



OpenText™ Documentum™ Content Management

Foundation REST API Development Guide

Build enterprise content management applications or Web APIs.

EDCPKRST250400-PGD-EN-01

**OpenText™ Documentum™ Content Management
Foundation REST API Development Guide**
EDCPKRST250400-PGD-EN-01
Rev.: 2025-Oct-18

This documentation has been created for OpenText™ Documentum™ Content Management CE 25.4.
It is also valid for subsequent software releases unless OpenText has made newer documentation available with the product,
on an OpenText website, or by any other means.

Open Text Corporation

275 Frank Tompa Drive, Waterloo, Ontario, Canada, N2L 0A1

Tel: +1-519-888-7111
Toll Free Canada/USA: 1-800-499-6544 International: +800-4996-5440
Fax: +1-519-888-0677
Support: <https://support.opentext.com>
For more information, visit <https://www.opentext.com>

© 2025 Open Text

Patents may cover this product, see <https://www.opentext.com/patents>.

Disclaimer

No Warranties and Limitation of Liability

Every effort has been made to ensure the accuracy of the features and techniques presented in this publication. However,
Open Text Corporation and its affiliates accept no responsibility and offer no warranty whether expressed or implied, for the
accuracy of this publication.

Table of Contents

1	Overview	7
1.1	Understanding RESTful programming	8
1.2	Relations with other OpenText Documentum CM APIs	8
2	General REST definitions	9
2.1	Common HTTP headers	9
2.2	Common query parameters	10
2.3	HTTP status codes	14
2.4	Supported MIME types	16
2.5	URI and URL	18
2.6	HTTP methods	19
2.7	Web client caching	19
2.8	Representation	20
2.9	The PUT and POST operations	31
2.10	Runtime property configuration	31
2.11	Batch operations	32
3	Deploy Foundation REST API	33
3.1	General deployment configuration	33
4	Resource specific features	35
4.1	Create an object with an attached Aspect and Lifecycle	35
4.2	Filter expression	37
4.3	Property view	49
4.4	NULL in REST	51
4.5	Whitespace in XML	53
4.6	Thumbnail link	53
4.7	Support for relative URIs	56
4.8	Feed pagination	59
4.9	Full text query in collection resources	60
4.10	Simple search language	60
4.11	Facet search	62
4.12	Lightweight and shareable objects	66
4.13	Configuring DQL resource access modes	68
4.14	Generating link relation in DQL results	70
4.15	Location of persistent data	74
4.16	MailApp support	77
4.17	Constraints validation of objects	78
4.18	Adding a document with content to Foundation REST API	79
5	Authentication	83

5.1	End user tracking	83
5.2	HTTP basic authentication	84
5.3	OAuth 2.0 authentication	87
5.4	Pre-authenticated authentication	100
5.5	Reverse proxy configuration	104
5.6	Client token	106
5.7	Auditing the Foundation REST API client events	111
5.8	Bypassing browser login forms upon HTTP 401 unauthorized	113
6	Using Kubernetes	115
7	Resource extensibility	117
7.1	Overview	117
7.2	Deprecated Java APIs	118
7.3	Get started with the development kit	119
7.4	Architecture of extensible Foundation REST API	125
7.5	Foundation REST API marshalling framework	130
7.6	Developing custom resources	196
7.7	Working with YAML configuration	231
7.8	Working with HAL	231
7.9	Registering URI templates	244
7.10	Normalization of custom URI templates	249
7.11	Registering root resources	251
7.12	Adding links to core resources	255
7.13	Disabling specific resources	257
7.14	Overriding specific resources	262
7.15	Turning off XML, JSON, or HAL media types	265
7.16	Creating custom error code mapping files	267
7.17	Creating custom message files	268
7.18	HTTP compression	268
7.19	Tutorial: Foundation REST API extensibility development	272
7.20	OpenText Documentum Content Management (CM) Accelerated Content Services and OpenText Documentum Content Management (CM) Branch Office Caching Services content upload ..	289
7.21	Conditional Request for Foundation REST API server	300
7.22	Using the AppWorks Gateway to integrate the Foundation REST API	306
7.23	Checksum for Foundation REST APIs	310
8	Authentication extensibility	313
8.1	Anonymous access	313
8.2	Generic servlet filter customization	315
8.3	Using Foundation REST API as the Authentication Handler for the AppWorks Gateway	317

8.4	Custom authentication development	319
9	Advanced security configuration	347
9.1	Default security Headers	347
9.2	CSRF Protection	352
9.3	Cross-Origin Resource Sharing (CORS) support	356
9.4	Request sanitization	358
10	Explore Foundation REST API	363
10.1	Prerequisites	363
10.2	Common tasks on folders, documents, and contents	363
10.3	Create a document and content with two Requests	375
10.4	Creating a document and content with a single Request	378
10.5	Updating content or renditions	379
10.6	Deleting content or renditions	380
11	Tutorial: Consume Foundation REST API programmatically	381
11.1	Basic navigation	381
11.2	Read entries	384
11.3	Filter, sort, and pagination	385
11.4	Create entries	386
11.5	Update entries	386
11.6	Delete entries	387
A	Link relations	389
B	Resource coding index	395

Chapter 1

Overview

This guide is intended for developers and architects who are building applications or Web APIs that consume OpenText Documentum Content Management (CM) Foundation REST API. Foundation REST API is a set of RESTful web service interfaces that interact with OpenText™ Documentum™ Content Management.

Developed in a purely RESTful style, makes Foundation REST API hypertext-driven, server-side stateless, and content negotiable. This provides you with a high degree of efficiency and simplicity in development all while making its resources easy to consume. These advantages make OpenText Documentum Content Management (CM) Foundation REST API the optimal choice for Web 2.0 applications, Web APIs, and mobile applications that interact with OpenText Documentum CM repositories.

Foundation REST API models objects in OpenText Documentum CM repositories as resources and identifies resources using their Uniform Resource Identifiers (URIs). It defines specific media types to represent its resources and drives application state transfers by using link relations. It uses standard HTTP methods such as GET, PUT, POST, and DELETE to manipulate its resources using the HTTP protocol.

Foundation REST API supports the following formats for resource representation:

- XML
- JSON
- JSON HAL (Hypertext Application Language)



Note: The following are the terminologies used and their explanation:

- Foundation REST API MVC – Built on top of Spring MVC, Foundation REST API MVC is a framework that facilitates the custom resource development in Foundation REST API. For more information, see [Foundation REST API MVC](#).
- Core resources – Any resources that are shipped with Foundation REST API out of the box. [Appendix C](#) provides a complete list of Core resources. Additionally, the *OpenText Documentum Content Management - Foundation REST API Reference Guide (EDCPKRST250400-PRE)* introduces each Core resource in detail.

1.1 Understanding RESTful programming

Foundation REST API delivers a deployable Java web archive (WAR) file that runs in the web container of a Java EE application server (refer to the release notes for system requirements). The WAR file exposes the interface as network-accessible resources identified by URIs.

Foundation REST API is programming language independent. Therefore, you can consume its services using any programming language that has an HTTP client library, such as Java, .NET, Python, Ruby, and so on.

You can access the source code of Foundation REST API sample clients on GitHub website. These client samples help demonstrate some critical aspects and programming techniques that can be used to build applications and Web APIs that consume Foundation REST API.



Note: The source code of these Foundation REST API client samples is made available to the public through the Apache 2 license.

1.2 Relations with other OpenText Documentum CM APIs

Foundation REST API relies on the OpenText™ Documentum™ Content Management Foundation Java API library to communicate with OpenText™ Documentum™ Content Management Server. Therefore, communication between the Foundation REST API server and OpenText Documentum Content Management (CM) Server is conducted using Netwise RPC. Foundation REST API is a lightweight alternative to the existing OpenText Documentum CM Web APIs, such as WDK and OpenText™ Documentum™ Content Management Foundation SOAP API. However, it is not intended to provide equivalent functionalities to the existing OpenText Documentum CM Web APIs. It can allow you to leverage the simplicity of RESTful services to achieve high productivity in software development.

Chapter 2

General REST definitions

2.1 Common HTTP headers

Foundation REST API supports the following common HTTP Headers:

HTTP Header name	Description	In request or Response?	Value range
Authorization	Authorization Header for authentication	Request	HTTP basic authentication Header with the credential part encoded, for example: Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
Accept	Acceptable media type for the Response body	Request	See " Supported MIME types " on page 16
Content-Type	MIME type of the Request body or Response body	Request/Response	When performing a POST operation that contains non UTF-8 characters in the Request body, the server displays the following error: E_INPUT_MESSAGE_NOT_READABLE This is because the Foundation REST API server ignores the charset parameter in the Content-Type Header. For more information, see " Supported MIME types " on page 16.

HTTP Header name	Description	In request or Response?	Value range
Content-Length	Size of the entity-body, in decimal number of OCTETS sent to the recipient	Request/Response	Any number greater than or equal to 0 (zero). Negative numbers are not valid values
Location	URI of the newly-created resource	Response	URI
Set-Cookie	Sets an HTTP cookie Response	Response	Used by client token
WWW-Authenticate	Indicates the authentication scheme that should be used to access the requested entity	Response	Used by HTTP basic authentication
ETag	ETag value generated by the Foundation REST API server	Response	String value
If-None-Match	Checks whether the resource is changed	Request	ETag value in the previous server's Response
Last-Modified	Last modified time of the resource	Response	HTTP-Date
If-Modified-Since	Checks whether the resource is changed since last modified time	Request	Last-Modified value in the previous server's Response

2.2 Common query parameters

Foundation REST API supports the following common query parameters:



Note: Each resource has a limited number of supported parameters. For more information on a resource's supported query parameters, see the resource.

