sie auf Durchsuchen, navigieren Sie zu `<ASSET_PICKER_HOME>`- und `select asset-viewer.war`-Datei.
6. Klicken Sie auf **Fertig stellen**.
   Der **War-Import: Das Dialogfeld Webbibliotheken** wird angezeigt.
7. Klicken Sie auf **Fertig stellen**.
8. Wählen Sie **Fenster > Ansicht anzeigen > Andere** aus.
   Der Dialog **Ansicht anzeigen** wird angezeigt.
9. Wählen Sie **Server** aus und klicken Sie auf **Öffnen**.

Als Beispiel wird die Verwendung von Apache Tomcat 9,0 für die Ausführung von Content Integration veranschaulicht. Sie können alle unterstützten Anwendungsserver verwenden und die erforderlichen Konfigurationen vornehmen.

a. Öffnen Sie die `conf/server.xml`-Datei aus Ihrem Apache Tomcat 9,0-Installationsverzeichnis und fügen Sie den folgenden Eintrag mit den entsprechenden Datenbankdetails innerhalb des *<GlobalNamingResources>*-Elements ein. Bitte ersetzen Sie, *<DRIVER_CLASS_NAME>*,*<URL_TO_YOUR_PLATFORM_DATABASE>*,*<DATABASE_USERN* und *<DATABASE_PASSWORD>* mit Platform- Datenbankdetails:

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

b. Öffnen Sie die `conf/context.xml`-Datei aus dem Installationsverzeichnis von Apache Tomcat 9,0 und fügen Sie den folgenden Eintrag in das *<Context>*-Element ein:

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

10. Um Apache Tomcat 9,0 als neuen Server in Eclipse hinzuzufügen, führen Sie die folgenden Schritte aus:

    a. Klicken Sie auf der Registerkarte **Server** auf den Link, um einen neuen Server zu erstellen.
    Das Fenster **Neuen Server definieren** wird geöffnet.

b. Wählen Sie den **Tomcat Version 9.0-Server** aus und geben Sie Werte für den **Hostnamen** und den **Servernamen** an.

c. Klicken Sie auf **Weiter**.
Der Server wurde erfolgreich hinzugefügt.

d. Doppelklicken Sie auf der Registerkarte **Server** auf den Eintrag für den neu hinzugefügten Server.
Der Dialog **Übersicht** wird angezeigt.

e. Klicken Sie auf den Link **Startkonfiguration öffnen**.
Das Dialogfenster **Einstellungen für Startkonfiguration bearbeiten** wird angezeigt.

f. Bearbeiten Sie die Startkonfigurationen, um die folgenden JVM-Argumente hinzuzufügen

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

g. Klicken Sie auf **OK**.

11. Um die importierte `asset-viewer.war`-Datei auf Apache Tomcat 9,0 auszuführen, klicken Sie mit der rechten Maustaste auf die `asset-viewer.war`-Datei und wählen Sie **Ausführen als > auf dem Server ausführen** aus.
Der Dialog **Auf Server ausführen** wird geöffnet.

12. Klicken Sie auf **Fertig stellen**.
Die `asset-viewer.war` wird mit der Ausführung auf Apache Tomcat starten.
Stoppen Sie nach der Überprüfung des Setups den Server und importieren Sie das Plugin-Entwicklungsstartprojekt.

13. Führen Sie die folgenden Schritte aus, um Content Integration SDK zu installieren:

a. Löschen Sie in den folgenden Verzeichnissen die bereits installierten SDKs:

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

Unter UNIX oder Mac OS X <LOCAL_M2_REPOSITORY> verweist auf das ~/.m2/repository-Verzeichnis.

Unter Microsoft Windows <LOCAL_M2_REPOSITORY> verweist auf das C:\Users\{your-username}\.m2\repository-Verzeichnis.

b. Verwenden Sie die folgenden Befehle, um Content Integration SDKs in Ihrem lokalen Maven-Repository zu installieren. Suchen Sie asset-integration-api.jar, integration-api.jar, standard-integrations.jar und entity-mapper-api.jar im <ASSET_PICKER_HOME>/dev-kits/sdk-Verzeichnis.