Query parameter name	Description	Data type	Value range	Default value
inline	Determines whether or not to show content (the object instance) in an atom entry or EDAA entry for a collection	boolean	<ul style="list-style-type: none"> • <i>true</i> - Returns the object instance and embeds the object instance into the entry's content element • <i>false</i> - Does not return object instance 	<p>false</p> <p> Note: Although the default value is <i>false</i>, we recommend that you set this parameter to <i>true</i> in your development environment so you can get the entire contents of resources. This allows you to explore the complete data structure</p> <p>In your production environment, you can set this parameter to <i>false</i> for better performance.</p>
items-per-page	Specifies the number of items to be rendered on one page	Integer	Any integer not less than 1	100

Query parameter name	Description	Data type	Value range	Default value
Page	<p>Specifies the page number of the page to return</p> <p>For example, if you set <code>items-per-page</code> to 200, and <code>page</code> to 2, the operation returns items 201 to 400</p>	Integer	Any integer not less than 1	1
view	<p>Specifies the object properties to retrieve.</p> <p> Caution This parameter works only when <code>inline</code> is set to <code>true</code></p>	String	See "Property view" on page 49.	default
include-total	Determines whether or not to return the total number of objects. For paged feeds, objects in all pages are counted	boolean	<ul style="list-style-type: none"> <code>true</code> - return the total number of objects. <code>false</code> - do not return the total number of objects. 	false

Query parameter name	Description	Data type	Value range	Default value
sort	<p>Specifies a set of the sort specifications in a returned collection</p>	String	<p>This parameter consists of multiple sort specifications, separated by comma (,).</p> <p>Each sort specification consists of a property (any non-repeating property) by which to sort the results and its sort order (DESC or ASC), separated by the whitespace character ().</p> <p>The sort order is optional. If not specified, the default sort order is ASC.</p> <p>If any property with an invalid name is specified, an error is thrown.</p> <p>Example:</p> <pre>sort=r_modify_date desc,object_id asc,title</pre>	NA
links	<p>Determines whether or not to return link relations in the object representation</p> <p>This parameter works only when <code>inline</code> is set to <code>true</code></p>	boolean	<ul style="list-style-type: none"> • <i>true</i> - return link relations. • <i>false</i> - do not return link relations. 	true

Query parameter name	Description	Data type	Value range	Default value
recursive	Determines whether or not to return all indirect children recursively when a request tries to get the children of an object	boolean	<ul style="list-style-type: none"> <i>true</i> - return all indirect children recursively when a request tries to get the children of an object. <i>false</i> - Only return direct children when a request tries to get the children of an object. 	false
filter	Filter expression	String	"Filter expression" on page 37 provides more details.	NA
q	Specifies full text search criterion when sending a GET request to the Search resource or certain collection resources	String	String that follows the simple search language syntax.	NA

2.3 HTTP status codes

Foundation REST API supports the following HTTP status codes:

Status code	Description
200 OK	The Request has succeeded. The information returned with the Response is dependent upon the method used in the Request, for example: GET an entity corresponding to the requested resource is sent in the Response; PUT an entity describing or containing the modified resource.

Status code	Description
201 Created	The Request has been fulfilled and resulted in a new resource being created. The newly created resource can be referenced by the URI(s) returned in the entity of the Response, with the most specific URI for the resource given by a Location Header field. The entity format is specified by the media type given in the Content-Type header field.
204 No Content	The server has fulfilled the Request but does not need to return an entity-body.
304 Not Modified	The server responds with this status code when the client has performed a conditional GET request and access is allowed, but the document has not been modified.
400 Bad Request	The Request could not be understood by the server due to malformed syntax, missing or invalid information (such as a validation error on an input field, or a missing required value). The client should not repeat the Request without modifications.
401 Unauthorized	The authentication credentials included with this request are missing or invalid. The Response must include a WWW-Authenticate Header field containing a challenge applicable to the requested resource.
403 Forbidden	The server has recognized your credentials, but you do not possess the authorization to perform this request, and the Request should not be repeated.
404 Not Found	The Request specifies a URI of a resource that does not exist.
405 Method Not Allowed	The HTTP verb specified in the Request (DELETE, GET, HEAD, POST, PUT) is not allowed for the resource identified by the Request URI.
406 Not Acceptable	The resource identified by this request URI is not capable of generating a representation corresponding to one of the media types in the Accept Header of the Request.
409 Conflict	The Request could not be completed, because it would cause a conflict in the current state of the resources supported by the server (for example, an attempt to create a new resource with a unique identifier already assigned to some existing resource).

Status code	Description
415 Unsupported Media Type	The server is refusing to service the Request, because the entity of the Request is in a format not supported by the requested resource for the Requested method.
500 Internal Server Error	The server encountered an unexpected condition, which prevented it from fulfilling the Request.

2.4 Supported MIME types

Adhering to RFC 2616, section 14 for content negotiation, Foundation REST API supports XML and JSON representations with the following MIME types:

MIME type	Usage
application/vnd.emc.documentum+json (default MIME type)	JSON representation
application/atom+xml	XML representation for feeds (collections)
application/vnd.emc.documentum+xml	XML representation for a single object
application/json	JSON representation for compatible viewing
application/xml	XML representation for compatible viewing



Notes

- When the client does not specify the *Accept* Header for the expecting Response, the Foundation REST API server uses the application/vnd.emc.documentum+json MIME type by default.
- When the client does not specify the *Content-Type* Header for a PUT or POST Request, the Foundation REST API server rejects the Request with the HTTP status code 415.
- In some scenarios, the MIME types that Foundation REST API supports are not limited to the types in this table. For more information about other supported types, see “[Other types](#)” on page 16.

2.4.1 Other types

2.4.1.1 Media type for home document

The Home Document resource supports the following media types:

- application/home+json
- application/home+xml

2.4.1.2 MIME type for content

For the content import or export operations, the Foundation REST API server accepts any MIME type registered to the `dm_format` table in a repository. For example, when the Foundation REST API client imports a new PDF rendition for a document object, the *Content-Type* in the Request body can be `application/pdf`.

2.4.1.3 Multipart type

Foundation REST API supports the `multipart/form` and `multipart/mixed` type in SysObject contents importing, and the `multipart/related` type for batch operations with content attachments. For more information about multipart type, see RFC 2616.

2.4.2 URL extension

Foundation REST API, allows you to use the `.xml` and `.json` URL extensions to negotiate the content type of the Response. For example:

- `/repositories.xml`: Returns a collection of repositories in an ATOM XML feed representation.
- `/repositories/acme01.xml`: Returns the repository acme01 in an XML representation.
- `/repositories.json`: Returns a collection of repositories in a JSON representation.
- `/repositories/acme01.json`: Returns the repository acme01 in a JSON representation.

When you use a URL extension in a GET operation, the URL extension takes precedence over the *Accept* Header. The content type of the return varies with the URL extension as follows:

URL extension	Content type of return
<code>.json</code>	<code>application/json</code>
<code>.xml</code>	<code>application/xml</code>
Other extension	<p>The <i>Accept</i> Header is used for content negotiation</p> <p>For more information about the <i>Accept</i> Header and content negotiation, see RFC 2616.</p>

URL extensions in POST and PUT operations are ignored. The Foundation REST API server reads the media type from the *Content-Type* Header.

2.4.3 Content type of an entry in a feed

When an *inline* parameter is set to *false* as an operation tries to retrieve a feed, the operation returns the content type for an entry. The entry's content type is determined by the content type of the feed that contains the entry.

For details, see the following table:

Feed content type	Entry content type
application/atom+xml	application/vnd.emc.documentum+xml
application/xml	application/xml
application/vnd.emc.documentum+json	application/vnd.emc.documentum+json
application/json	application/json

2.5 URI and URL

A URI is a compact sequence of characters that can identify an abstract or physical resource. In Foundation REST API, URIs are used as the only resource identifiers.

In Foundation REST API, a uniform resource locator (URL) is used to locate a resource. The URL is designed in a pattern that is only known to and interpretable by the Foundation REST API server. Foundation REST API clients cannot parse or concatenate the URL for a resource by using the object ID or name. All Foundation REST API clients can follow the link relations in a resource representation to locate related resources.

URLs in the resource representations are in the form of an absolute path. When the Foundation REST API server is deployed behind a reverse proxy server or a load balancer, you must configure the proxy rules to replace the backend hostname with the front hostname.

For example, replace the following reverse proxy configuration in the Apache HTTP server maps `http://internal-node1.acme.com:8080/dctm-rest` to `http://reverse-proxy-server:80`.

```
ProxyPass / http://internal-node1.acme.com:8080/dctm-rest/
ProxyPassReverse / http://internal-node1.acme.com:8080/dctm-rest/

<Location>
AddOutputFilterByType SUBSTITUTE application/xml
Substitute "s|internal-node1.acme.com:8080/dctm-rest| reverse-proxy-server |i"
</Location>
<Location>
AddOutputFilterByType SUBSTITUTE application/json
Substitute "s|internal-node1.acme.com:8080/dctm-rest| reverse-proxy-server |i"
</Location>
<Location>
AddOutputFilterByType SUBSTITUTE application/atom+xml
Substitute "s|internal-node1.acme.com:8080/dctm-rest| reverse-proxy-server |i"
```

```

</Location>
<Location/>
AddOutputFilterByType SUBSTITUTE application/vnd.emc.documentum+xml
Substitute "s|internal-node1.acme.com:8080/dctm-rest| reverse-proxy-server |i"
</Location>

<Location/>
AddOutputFilterByType SUBSTITUTE application/vnd.emc.documentum+json
Substitute "s|internal-node1.acme.com:8080/dctm-rest| reverse-proxy-server |i"
</Location>

<Location/>
AddOutputFilterByType SUBSTITUTE text/html
Substitute "s|internal-node1.acme.com:8080/dctm-rest| reverse-proxy-server |i"
</Location>

```

For more information about URI encoding in Foundation REST API, see the RFC 3986.

2.6 HTTP methods

Foundation REST API supports the following HTTP methods:

- GET: Use this method to retrieve a representation of a resource.
- POST: Use this method to create new resources, or update existing resources.
- PUT: Use this method to update existing resources.
- PUT: Use this method to delete a resource.



Note: Foundation REST API does not support the <OPTIONS> method except in Cross-Origin Resource Sharing preflight Requests. For more information, see the section titled *Cross-Origin Resource Sharing (CORS) support* in this guide.

Do not use this method to request a list of available operations from a resource. The *OpenText Documentum Content Management - Foundation REST API Reference Guide (EDCPKRST250400-PRE)* provides detailed information about the operations that can be performed on each resource.

2.7 Web client caching

Caching is one of the most useful features built on the top of the HTTP's uniform interface. HTTP clients can take advantage of caching to reduce the perceived latency to users, increase reliability, reduce bandwidth usage and cost, and reduce server load. Foundation REST API supports web client caching on the content media resource using an HTTP ETag token.

2.8 Representation

Foundation REST API supports two representation formats, JSON and XML.

JSON is the primary format for resource representation. Collection-based resources are presented as EDAA, which is a JSON representation of an Atom feed.

For more information, see *OpenText Documentum Content Management Foundation Java API Data Access API*.

JSON supports basic data types. Therefore, OpenText Documentum CM properties are mapped to JSON data types as shown in the following table:

Table 2-1: Mapping between OpenText Documentum CM property data type and JSON data type

OpenText Documentum CM property data type	JSON data type
boolean	boolean
Integer	Number
Double	Number
String	String
ID	String
Time	String
Repeating	Array

XML is the other format for resource representation in Foundation REST API. Like the JSON format, collection-based resources are presented as feeds, defined by Atom.

For more information about ATOM, see RFC 4287.

2.8.1 Collection resource

In XML and JSON formats, items (object instances) in the collection are represented as entries containing metadata and links in the feed. By default, the detail of an object instance is not presented in the entry body. Instead, a content `src` link points to the single instance resource. Alternatively, the object instance can be embedded in the entry by enabling `inline`.

The metadata of a feed consists of the following elements/properties:

Table 2-2: Metadata of a feed

Feed metadata	Description
id	URI of the collection resource without the file extension
title	Feed Title
updated	Last update time of the feed
author	Feed author
self	URI of the collection resource with the file extension

The metadata of an entry consists of the following elements/properties:

Table 2-3: Metadata of an entry

Feed metadata	Description
id	URI of the single resource without the file extension.
title	Entry title.
updated	Last update time of the entry.
author	Entry author.
summary	Entry description.
content	When the entry is not inline, it contains a <code>src</code> attribute whose value is the URI of the single item resource. When the entry is inline, it embeds the full representation of the single item resource. For more information, see " "Embedded entry" on page 23 ". When you set <code>inline</code> to <code>false</code> , <code>content-type</code> in JSON and <code>type</code> in XML indicate the MIME type of the resource.
edit	URI of the single resource with the file extension

The following sample illustrates a feed (EDAA feed) of the `Cabinets` resource in JSON. Note that only a URI of the single resource is presented in this sample as `inline` is not enabled.

```
{  
    id: "http://core-rs-demo.lss.emc.com:8080/dctm-rest/repositories/acme01/cabinets" ← Feed Id  
    title: "Cabinets" ← Feed Title  
    updated: "2013-06-19T15:14:39.679+08:00" ← Feed Updated  
    -authors: [1] ← Feed Authors  
        -0: {  
            name: "EMC Documentum"  
        }  
    -links: [1] ← Feed Links  
        -0: {  
            rel: "self"  
            href: "http://core-rs-demo.lss.emc.com:8080/dctm-rest/repositories/acme01/cabinets"  
        }  
    -entries: [1] ← Feed Entries  
        -0: {  
            id: "http://core-rs-demo.lss.emc.com:8080/dctm-rest/repositories/acme01/objects/0c0004d280000104" ← Entry Id  
            title: "acme01" ← Entry Title  
            updated: "2012-10-15T15:27:30.000+08:00" ← Entry Updated  
            summary: "dm_cabinet 0c0004d280000104"  
            -authors: [1] ← Entry Authors  
                -0: {  
                    name: "acme01"  
                    uri: "http://core-rs-demo.lss.emc.com:8080/dctm-rest/repositories/acme01/users/acme01"  
                }  
            -content: { ← Entry Content, pointing to the JSON format of cabinet resource. Alternately,  
                content-type: "application/json" full representation of the cabinet can be embedded within entry content  
                src: "http://core-rs-demo.lss.emc.com:8080/dctm-rest/repositories/acme01/cabinets/0c0004d280000104"  
            }  
            -links: [1] ← Entry Links  
                -0: {  
                    rel: "edit"  
                    href: "http://core-rs-demo.lss.emc.com:8080/dctm-rest/repositories/acme01/cabinets/0c0004d280000104"  
                }  
        }  
    -1: { ... }  
}
```

The following sample illustrates a feed of the Cabinets resource in XML. Note that only a URI of the single resource is presented in this sample as inline is not enabled.

```

{
  "id": "http://core-rs-demo.lss.emc.com:8080/dctm-rest/repositories/acme01/cabinets", ← Feed Id
  "title": "Cabinets", ← Feed Title
  "updated": "2013-06-19T15:14:39.679+00:00", ← Feed Updated
  "author": [{}], ← Feed Authors
    "0": {
      "name": "EMC Documentation"
    }
  "links": [{}], ← Feed Links
    "0": {
      "rel": "self",
      "href": "http://core-rs-demo.lss.emc.com:8080/dctm-rest/repositories/acme01/cabinets"
    }
  "entries": [{}], ← Feed Entries
    "0": {
      "id": "http://core-rs-demo.lss.emc.com:8080/dctm-rest/repositories/acme01/objects/0c0004d280000104", ← Entry Id
      "title": "acme01", ← Entry Title
      "updated": "2012-10-15T15:27:30.000+08:00", ← Entry Updated
      "summary": "dm_cabinet 0c0004d280000104"
      "author": [{}], ← Entry Authors
        "0": {
          "name": "acme01"
        }
      "uri": "http://core-rs-demo.lss.emc.com:8080/dctm-rest/repositories/acme01/users/acme01"
    }
    "content": { ← Entry Content, pointing to the JSON format of cabinet resource. Alternately,
      "content-type": "application/json", ← full representation of the cabinet can be embedded within entry content
      "src": "http://core-rs-demo.lss.emc.com:8080/dctm-rest/repositories/acme01/cabinets/0c0004d280000104"
    }
    "links": [{}], ← Entry Links
      "0": {
        "rel": "edit",
        "href": "http://core-rs-demo.lss.emc.com:8080/dctm-rest/repositories/acme01/cabinets/0c0004d280000104"
      }
    }
  "1": {...}
}

```

2.8.2 Embedded entry

For collection resources, when the `inline` parameter is set to `true`, a full representation of the object instance is embedded in the `content` element of the entry.

The following sample illustrates an entry that contains a full representation of `Cabinet` in JSON:

```

"entries": [
  {
    "id": "http://localhost:8080/dctm-rest/repositories/acme01/
          objects/0c0004d280000104",
    "title": "acme01",
    "updated": "2012-10-15T15:27:30.000+08:00",
    "author": [
      {
        "name": "acme01",
        "uri": "http://localhost:8080/dctm-rest/repositories/acme01/
              users/61636d653031"
      }
    ],
    "content": {
      "name": "cabinet",
      "type": "dm_cabinet",
      "definition": "http://localhost:8080/dctm-rest/repositories/
                     acme01/types/dm_cabinet",
      "properties": {
        "r_object_id": "0c0004d280000104",
        "object_name": "acme01",
        "title": "Super User Cabinet",
        "subject": "",
        "resolution_label": ""
      }
    }
  }
]

```

```

    "owner_name": "acme01",
    "owner_permit": 7,
    "group_name": "docu",
    "group_permit": 5,
    "world_permit": 3,
    "log_entry": "",
    "acl_domain": "acme01",
    "acl_name": "dm_450004d280000100",
    "language_code": "",
    "r_object_type": "dm_cabinet",
    "r_creation_date": "2012-10-15T15:27:30.000+08:00",
    "r_modify_date": "2012-10-15T15:27:30.000+08:00",
    "a_content_type": "",
    "authors": null,
    "r_lock_owner": "",
    "i_antecedent_id": "0000000000000000",
    "i_chronicle_id": "0c0004d280000104",
    "i_folder_id": null,
    "i_cabinet_id": "0c0004d280000104"
},
"links": [
{
    "rel": "self",
    "href": "http://localhost:8080/dctm-rest/repositories/acme01/
        cabinets/0c0004d280000104"
},
{
    "rel": "edit",
    "href": "http://localhost:8080/dctm-rest/repositories/acme01/
        cabinets/0c0004d280000104"
},
{
    "rel": "http://identifiers.emc.com/documentum/linkrel/delete",
    "href": "http://localhost:8080/dctm-rest/repositories/acme01/
        cabinets/0c0004d280000104"
},
{
    "rel": "http://identifiers.emc.com/documentum/linkrel/folders",
    "href": "http://localhost:8080/dctm-rest/repositories/acme01/
        folders/0c0004d280000104/folders"
},
{
    "rel": "http://identifiers.emc.com/documentum/linkrel/documents",
    "href": "http://localhost:8080/dctm-rest/repositories/acme01/
        folders/0c0004d280000104/documents"
},
{
    "rel": "http://identifiers.emc.com/documentum/linkrel/objects",
    "href": "http://localhost:8080/dctm-rest/repositories/acme01/
        folders/0c0004d280000104/objects"
},
{
    "rel": "http://identifiers.emc.com/documentum/linkrel/child-links",
    "href": "http://localhost:8080/dctm-rest/repositories/acme01/
        folders/0c0004d280000104/child-links"
},
{
    "rel": "http://identifiers.emc.com/documentum/linkrel/relations",
    "href": "http://localhost:8080/dctm-rest/repositories/acme01/
        relations?related-object-id=0c0004d280000104&
        related-object-role=any"
}
],
"links": [
{
    "rel": "edit",
    "href": "http://localhost:8080/dctm-rest/repositories/acme01/
        cabinets/0c0004d280000104"
}
]
}