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

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

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

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

14. Zum Importieren des Plugin-Entwicklungs-Starterprojekts wählen Sie **Datei > Importieren**aus.
    Das Dialogfeld **Auswählen** wird geöffnet.

15. Wählen Sie Bestehende Maven-Projekte aus und klicken Sie auf **Weiter**.
    Das Dialogfeld **Maven-Projekte** wird geöffnet.

16. Klicken Sie**Durchsuchen**, um das Projekt auszuwählen und dann auf **Fertigstellen**.

17. Um Maven-Abhängigkeiten des `asset-integration-starter`-Projekts zu aktualisieren, klicken Sie mit der rechten Maustaste auf das `asset-integration-starter`-Projekt und wählen Sie **Maven > Update Projekt**aus.

18. Stellen Sie sicher, dass das neu importierte Projekt Java 8 zum Kompilieren von Quellen verwendet. Öffnen Sie die Projekteigenschaften und führen Sie die folgenden Schritte zur Konfiguration des Compilers aus:
    a. Wählen Sie **Java Compiler**aus.
    b. Wenn die Konformitätsstufe des Compilers nicht editierbar ist, wählen Sie **Projektspezifische Einstellungen aktivieren**aus.
    c. Ändern Sie die Konformitätsstufe des Compilers auf `1,8`.
    d. Klicken Sie auf **Übernehmen und Schließen**.

19. Um sicherzustellen, dass die richtige Java-Bibliothek im Buildpfad eingerichtet ist, führen Sie die folgenden Schritte aus:

    a. Wählen Sie **Java-Buildpfads > Bibliotheken**aus.

    b. Wählen Sie die **JRE-Systembibliothek (J2SE 1,5)**aus.

    c. Klicken Sie auf **Entfernen**.

d. Klicken Sie auf **Bibliothek hinzufügen**.
   Das Dialogfeld **Bibliothek hinzufügen** wird geöffnet.

e. Wählen Sie die **JRE-Systembibliothek > Weiter**aus.
   Die **JRE-Systembibliothek** wird angezeigt.

f. Wählen Sie eine geeignete Bibliothek aus und klicken Sie auf **Fertigstellen**.

20. Um die Verarbeitung von Annotationen zu ermöglichen, führen Sie die folgenden Schritte aus:
    a. Wählen Sie **Java Compiler > Annotationsverarbeitung**aus.
    b. Wählen Sie **Projektspezifische Einstellungen aktivieren**.
    c. Wählen Sie **Übernehmen und Schließen**.

21. Führen Sie die folgenden Schritte aus, um Lombok zu installieren:

    a. Doppelklicken Sie auf die `LOCAL_M2_REPOSITORY\org\projectlombok\lombok\1.18.16\lombok-1.18.16.jar`.
       Das Installationsprogramm wird angezeigt.

    b. Klicken Sie auf **Position angeben**, um die Installationsposition Ihrer IDE anzugeben.

    c. Um die Installation abzuschließen, klicken Sie auf **Installieren / Aktualisieren**.

    d. Nach der Installation von Lombok müssen Sie die IDE erneut starten.

22. Gehen Sie wie folgt vor, um den Projektnamen zu ändern:
    a. Öffnen Sie die Datei `pom.xml` und ändern Sie die Projekteigenschaften von Maven.
    b. Klicken Sie mit der rechten Maustaste auf das Projekt "Asset-Integration-Starter" und wählen Sie **Refactor > Umbenennen** .

23. Deklarieren Sie in der `<ASSET_PICKER_HOME>/conf/custom-plugin-services.yml`-Datei die Plugin-Services. Sie können später auf diese Datei zugreifen, um bei der Einführung von Services für Ihre Plugins Deklarationen hinzuzufügen.