```

```

        "thumbnail": {
            "url": "http://localhost:8081/thumbsrv/getThumbnail?object_type=dm_cabinet
&format=&is_vdm=false&repository=1234"
        }
    }
]

```

The following sample illustrates an entry that contains a full representation of *Cabinet* in XML:

```

<entry>
  <id>
    http://localhost:8080/dctm-rest/repositories/acme01/objects/
    0c0004d280000104
  </id>
  <title>acme01</title>
  <updated>2012-10-15T15:27:30.000+08:00</updated>
  <author>
    <name>acme01</name>
    <uri>
      http://localhost:8080/dctm-rest/repositories/acme01/users/
      61636d653031
    </uri>
  </author>
  <content>
    <dm:cabinet xsi:type="dm:dm_cabinet"
      definition="http://core-rs-demo.lss.emc.com:8080/
      dctm-rest/repositories/acme01/types/dm_cabinet">
      <dm:properties xsi:type="dm:dm_cabinet-properties">
        <dm:r_object_id>0c0004d280000104</dm:r_object_id>
        <dm:object_name>acme01</dm:object_name>
        <dm:title>Super User Cabinet</dm:title>
        <dm:subject/>
        <dm:resolution_label/>
        <dm:owner_name>acme01</dm:owner_name>
        <dm:owner_permit>7</dm:owner_permit>
        <dm:group_name>docu</dm:group_name>
        <dm:group_permit>5</dm:group_permit>
        <dm:world_permit>3</dm:world_permit>
        <dm:log_entry/>
        <dm:acl_domain>acme01</dm:acl_domain>
        <dm:acl_name>dm_450004d280000100</dm:acl_name>
        <dm:language_code/>
        <dm:r_object_type>dm_cabinet</dm:r_object_type>
        <dm:r_creation_date>2012-10-15T15:27:30.000+08:00</dm:r_creation_date>
        <dm:r_modify_date>2012-10-15T15:27:30.000+08:00</dm:r_modify_date>
        <dm:a_content_type/>
        <dm:authors xsi:nil="true"/>
        <dm:r_lock_owner/>
        <dm:i_antecedent_id>0000000000000000</dm:i_antecedent_id>
        <dm:i_chronicle_id>0c0004d280000104</dm:i_chronicle_id>
        <dm:i_folder_id xsi:nil="true"/>
        <dm:i_cabinet_id>0c0004d280000104</dm:i_cabinet_id>
      </dm:properties>
      <dm:links>
        <dm:link rel="self"
          href="http://localhost:8080/dctm-rest/repositories/
          acme01/cabinets/0c0004d280000104" />
        <dm:link rel="edit"
          href="http://localhost:8080/dctm-rest/repositories/
          acme01/cabinets/0c0004d280000104" />
        <dm:link rel="http://identifiers.emc.com/documentum/linkrel/delete"
          href="http://localhost:8080/dctm-rest/repositories/acme01/
          cabinets/0c0004d280000104" />
        <dm:link rel="http://identifiers.emc.com/documentum/linkrel/folders"
          href="http://localhost:8080/dctm-rest/repositories/acme01/
          folders/0c0004d280000104/folders" />
        <dm:link rel="http://identifiers.emc.com/documentum/linkrel/documents"
          href="http://localhost:8080/dctm-rest/repositories/
          acme01/folders/0c0004d280000104/documents" />
      </dm:links>
    </dm:cabinet>
  </content>
</entry>

```

```
<dm:link rel="http://identifiers.emc.com/documentum/linkrel/objects"
         href= "http://localhost:8080/dctm-rest/repositories/
acme01/folders/0c0004d280000104/objects" />
<dm:link rel="http://identifiers.emc.com/documentum/linkrel/child-links"
href= "http://localhost:8080/dctm-rest/repositories/
acme01/folders/0c0004d280000104/child-links" />
<dm:link rel="http://identifiers.emc.com/documentum/linkrel/relations"
         href= "http://localhost:8080/dctm-rest/repositories/acme01/
relations?related-object-id=0c0004d280000104&
related-object-role=any" />
</dm:links>
</dm:cabinet>
</content>
<link rel="edit"
      href= "http://localhost:8080/dctm-rest/repositories/acme01/cabinets/
0c0004d280000104" />
<media:thumbnail url="http://localhost:8081/thumbsrv/getThumbnail?object_type=
dm_cabinet&format=&is_vdm=false&repository=1234" />
</entry>
```

2.8.3 Single resource

The representation of a single resource consists of the following elements/properties:

Table 2-4: Elements and properties in a single resource

Element in XML	Property in JSON	Description
root	name	Category of the resource
xsi:type	type	Type name
definition	definition	URI pointing to the DML representation of the type
properties	properties	Properties of the resource
links	links	Link relations of the resource

The following sample illustrates the *Cabinet* resource in JSON:

```

{
  name: "cabinet" ← name property as the resource category
  type: "dm_cabinet" ← Object Type
  definition: "http://core-rs-demo.lss.emc.com:8080/dctm-rest/repositories/acme01/types/dm_cabinet" ← Object Type Resource
  -properties: {
    r_object_id: "0c0004d280000104" ← Object Properties
    object_name: "acme01" ← Single Property
    r_object_type: "dm_cabinet"
    title: "Super User Cabinet"
    subject: ""
    i_vstamp: 0
    i_ancestor_id: []
      0: "0c0004d280000104"
    is_private: false
    -r_folder_path: [1] ← Repeating Property
      0: "/acme01"
  }
  -links: [0] ← Links
    -0: {
      rel: "self"
      href: "http://core-rs-demo.lss.emc.com:8080/dctm-rest/repositories/acme01/cabinets/0c0004d280000104"
    }
    -1: {
      rel: "edit"
      href: "http://core-rs-demo.lss.emc.com:8080/dctm-rest/repositories/acme01/cabinets/0c0004d280000104"
    }
  }
}

```

The following sample illustrates the *Cabinet* resource in XML.

```

<?xml version="1.0" encoding="UTF-8"?> ← Object Type
<cabinet xsi:type="dm_cabinet" definition="http://core-rs-demo.lss.emc.com:8080/dctm-rest/repositories/acme01/types/dm_cabinet"> ← Object Type Resource
  <properties xsi:type="dm_cabinet-properties"> ← Object Properties
    <r_object_id>0c0004d280000104</r_object_id> ← Object Type Resource
    <object_name>acme01</object_name> ← Single Property
    <r_object_type>dm_cabinet</r_object_type>
    <title>Super User Cabinet</title>
    <subject />
    <i_vstamp>0</i_vstamp>
    <i_ancestor_id>
      <item>0c0004d280000104</item>
    </i_ancestor_id>
    <is_private>false</is_private>
    <r_folder_path> ← Repeating Property
      <item>/acme01</item>
    </r_folder_path>
  </properties>
  <links> ← Links
    <link rel="self" href="http://core-rs-demo.lss.emc.com:8080/dctm-rest/repositories/acme01/cabinets/0c0004d280000104" />
    <link rel="edit" href="http://core-rs-demo.lss.emc.com:8080/dctm-rest/repositories/acme01/cabinets/0c0004d280000104" />
  </links>
</cabinet> ← root element as the resource category

```

Not all resources have the same structure of the representation. For example, the *Repository* resource representation does not have the *type* or *properties* keys.

2.8.4 Multi-part request representation

Some operations may use multi-part Requests. This section provides examples of multi-part Requests in JSON and XML.

➡ Example 2-1: JSON representation in multi-part Request

```
POST http://localhost/rest-api-web/repositories/myrepo/
folders/0c00000c80000105/documents
Content-Type: multipart/form-data; boundary=314159265358979

--314159265358979
Content-Disposition: form-data; name=metadata
Content-Type: application/vnd.emc.documentum+json

{"properties": {"object_name": "rest-api-test"}}
--314159265358979
Content-Disposition: form-data; name=binary1
Content-Type: text/plain

This is primary content
--314159265358979--
```



➡ Example 2-2: XML representation in multi-part Request

```
POST http://localhost/rest-api-web/repositories/myrepo/
objects/0c00000c80000105/objects
Content-Type: multipart/form-data; boundary=314159265358979

--314159265358979
Content-Disposition: form-data; name=metadata
Content-Type: application/vnd.emc.documentum+xml

<?xml version="1.0" encoding="UTF-8"?>
<dm:object xmlns:dm="http://identifiers.emc.com/documentum">
<properties>
<dm:object_name>hello</dm:object_name>
<properties>
</dm:object>
--314159265358979
Content-Disposition: form-data; name=binary
Content-Type: text/plain

This is a sample
--314159265358979--
```



2.8.5 Error representation

An error representation contains the following items:

- HTTP Status code (mandatory) - identical to the status code in the Header.
- REST Error Code (mandatory) - REST application-specific code.
- REST Error Message (mandatory) - descriptive message for the error code.
- Root Causes (optional) - a set of error code/message mappings of the under layer.
- REST Error Id (mandatory) - a unique identifier of the error in the log.

Foundation REST API supports the JSON and XML formats in error responses.

➡ Example 2-3: XML Error Representation

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<error>
    <status>xxx</status>
    <code>xxx</code>
    <message>xxx</message>
    <details>xxx</details>
    <id>xxx</id>
</error>
```



➡ Example 2-4: Sample 1. Get an object with an invalid ID

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<error>
    <status>404</status>
    <code>E_RESOURCE_NOT_FOUND</code>
    <message>The resource is not found.</message>
    <details>(DM_API_E_EXIST) Document/object specified by 0b00000c80000dda
does not exist;
Cannot fetch a sysobject - Invalid object ID : 0b00000c80000dda;</details>
    <id>2275d26b-9761-43c9-9ca0-cd3a006d6c41</id>
</error>
```



➡ Example 2-5: Sample 2. Create a folder with an empty input message

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<error>
    <status>400</status>
    <code>E_VALIDATION_ATTR_MISSING</code>
    <message>Input message contains invalid items.</message>
    <details>Properties 'r_object_type' should not be null or empty;
Properties 'object_name' should not be null or empty;</details>
    <id>9dd9d03c-78fb-4c86-881c-d06cf2b5884b</id>
</error>
```



► **Example 2-6: JSON Error Representation**

```
{  
    "status":xxx,  
    "code":"xxx",  
    "message":"xxx",  
    "details":"xxx",  
    "id":xxx  
}
```



► **Example 2-7: Sample 1. Get an object with an invalid ID**

```
{  
    "status":404,  
    "code":"E_RESOURCE_NOT_FOUND",  
    "message":"The resource is not found.",  
    "details":"(DM_API_E_EXIST) Document/object specified by 0b00000c80000dda  
does not exist;  
Cannot fetch a sysobject - Invalid object ID : 0b00000c80000dda; ",  
    "id":"2275d26b-9761-43c9-9ca0-cd3a006c4541"  
}
```



► **Example 2-8: Sample 2. Create folder with empty input message**

```
{  
    "status":400,  
    "code":"E_VALIDATION_ATTR_MISSING",  
    "message":"Input message contains invalid items.",  
    "details":"Properties 'r_object_type' should not be null or empty;  
Properties 'object_name' should not be null or empty;",  
    "id":"2275d26b-9761-43c9-9ca0-cd3a006d6e64"  
}
```



2.8.6 Transaction support

Foundation REST API provides basic transaction support for copy and delete operations. The transaction begins at the start of an operation and is committed at the end of the operation, when it completes successfully. When any part of the operation fails, the entire operation is rolled back.

In a batch request, you can control the transaction behavior of the batch by using the transactional property.

2.9 The PUT and POST operations

When you perform a PUT operation or POST operation to update a resource, changes to the root element and links are ignored. When you try to update the namespace or read-only properties, an error is returned.

When performing a POST operation to create a resource, you can use the type property in JSON or the xsi:type property in XML to specify the object type of the resource. For Sysobject and its sub types, you can also use the r_object_type property to specify the object type. When the value of type (or xsi:type) and that of r_object_type are not consistent, the value of type (or xsi:type) takes precedence.

2.10 Runtime property configuration

Runtime property configuration files enable you to set preferences for how Foundation REST API handles certain choices in the course of its execution. For example, setting authentication schemes and specifying the default page size. Foundation REST API leverages two configuration files for runtime property setting. Both files are located in `dctm-rest.war\WEB-INF\classes`.

`rest-api-runtime.properties.template`

This file holds the default settings for all runtime properties. Do not modify the content in this file.

`rest-api-runtime.properties`

Out-of-the-box, this file contains no settings, indicating that all properties use their default values. When you need to modify the value of a certain runtime property, add an entry in this file, specifying the name and value of the property. Settings in this file override the settings in `rest-api-runtime.properties.template`.

2.10.1 Runtime profile

You can customize the runtime profile by using the `<rest.runtime.profile=>` runtime property.

There are two runtime profiles that control how error messages are shown. The two runtime profiles are *development* and *production*. This property specifies which runtime profile is used for error messages.

The default runtime profile is *production*, which returns a wrapped exception message so that clients do not see server implementation specific information, which could be a security issue making the Foundation REST API server vulnerable to attack.

The other runtime profile is *development*, and it returns implementation specific messages (the original error messages) including all server information. This runtime profile is meant to be used for development purposes only.

2.11 Batch operations

Starting from release 7.2, you can leverage the newly added Batches collection resource to execute a series of RESTful Web service operations in one Request. This resource also provides a number of batch options such as transactional, sequential, and so on. Furthermore, Foundation REST API provides the Batch Capabilities resource for you to check the list of resources that are batchable at runtime.

For more information, see *OpenText Documentum Content Management - Foundation REST API Reference Guide (EDCPKRST250400-PRE)*.

Chapter 3

Deploy Foundation REST API

For information about deploying Foundation REST API, see *OpenText Documentum Content Management - Server and Server Extensions Installation Guide (EDCSY250400-IGD)*.

3.1 General deployment configuration

On all application servers, the minimal and mandatory configuration is to use the `<dctm-rest.war>\WEB-INF\classes\dfc.properties` file to set your connection broker information.

For information on customizing the Foundation REST API server, such as how to change an authentication scheme, see the Foundation REST API runtime property file, `rest-api-runtime.properties`, which is located at `<dctm-rest.war>\WEB-INF\classes\rest-api-runtime.properties`.

To see all available configuration parameters, see the template file, `rest-api-runtime.properties.template`, which is located at `<dctm-rest.war>\WEB-INF\classes\rest-api-runtime.properties.template`.

For capturing logs and changing the logging level, use the properties file, `log4j.properties`, which is located at `<dctm-rest.war>\WEB-INF\classes\log4j.properties`.

Chapter 4

Resource specific features

4.1 Create an object with an attached Aspect and Lifecycle

When you import or create an object or document in the *Folder Child* series of resources, or create a cabinet in the *Cabinets* collection resource, the *Aspects* for the object, document, folder, or cabinet can be attached and the *Aspect* values can be set in the new object, document, folder, or cabinet. In addition to this, the *Lifecycle* can also be attached to the new object, document, folder, or cabinet.

Several resources have been improved to allow you to create or import an object and attach the *Aspects* and *Lifecycle* to that object. These include the following:

- *Folder Child Objects* collection resource
- *Folder Child Documents* collection resource
- *Folder Child Folders* collection resource
- *Cabinets* collection resource
- *Object Lightweight Objects* resource

4.1.1 Request

Request media types

- application/vnd.emc.documentum+xml
- application/vnd.emc.documentum+json

4.1.1.1 Using a combined approach

The following code demonstrates how an object can be created using a combined representation of each step in the larger process, which includes:

- The creation step (where the object, document, folder, or cabinet is created).
- Attaching *Aspects* and setting their values.
- Attaching a *Lifecycle* to the newly created object, document, folder, or cabinet as the case may be.

Example 4-1: Request body in XML

```
<?xml version="1.0" encoding="UTF-8"?>
<object>
  <properties>
```

```
<object_name>test ooo</object_name>
<r_object_type>dm_document</r_object_type>
<rest_aspect.rest_str>new created aspect string</rest_aspect.rest_str>
<rest_aspect.rest_r_int>
    <item>123456</item>
</rest_aspect.rest_r_int>
</properties>
<object-lifecycle>
    <lifecycle-id>4600000580004d14</lifecycle-id>
    <current-state>draft</current-state>
    <aliaset-name>doc-as</aliaset-name>
</object-lifecycle>
<object-aspects>
    <aspect>rest_aspect</aspect>
</object-aspects>
</object>
```



Example 4-2: Request body in JSON