24. Wenn Sie ein Plugin-Projekt zur Bereitstellungsbaugruppe des `asset-viewer.war`-Projekts hinzufügen möchten, führen Sie die folgenden Schritte aus:

    a. Klicken Sie mit der rechten Maustaste das `asset-viewer.war`-Projekt an und wählen Sie **Eigenschaften** aus.
    Das Dialogfeld **Eigenschaften für den Asset-Viewer** wird geöffnet.

    b. Wählen Sie **Deployment Assembly** aus.

    c. Wählen Sie **Hinzufügen** aus.
    Das Dialogfeld **Richtlinientyp** auswählen wird geöffnet.

    d. Wählen Sie **Projekt** aus und klicken Sie auf **Weiter**.

    e. Wählen Sie das in den vorherigen Schritten importierte `asset-integration-starter`-Plug-in-Projekt aus und klicken Sie auf **Fertig stellen**.

25. Bereinigen Sie die Projekte bei Bedarf.

26. Nehmen Sie die entsprechende Konfiguration für Ihr System in `<ASSET_PICKER_HOME>/conf/systems.properties` vor (siehe `sample-systems.properties`-Datei im `<ASSET_PICKER_HOME>/dev-kits/asset-integration-starter`-Projekt). Alle im *Unica Content Integration-Administrationshandbuch* erwähnten System-Onboarding-Konfigurationen werden in `systems.properties` unter Verwendung relevanter Eigenschaften unterstützt.

27. Wenn Sie Ihr Plug-in entwickeln, aktivieren Sie es, indem Sie das `asset-viewer.war`-Projekt auf einem zuvor konfigurierten Anwendungsserver ausführen. Da das Projekt bereits der Deployment Assembly von `asset-viewer.war` hinzugefügt würde, werden Änderungen an Ihrem Plugin-Projekt implementiert, wenn Sie das `asset-viewer.war`-Projekt ausführen.

28. Verwenden Sie beim Entwickeln Ihres Plugins durch Hinzufügen von Diensten ein Werkzeug Ihrer Wahl, um die folgenden REST-Endpunkte zu erreichen (ändern Sie den Kontextstamm entsprechend Ihres Setups), um die Richtigkeit Ihrer Implementierung zu überprüfen:

a. **Systemvoraussetzungen sicherstellen**

Endpunkt-URL	`http://localhost:8888/asset-viewer/api/AssetPicker/instances`
Anforderungsmethode	`GET`

b. **Dienst Einfache Suche überprüfen.**

Endpunkt-URL	`http://localhost:8888/asset-viewer/api/AssetPicker/mysystem/assets?query=mountain&page=0&size=10&types=Phto` Dabei gilt: • `mysystem` stellt die von der Plug-in-Implementierung ausgewählte System-ID dar. • `query` enthält das Suchschlüsselwort für die Suche nach Inhalten. • `page & size` enthält Paginierungsdetails, wobei Seite die Seriennummer der abzurufenden Seiten und Größe die Gesamtzahl der Suchelemente auf einer einzelnen Seite ist. • Bei `types` handelt es sich um eine der unterstützten Inhaltskategorien (Typen) zum Filtern der Suchpositionen.
Anforderungsmethode	`GET`

Wenn Sie den URL treffen, stellen Sie sicher, dass die Antwort JSON das erwartete Ergebnis enthält. Für alle Suchelemente werden nur Präsentationsdetails

eingeschlossen. Andere Inhaltseigenschaften werden aus Gründen der Sicherheit und der Leistung ausgeschlossen.

c. **Resource Loader-Dienst überprüfen**

Endpunkt-URL	`http://localhost:8888/asset-viewer/api/AssetPicker/mysystem/download?resource=`
	`http://repository_base_url/contents/sample_image.jpg%26resourceId=12345"`
	Dabei gilt Folgendes
	• `mysystem` stellt die von der Plug-in-Implementierung ausgewählte System-ID dar.
	• `resource` enthält den absoluten URL-Inhalt, der heruntergeladen werden soll.
	• `resourceId` enthält die Kennung des herunterzuladenden Inhalts.
	(Plug-in kann entweder resource oder resourceId oder beides zum Laden des Inhalts verwenden.)