```
{
    "properties" : {
        "object_name" : "test object",
        "r_object_type" : "dm_document",
        "rest_aspect.rest_str" : "new created aspect string",
        "rest_aspect.rest_r_int" : [
            123456
        ]
    },
    "object-lifecycle" : {
        "lifecycle-id" : "4600000580004d14",
        "current-state" : "draft",
        "aliaset-name" : "doc-as"
    },
    "object-aspects" : {
        "aspects":[
            "rest_aspect"
        ]
    }
}
```



4.1.2 Transactions and batches



Note: A transaction is a programming construct that internally performs a sequence of work steps and returns one result; either success or failure. A transaction is treated as a single unit of work for the purposes of satisfying a single Request and to ensure database integrity. When all of the internal work steps within a given transaction succeed, then the transaction succeeds. If any one of the internal work steps within the Transaction fail, then the whole Transaction fails.

When a value for the *object-lifecycle* parameter is provided, if the operation fails and the server is unable to roll back its changes, or drop the newly created object, then the server logs the error. When this happens, it also means that even though the operation has failed, the object was still created and now exists within the system.

At this point, it is strongly recommended that you do one of the following:

- Manually attach the *<Lifecycle>* to the new object by using the *object-lifecycle* resource.
- Manually drop the newly created object from the system.



Caution

It is extremely important that you complete one of the two preceding procedures. Otherwise, you will leave behind an orphaned object in your system.

4.1.2.1 Using a batch

When running the operation as part of a batch, if the operation has a value for the *object-lifecycle* parameter defined, then the batch cannot be put into a Transaction. The operation can only be run as part of a batch when a value for the *object-lifecycle* parameter is not defined.

4.2 Filter expression

A filter expression, which is the value of the `filter` URI parameter, enables you to filter entries in a collection of results according to the specified criterion. For instance, this can be used on the Cabinets Resource, Folder Child Documents Resource and so on. For example, GET requests to this URL returns all cabinets owned by user `<dmadmin>: /repositories/REPO/cabinets?inline=true&filter=owner_name='dmadmin'`.

This section describes the building blocks of a filter expression, including:

- “[Literals](#)” on page 38, which describes the literal formats in filter expressions.
- “[Filter expression functions](#)” on page 41, which describes the functions in filter expressions.
- “[Logical operators](#)” on page 45, which describes the logical operators supported by filter expressions.
- “[Comparison operators](#)” on page 45, which describes the comparison operators supported by filter expressions.



Notes

- All operator names and function names are case sensitive.
- The attributes that can be used in a filter expression are determined by the implementation of the individual collection resources. For example, on the Cabinets Resource, all `<dm_cabinet>` type attributes can be used in a filter expression. On the Folder Child Objects Resource, all `<dm_sysobject>` attributes can be used.

When you want to reduce collection items to make them a sub type of the collection, you can use the `<filter>` and `<object-type>` query parameters together. In this case, the attributes used in the filter expression can come

from the type specified by `<object-type>` parameter. For example, GET requests to this URL returns all cabinets of sub type `<custom_cabinet>` with a `<custom_region>` sub type attribute that equals `<apj>: /repositories/REPO/cabinets?inline=true&object-type&filter=custom_region='apj'`

4.2.1 Literals

Literals are values that are interpreted by the server exactly as they are entered. Filter expressions introduce the following types of literals:

- “Numeric literal” on page 38
- “String literal” on page 39
- “boolean literal” on page 39
- “Datetime literal” on page 40

4.2.1.1 Numeric literal

Filter expressions support integer literals and floating point literals.

4.2.1.1.1 Integer literal

An Integer literal specifies any whole number and is expressed in the following format:

[+ | -] n

Where n is any number between 0 and 2,147,483,647.

4.2.1.1.2 Floating point literal

- 5.347 (regular floating point literal)
- -4.12 (negative floating point literal)
- 21. (floating point literal with a blank fractional part)
- .66 (floating point literal with a blank integer part)

A floating point literal specifies any number that contains a decimal point and is expressed in the following format:

4.2.1.2 String literal

String literals are strings of printable characters and are enclosed in a pair of single quotes or a pair of double quotes.

If a string literal is enclosed in a pair of single quotes, the double quote character ("") can be included as a part of the literal without any change. For example:

```
'foo"bar'
```

Similarly, if a string literal is enclosed in a pair of double quotes, the single quote character ('') can be included as a part of the literal without any change. For example:

```
"The company's third quarter results were very good."
```

However, to include a single quote character ('') as a part of a literal that is enclosed in a pair of single quotes, you must include the single quote character twice. For example:

```
'The company''s third quarter results were very good.'
```

Similarly, to include a double quote character ("") as a part of a literal that is enclosed in a pair of double quotes, you must include double quote character twice. For example:

```
"foo""bar"
```

The maximum length of a string literal is determined by the maximum allowed by the underlying RDBMS, but in no case will the maximum length exceed 1,999 bytes. If a property is defined as a string data type, the maximum length of the string literal you can place in the property is defined by the property's defined length. If you attempt to place a longer value in the property, OpenText Documentum Content Management (CM) Foundation Java API throws an exception. You can change the behavior and allow Foundation Java API to truncate the character string value to fit the property by setting the Foundation Java API preference called `dfc.compatibility.truncate_long_values` in `dfc.properties`.

4.2.1.3 boolean literal

boolean literals specify constant values for the true and false values used in filter expressions. There are two boolean literal values: `true` and `false`.

4.2.1.4 Datetime literal

A DateTime literal represents a date or a combined date and time representation, which is enclosed in a pair of single quotes or a pair of double quotes, using one of the following syntaxes:

- `date ("YYYY-MM-DD")`
- `date ("YYYY-MM-DDThh:mm:ss.[sss][TZD]")`

Where:

- YYYY = four-digit year
- MM= two-digit month (01=January, and so on.)
- DD= two-digit day of month (01 through 31)
- hh = two digits of hour (00 through 23) (am/pm NOT allowed)
- mm= two-digit of minute (00 through 59)
- ss = two-digit of second (00 through 59)
- sss = three-digit millisecond. (000 through 999, an optional field which is ignored when being processed. This is because Documentum CM Server does not store the millisecond field in a DateTime property.)
- The optional field TZD represents the time zone designator:
 - If this field is set to the special UTC designator ("Z") or unspecified, the time is expressed in UTC (Coordinated Universal Time).
 - The time is expressed in local time, together with a time zone offset in hours and minutes, which represents the difference between the local time and UTC.



Notes

- If a date without time is entered, the time 00:00:00 is assumed.
- In SQL Server, both `date ("1970-01-01T00:00:00.000+0000")` and `date ("1753-01-01T00:00:00.000+0000")` are taken as a null DateTime.
- In Oracle, `date ("0001-01-01T00:00:00.000+0000")` is taken as a null DateTime.



Example 4-3: Examples for DateTime literals

- `date ("2013-01-01")`

Because the time field is not specified. The time 00:00:00 is assumed. This literal equals to `date ("2013-01-01T00:00:00Z")`.

- `date ('2007-07-16T19:20:30.45')`

Because the TZD field is not specified, this DateTime literal is expressed in UTC. This literal equals to `date ('2007-07-16T19:20:30.45Z')`.

- `date ("2007-07-16T19:20:30.45Z")`
- `date ("2007-07-16T19:20:30.45+08:00")`
- `date ('2007-07-16T19:20:30.45-03:00')`



4.2.2 Filter expression functions

Filter expression functions are operations on values. Filter expressions support the following functions:

- “The starts-with function” on page 41
- “The contains function” on page 42
- “The between function” on page 42
- “The type function” on page 43
- “The nilled function” on page 44
- “Scalar functions” on page 47

4.2.2.1 The starts-with function

The `starts-with` function checks the starting string of a property.

This function is a boolean term that returns `True` if the specified property starts with a specified literal string, and `False` otherwise.

Syntax:

```
starts-with(<property-name>, "<string-literal>")
```

Arguments:

- `property-name`: Specifies the property whose starting string this function tests.
- `string-literal`: String literal with which the starting literal of the specified property is compared.



Example 4-4: Example for starts-with

```
starts-with(object_name, "foo")
```

By using this filter expression, the Request only returns objects whose `object_name` starts with `foo`.



4.2.2.2 The contains function

The `contains` function checks whether or not a property contains a specified literal string.

This function is a boolean term that returns `True` if the property contains the specified string, and `False` otherwise.

Syntax:

```
contains(<property-name>, "<string-literal>")
```

Arguments:

- `property-name`: Specifies the property that this function tests.
- `string-literal`: String literal with which the specified property is compared.

Example 4-5: Example for contains

```
contains(object_name, 'hello')
```

By using this filter expression, the Request only returns objects whose `object_name` contains hello.



4.2.2.3 The between function

The `between` function checks whether or not the value of a property lies between a specified range.

This function is a boolean term that returns `True` if the property lies between the specified range, and `False` otherwise.

Syntax:

```
between(<property-name>, from, to)
```

Arguments:

- `property-name`: Specifies the property that this function tests.
- `from`: The lowest value in the range that this function evaluates. This argument can be one of the following types:
 - Datetime
 - Numeric
- `to`: The highest value in the range that this function evaluates. This argument can be one of the following types:
 - Datetime

- Numeric



Note: All arguments in this function must be of the same type.

➡ **Example 4-6: Examples for between**

```
between(r_link_cnt, 1, 5)
```

By using this filter expression, the Request returns objects whose r_link_cnt is no less than 1 and no greater than 5.

```
between(r_modify_date, date("2013-01-16"), date("2013-01-18"))
```

By using this filter expression, the Request only returns objects whose r_modify_date is no earlier than 2013-01-16 and no later than 2013-01-18.



4.2.2.4 The type function

The type function checks whether or not an object is an instance of a specified type or its subtype.

This function is a boolean term that returns True if the object is an instance of the specified type or its subtype, and False otherwise.

Syntax:

```
type(<type-name>)
```

Arguments:

- type-name: Specifies the type that this function uses as a filter.

➡ **Example 4-7: Example for type**

```
type(dm_folder)
```

By using this filter expression, the Request only returns objects of the dm_folder type and objects of any subtype of dm_folder.



4.2.2.5 The nilled function

The `nilled` function checks the nullity of a property.

This function is a boolean term that returns `True` if the value of the specified property is null, and `False` otherwise.

Syntax:

```
nilled(<property-name>)
```

Arguments:

- `property-name`: Specifies the property that this function tests.

➡ **Example 4-8: type**

```
nilled(object_name)
```

By using this filter expression, the Request only returns objects whose `object_name` is null.



4.2.2.6 Using scalar functions with built-in functions

Scalar functions can be added to attribute names and used with the built-in functions to perform operations.

Syntax:

```
starts-with(scalar-function(<attribute-name>), <string-literal>)
...
contains(scalar-function(<attribute-name>), <string-literal>)
...
between(scalar-function(<attribute-name>), from, to)
...
```

Arguments:

- `attribute-name`: Specifies the name of an attribute that this function parses.
- `string-literal`: Specifies the string that the function uses in performing its operation.
- `from, to`: Specifies the ASCII character numbers, as a range of characters, to use to find objects whose `object_name` starts with any one of the characters in the range specified.

➡ **Example 4-9: With the starts-with built-in function**

With this filter expression, the Request only returns objects whose `object_name` starts with the string `hello`, case insensitive.

```
starts-with(upper(object_name), 'HELLO')
```



➡ Example 4-10: With the contains built-in function

With this filter expression, the Request only returns objects whose `object_name` contains the string `hello`, case insensitive.

```
contains(lower(object_name), 'hello')
```



➡ Example 4-11: With the between built-in function

With this filter expression, the Request only returns the objects whose `object_name` begins with the characters '`c`', '`d`', or '`e`'. These are the ASCII character numbers, as a range, that the function uses to look for objects.

```
between(ascii(object_name), 99, 101)
```



4.2.3 Logical operators

Logical operators apply to boolean terms and return boolean values. The following logical operators are supported in filter expressions.

- `not`
- `and`
- `or`

These operators follow the standard logical semantics. They are listed in order of precedence, with the highest-precedence one at the top.

4.2.4 Comparison operators

Comparison operators compare one expression to another. Filter expressions support two sets of comparison operators: value comparison operators and general comparison operators. These two sets of operators function the same except for note [1] in the following table:

Value comparison operator	General comparison operator	Description	Example
<code>eq [1]</code>	<code>=</code>	Equal	<code>object_name = "REST"</code> <code>object_name eq REST"</code>

Value comparison operator	General comparison operator	Description	Example
ne	!=	Not equal	object_name != "REST" object_name ne "REST"
lt	<	Less than	r_modify_date < "2013-01-16" r_modify_date lt "2013-01-16"
le	<=	Less than or equal to	r_full_content_size <= 2000 r_full_content_size le 2000
gt	>	Greater than	r_full_content_size > 2000 r_full_content_size gt 2000
ge	>=	Greater than or equal to	r_modify_date >= "2013-01-16" r_modify_date ge "2013-01-16"
[1] This operator cannot be used to compare a property with a list of values. For more information about how to compare the value of a property with a list of values, see " Comparison with multiple values " on page 46.			

4.2.4.1 Comparison with multiple values

The = comparison operator allows you to compare the value of a property with a list of values. When a filter expression compares the value of a property with multiple literals by using the = comparison operator, the expression returns True if the value equals to any of the literals, and False otherwise.

Syntax:

```
<property-name> = ("literal1", " literal2", "literal3" ... )
```

 **Example 4-12: Example for comparison with multiple values**

```
object_name = ("foo", "bar")
```



Notes

- The list of literals must be enclosed in a pair of brackets.
- The eq comparison operator does not support this function.

4.2.4.2 Comparison with repeating properties

The following syntax enables you to check whether or not any item in a repeating property matches the comparison.

Syntax:

```
<repeating-property-name> /item <comparison-operator><literal-or-value-list>
```

The expression returns `True` if any value of the repeating property matches the comparison, and `False` otherwise.

► **Example 4-13: Example for comparison with repeating properties**

- `keywords/item = "hey"`
- `keywords/item = ("hello", "world")`



4.2.4.3 Scalar functions

The following scalar functions are supported:

- ASCII
- BITAND
- BITCLR
- BITSET
- UPPER
- LOWER
- SUBSTR

4.2.4.3.1 Comparison with scalar functions

All of the preceding scalar function can be used on attribute-names to perform comparisons.

Syntax:

```
scalar-function(<attribute-name>) <comparison-operator> value
```

► **Example 4-14: Return a specific object name**

This code sample shows you how to use a scalar function (in this case the `upper(string)` function) to return only those objects whose object name is `<document>`, case insensitive.

The `upper` function used in this sample takes an `object_name` as its only parameter, converts it all to upper case characters, and returns it. This returned

value is then compared to the value *DOCUMENT* and if it matches, then a boolean value of *true* is returned.

```
upper(object_name) = 'DOCUMENT'
```



➤ Example 4-15: Return specific characters by placement

This code sample shows you how to return only those characters that are in the 3rd, 4th, 5th, and 6th position, within the object name, which when concatenated together, form the word test. We use the `substr(<attribute-name>, starting pos, num of chars)` function starting at the third character position (from a zero index starting point) and return 4 characters to form the string test.

```
substr(object_name,3,4) = 'test'
```



Caution

Embedding a scalar function within another scalar function is not supported.

4.2.5 Examples of the filter expression

This section provides examples that demonstrate how to use filter expressions.

➤ Example 4-16: Return objects modified in a specified time period

```
type(dm_document) and between(r_modify_date, date("2013-01-16"), date("2013-01-18"))
```

With this expression, the Request returns only the instances of `dm_document` that were modified between 2013-01-16 and 2013-01-18.



➤ Example 4-17: Filtering out objects

With this expression, the Request filters out those resources whose `object_name` starts with `foo`.

```
not(starts-with(object_name, "foo"))
```



➤ Example 4-18: Return specific objects according to criteria

With this expression, the Request returns only those resources whose `object_name` contains `hello` or `hey`. Additionally, the resources that were modified later than 2013-01-01 are filtered out and therefore excluded from the returned values.

```
(contains(object_name, 'hello') or contains(object_name, 'hey')) and r_modify_date le date("2013-01-01")
```



➡ Example 4-19: Return objects containing specified keywords

With this expression, the Request returns only those resources that have the keyword hello or world. Additionally, resources whose author is Jack are filtered out and therefore excluded from the returned values.

```
author != "Jack" and keywords/item = ("hello", "world")
```



4.3 Property view

You can use the property view to specify the object properties to retrieve in an operation. This can be done by creating a view expression in the `view` query parameter. A view expression can either be a predefined view expression or a custom view expression that contains a list of property names.

4.3.1 Predefined view expression

Foundation REST API supports the following predefined view expressions:

View expression	Description
all	Returns all properties of the Requested object to the client. If you use this view expression, the performance may degrade when a request tries to retrieve a collection resource.
default	Returns the default set of properties of the Requested object to the client. This is the default value if there is no value specified for the <code>view</code> query parameter.

4.3.2 Custom view expression

A custom view expression contains a list of property names of the properties, within parentheses, that you want to retrieve, separated by commas. For example:
`(<property1>, <property2>, ...)`.

A property name must conform to the following rules:

- The maximum allowed length of the name is 27 characters.
- The first character of the name must be a letter. The remaining characters can be letters, digits, or underscores.
- The name cannot contain spaces or punctuation.
- The name cannot be any of the words reserved by the underlying RDBMS, such as SQL keywords.



Note: Property names are not case sensitive.

When a property name, that is specified in the expression, violates any of the preceding naming rules, the view expression is considered to be invalid and is therefore rejected. In this case, the server returns an HTTP 400 Bad Request error code status.

4.3.2.1 View expression on collections

Except for the preceding rules, in versions prior to 7.3, you cannot specify a custom property in a custom view expression when trying to get a collection resource. Otherwise, an HTTP 400 Bad Request error code status is returned.

Example 4-20: URLs Regarded as Bad Requests

For example, GET requests to a URL that resembles the following examples are regarded as bad requests:

```
/repositories/<repository>/folders/<folderID>/documents?inline=true  
&view=r_object_id,<custom_property_name>
```



To retrieve custom properties in a collection resource in versions before 7.3, you can set the *view* parameter to :all. Foundation REST API release 7.3 and later support tolerant view expressions on most collection resources. The *view* parameter's usage on singular resources remains the same.



Note: When the *view* parameter is used on a single object resource that supports this parameter (such as the Cabinet resource, Document resource, and so on). The *view* attribute names must match the definition of the object data type. A Bad request (400) error is returned when any unknown attributes for the object data type are specified in the *view* parameter.

When the *view* parameter is used on a collection based resource that supports this parameter (such as the Cabinets resource, Folder Child Documents resource, and so on), a comma-separated list of *view* attributes can contain both base type attribute names and arbitrary extended attribute names.

For example, let's assume that a folder X has a number of documents in it with the object type *<dm_document>* and document sub types *<doc_ext1>*, *<doc_ext2>*, and so on. When a Foundation REST API client retrieves the folder children, it can specify the *view* attributes from both *<dm_document>* and its sub types.

Here is a code sample that shows you the Request:

```
// Get a folder child documents feed.  
// In this sample, attributes 'r_object_id' and 'object_name' come from dm_document.  
// 'doc_ext1_attr' comes from doc_ext1, and 'doc_ext2_attr' comes from dmc_ext2.  
GET /documents?inline=true&  
view=r_object_id,object_name,doc_ext1_attr,doc_ext2_attr HTTP/1.1
```

Note the following *view* usage on a collection based resource:

- Aspect type attributes can also be put into the *view* when the aspect type is attachable to the collection base type, such as <*dm_document*> as shown in the preceding sample.
- Collection items return the specified attributes only when the attributes exist in those objects.
- Unknown attributes for collection items are ignored.

4.3.3 View expression syntax

The syntax of a view expression is defined with the following EBNF:

```

view-expression = predefined-view | properties-list ;
predefined-view = leading-separator, predefined-view-name ;
leading-separator = ":" ;
predefined-view-name = "default" | "all" | "none" ;
properties-list = property-name, { "," property-name } ;
property-name = character, 26 * [ character ] ;
character = letter | digit | underscore ;
letter = "a" .. "z" | "A" .. "Z" ;
digit = "0" .. "9" ;
underscore = "_" ;

```

 **Example 4-21: Predefined view example**

```
view=:all
```

This example sets the *view* parameter to the predefined view expression:*:all*, meaning that all properties of the Requested object are returned.



 **Example 4-22: Custom view example**