Anforderungsmethode	`GET`

d. **Listenordnerservice überprüfen**

Endpunkt-URL	`http://localhost:8888/asset-viewer/api/AssetPicker/mysystem/folders?parentFolderId=1234` Dabei gilt Folgendes: • `mysystem` stellt die von der Plug-in-Implementierung ausgewählte System-ID dar.

	• `parentFolderId` enthält die Kennung des übergeordneten Ordners, dessen unmittelbare Unterordner als Antwort erwartet werden. Dieser Abfrageparameter ist optional und wird beim Auflisten der Ordner auf oberster/ Stammebene nicht angegeben.
Anforderungsmethode	`GET`

e. **Listeninhaltsdienst überprüfen**

Endpunkt-URL	`http://localhost:8888/asset-viewer/api/AssetPicker/mysystem/folders/1234/contents` Dabei gilt Folgendes: • `mysystem` stellt die von der Plug-in-Implementierung ausgewählte System-ID dar. • `1234` steht für die ID des Ordners, dessen direkter Inhalt als Antwort erwartet wird.
Anforderungsmethode	`GET`

Für jeden vom `list-contents`-Dienst aufgelisteten Inhalt sind nur Präsentationsdetails enthalten. Andere Inhaltseigenschaften werden aus Gründen der Sicherheit und der Leistung ausgeschlossen.

f. **Dienst get-content-details überprüfen**

Endpunkt-URL	`http://localhost:8888/asset-viewer/api/AssetPicker/mysystem/assets/Images/1234` Dabei gilt Folgendes:

	• `mysystem` stellt die von der Plug-in-Implementierung ausgewählte System-ID dar. • `Images` stellt die Kategorie-ID des Inhalts dar, dessen Details als Antwort erwartet werden. • `1234` stellt die ID des Inhalts dar, dessen Details als Antwort erwartet werden.
Anforderungsmethode	`GET`

Die vom `get-content-details`-Dienst erzeugte JSON-Antwort enthält neben den Präsentationsdetails alle Inhaltseigenschaften.

g. **Dienst get-object-schema überprüfen**

Endpunkt-URL	`http://localhost:8888/asset-viewer/api/AssetPicker/object-mapping/application/mysystem/object/Images/1234/schema` Dabei gilt Folgendes: • mysystem stellt die von der Plug-in-Implementierung ausgewählte System-ID dar. • Bilder stellen die Kategorie des Referenzinhalts dar, der für die Schemaerstellung verwendet wird. • 1234 steht für die ID des Referenzinhalts, der für die Schemagenerierung verwendet wird. Ab 12.1.0.4 sind Inhaltskennung und Kategorie nicht mehr relevant, da erwartet wird, dass das Schema

	Attribute für alle unterstützten Inhaltskategorien enthält.
Anforderungsmethode	`GET`

Die JSON-Antwort muss die reduzierte Liste aller zuordnbaren Attribute und ihrer Metadaten enthalten.

h. **Listeninhaltskategorien-Dienst überprüfen**

Endpunkt-URL	`http://localhost:8888/asset-viewer/api/AssetPicker/mysystem/categories` Dabei gilt Folgendes: • `mysystem` stellt die von der Plug-in-Implementierung ausgewählte System-ID dar.
Anforderungsmethode	`GET`

i. **Dienst get-des-analysis überprüfen**

Endpunkt-URL	`http://localhost:8888/asset-viewer/api/AssetPicker/actions/cognize?url=absolute_image_url` Dabei gilt Folgendes: • `url` enthält die absolute URL des Bildes, für das die kognitive Analyse abgerufen werden soll
Anforderungsmethode	`GET`

Chapter 5. Überprüfung und Fehlerbehebung

Um die End-to-End-Integration zu überprüfen, platzieren Sie die JAR-Datei mit der Plugin-Implementierung im Klassenpfad des Anwendungsservers, auf dem die Inhaltsintegration bereitgestellt wird. Konfigurieren Sie außerdem das entsprechende Inhalts-Repository in der `<ASSET_PICKER_HOME>/conf/systems.properties`-Datei (Sie können auf die `sample-systems.properties`-Datei im `<ASSET_PICKER_HOME>/dev-kits/asset-integration-starter`-Projekt verweisen).

Alle im Unica Content Integration-Administrationshandbuch erwähnten System-Onboarding-Konfigurationen werden in `systems.properties` mit relevanten Eigenschaften unterstützt. Sie müssen das `-Dspring.profiles.active=platform-disintegrated`-JVM-Argument angeben, damit `systems.properties` wirksam wird (Sie können immer die Konfigurationen der Plattform anstelle von systems.properties verwenden, indem Sie das `-Dspring.profiles.active=platform-disintegrated`-JVM-Argument entfernen).

Note: Derzeit können nur Unica Centralized Offer Management und Unica Plan auf Content Integration zugreifen.

Starten Sie die Content Integration-Anwendung neu, nachdem das Plugin bereitgestellt und die Systemkonfigurationen vorgenommen wurden.

Obwohl Sie Content Integration mithilfe der im vorherigen Abschnitt genannten REST-Endpunkte überprüfen können, empfehlen wir Ihnen, die End-to-End-Integration zu überprüfen, indem Sie die entsprechende Benutzeroberfläche in Unica Centralized Offer Management und Unica Plan ausführen. Weitere Informationen zum Zugriff auf die Funktionender Content Integration in den jeweiligen Produkten finden Sie in den entsprechenden Benutzerhandbüchern.

Verwenden Sie bei Bedarf die von den unterstützten Browsern bereitgestellten Entwicklerwerkzeuge, um Fehler bei den API-Aufrufen zu beheben.

Übersicht der Logger (Protokollfunktionen)

Wie unter Überprüfung der Integration erwähnt, ist die Protokollierungskonfiguration für Content Integration in der Datei `log4j2.xml` verfügbar, die sich im Ordner `AssetPicker/conf/logging` innerhalb von Platform home befindet.

Content Integration verwendet Apache `Log4j2` für die Protokollverwaltung. Der `RandomAccessFilePlatform`-Appender steuert zusammen mit dem in `log4j2.xml` konfigurierten `com.unica`-Logger die von Platforms `unica-common.jar` und `unica-helper.jar` erstellten Protokolle, die bei der Inhaltsintegration verwendet werden. Die verbleibenden Einstellungen steuern die Protokollierung für andere zentrale Aktivitäten von Content Integration.

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 Content Integration 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.

Die `log4j2.xml`-Datei enthält auch die erforderlichen Konfigurationen, um alle Protokollierer für einen bestimmten Benutzer an eine dedizierte Protokolldatei weiterzuleiten. Standardmäßig werden diese Konfigurationen kommentiert. Eine entsprechende Beschreibung wird in `log4j2.xml` oben in jedem Konfigurationselement hinzugefügt, um die Aktivierung der dedizierten Protokolldatei zu erleichtern.

Nützliche Logger in der log4j2.xml-Datei

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

Table 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

Protokollfunktionen (Loggers)	Information
	Antwortdetails für alle eingehenden HTTP-Anforderungen an Content Integration. 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 Content Integration Framework und 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

Protokollfunktionen (Loggers)	Information
	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 Content Integration 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 Content Integration 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 Content Integration 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 Content Integration Framework identifiziert werden.

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

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

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

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

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

- **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...
```

- **Ungültigmachung des Plattformkonfigurationscaches und Neuinitialisierung des Dienstes** - Alle Plattformkonfigurationen für die Inhaltsintegration werden beim Start der Anwendung zwischengespeichert. Diese Konfigurationen werden nach bestimmten Intervallen aktualisiert (standardmäßig alle 30 Minuten, sofern nicht für die Verwendung eines anderen Intervalls konfiguriert). Die folgenden Logger werden immer dann auf der INFO-Ebene erstellt, wenn die Konfigurationsaktualisierung beginnt:

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

  In ähnlicher Weise werden die folgenden Zeilen auf INFO-Ebene generiert, wenn es beendet ist:

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