```
view=object_name,r_object_type
```

This example sets the *view* parameter to the custom view expression *object_name,r_object_type*, meaning that only the *object_name* and *r_object_type* properties of the Requested object are returned.



4.4 NULL in REST

Foundation REST API sets the value of a single property to the corresponding Foundation Java API default value when the property is set to NULL. For details, see the following table.

OpenText Documentum CM data type	JSON/XML data type	Foundation Java API default value
DM_BOOLEAN	boolean	false
DM_INTEGER	Number	0

OpenText Documentum CM data type	JSON/XML data type	Foundation Java API default value
DM_DOUBLE	Number	0.0
DM_STRING	String	""
DM_ID	String	"0,000,000,000,000"
DM_TIME	String	"nulldate"

For repeating properties, setting NULL for a property or leaving a property empty removes all values from the repeating property.

4.4.1 NULL value representation

In a response that is returned from the Foundation REST API server, NULL or empty values are represented as follows:

► **Example 4-23: NULL value for Datetime in XML**

```
<dm:property_name xsi:nil="true"/>
```



► **Example 4-24: NULL value for Datetime in JSON**

```
"property_name":null
```



► **Example 4-25: NULL value for string in XML**

```
<dm:property_name/>
```



► **Example 4-26: NULL value for string in JSON**

```
"property_name": ""
```



► **Example 4-27: NULL value for repeating properties in XML**

```
<dm:property_name xsi:nil="true"/>
```



► **Example 4-28: NULL value for repeating properties in JSON**

```
"property_name":null
```





Note: The preceding examples show how the Foundation REST API server handles NULL values in a response. Foundation REST API clients are free to use any valid format to represent NULL values in a request. For example, you can use this pattern to represent a NULL value in XML:

```
<dm:property_name><dm:property_name/>
```

For atom feeds, if a property is set to NULL or empty, the property is not displayed in the Response.

4.5 Whitespace in XML

Foundation REST API preserve XML space for both input and output XML messages, so the client must normalize the Request before sending. For details, see the following examples:

➡ Example 4-29: Preserve Whitespace Within the Request

```
<document>
    <properties>
        <object_name>    my    document    </object_name>
    </properties>
</document>
```



➡ Example 4-30: Preserve Whitespace in the Response

```
<document xmlns="..." xsi:type="dm_document" definition="...">
    <properties xsi:type="dm_document-properties">
        <r_object_id>0900000180010d8e</r_object_id>
        <object_name>    my    document    </object_name>
    ...

```



Note: Even if all whitespace characters are processed by Foundation REST API correctly, some of them can be ignored. This is a limitation in Documentum CM Server.

4.6 Thumbnail link

Thumbnails on atom feed entries help mobile clients preview the documents within a collection resource. Thumbnail links are available to the following collection resources:

- Cabinets
- Folder Child Documents
- Folder Child Objects
- Folder Child Folders
- Checked Out Objects

For more information about these resources, see *OpenText Documentum Content Management - Foundation REST API Reference Guide (EDCPKRST250400-PRE)*.

To enable thumbnail links, the following conditions must be met:

- The OpenText™ Documentum™ Content Management Thumbnail Server must be installed.
 - The thumbnail query parameter is set to *true*.

When retrieving these collection resources, you can use the `thumbnail` query parameter to determine whether or not to return the thumbnail link for each entry.

Variable	Description	Data type	Default value
thumbnail	<p>Specifies whether or not to return the thumbnail link for each entry.</p> <ul style="list-style-type: none">• <i>true</i> - Return the thumbnail link for each entry in the collection resource.• <i>false</i> - Do not return the thumbnail link for each entry in the collection resource.	boolean	Value of the rest.entry.thumbnail.default parameter in the rest-api-runtime.properties file, which defaults to <i>false</i> .

Example 4-31: Thumbnail link in XML

```

</entry>      5f.jpg&store=thumbnail_store_01"/>
<entry>
...
</entry>
</feed>

```



Example 4-32: Thumbnail link in JSON

```

{
  "id": "/repositories/acme01/folders/0c0004d280000d1f/documents",
  "title": "Folder child objects",
  "updated": "2013-05-08T23:54:25.165-0700",
  "author": [
    {
      "name": "OpenText Documentum"
    }
  ],
  "links": [
    {
      "rel": "self",
      "href": "/repositories/acme01/folders/0c0004d280000d1f/documents"
    }
  ],
  "entries": [
    {
      "id": "/repositories/acme01/objects/090004d280005123",
      "title": "testOverview.ppt",
      "updated": "2013-05-07T01:16:36.000-0700",
      "author": [
        {
          "name": "dmadmin",
          "uri": "/repositories/acme01/users/646d61646d696e"
        }
      ],
      "content": {
        "content-type": "<literal>application/json</literal>",
        "src": "/repositories/acme01/documents/090004d280005123"
      },
      "links": [
        {
          "rel": "edit",
          "href": "/repositories/acme01/documents/090004d280005123"
        },
        {
          "rel": "icon",
          "href": "/thumbsrv/getThumbnail?path=000004d2\80\00\00\
5f.jpg&store=thumbnail_store_01"
        }
      ]
    }
  ]
}

```



4.7 Support for relative URIs

URIs come in different formats, including:

- *Absolute URI*: An absolute URI is a URI that contains both the scheme (i.e. http, https) and the authority (i.e. host, port).
- *Base URI*: A base URI is defined by the context of the application. It is usually the URI with the root path to the application. This type of URI must follow the same syntax as an absolute URI.
- *Relative URI*: A relative URI is a sub component of the URI relative to the base URI of an absolute URI. This type of URI usually starts with a slash (/) or an empty character.

Foundation REST API produce absolute URIs in resource representations. Here is an example of an absolute URI that was created in the Cabinet resource.

Example 4-33: Cabinet resource – JSON – Absolute URIs

```
{
  "name": "cabinet",
  "type": "dm_cabinet",
  "definition": "http://localhost:8080/dctm-rest/repositories/REPO/types/
                dm_cabinet.json",
  "properties": {
    "object_name": "demo",
    "owner_name": "Administrator"
  },
  "links": [
    {
      "rel": "self",
      "href": "http://localhost:8080/dctm-rest/repositories/REPO/cabinets/
               0c000058000211b.json?view=object_name,owner_name"
    },
    ...
  ]
}
```

Using an absolute URI on the client side allows the HTTP client to use the href directly without the need for further URI resolution.

However, with the maturity of HTTP and REST development, most client libraries today support relative URIs as well. The benefits in using relative URIs, as compared to absolute URIs, are as follows:

- Removing the scheme and authority part from the href simplifies the Response body rewrite rules in proxy or balancer servers.
- Removing the scheme and authority part from the href separates deployment and resource identification.
- Removing the scheme and authority part from the href reduces the payload bytes in responses.

Foundation REST API 16.3 supports relative URIs in resource responses.

4.7.1 Using relative URIs in responses

To enable relative URIs in responses, the administrator needs to set the `rest.use.relative.url` property in `dctm-rest.war\WEB-INF\classes\rest-api-runtime.properties` to true. To remain backward compatible, the default deployment of REST continues to produce absolute URIs.

➤ Example 4-34: rest-api-runtime.properties

```
# To produce relative URIs in resource responses, set the below property value to true.
# When the property is set to true, URIs in both link relations and atom IDs will be in
# the format of a relative URI.
# The default value is false.
rest.use.relative.url = true
```



When relative URIs have been enabled, the resource Response is similar to the following samples:

➤ Example 4-35: Cabinet resource - JSON - relative URIs

```
{
  "name": "Cabinet",
  "type": "dm_cabinet",
  "definition": "/repositories/REPO/types/dm_cabinet.json",
  "properties": {
    "object_name": "demo",
    "owner_name": "Administrator"
  },
  "links": [
    {
      "rel": "self",
      "href": "/repositories/REPO/cabinets/0c0000058000211b.json?
view=object_name,owner_name"
    },
    ...
  ]
}
```



➤ Example 4-36: Cabinets feed resource - JSON - Relative URIs

```
{
  "id": "/repositories/REPO/cabinets",
  "title": "Cabinets",
  "author": [
    {
      "name": "OpenText Documentum"
    }
  ],
  "updated": "2017-08-18T02:52:30.556+00:00",
  "page": 1,
  "items-per-page": 2,
  "links": [
    {
      "rel": "self",
      "href": "/repositories/REPO/cabinets.json?items-per-page=2"
    },
    {
      "rel": "next",
      "href": "/repositories/REPO/cabinets.json?items-per-page=2&page=2"
    }
  ]
},
```

```

{
  "rel": "first",
  "href": "/repositories/REPO/cabinets.json?items-per-page=2&page=1"
}
],
"entries": [
  {
    "id": "/repositories/REPO/cabinets/0c00000580000105",
    "title": "Administrator",
    "author": [
      {
        "name": "REPO_ADMIN",
        "uri": "/repositories/REPO/users/REPO_ADMIN.json"
      }
    ],
    "summary": "dm_cabinet 0c00000580000105",
    "updated": "2015-03-03T10:15:33.000+00:00",
    "published": "2015-03-03T10:15:33.000+00:00",
    "links": [
      {
        "rel": "edit",
        "href": "/repositories/REPO/cabinets/0c00000580000105.json"
      }
    ],
    "content": {
      "type": "application/json",
      "src": "/repositories/REPO/cabinets/0c00000580000105.json"
    },
    ...
  ]
}

```



4.7.2 Using relative URIs in requests

Some Core resources accept resource URIs in their request bodies. For example:

- Make a batch request in the `Batches` resource.
- Make a copy request in the `Folder Child Objects` resource.

In these resources, both absolute URIs and relative URIs are considered as valid inputs to identify other resources. The usage of relative URIs in request bodies is not impacted by the `rest.use.relative.url.runtime` property setting.

4.7.3 Other notes

In the Content Metadata resource, the Response can produce OpenText™ Documentum™ Content Management Accelerated Content Services or OpenText™ Documentum™ Content Management Branch Office Caching Services server content URLs. These links remain as absolute URIs.

In REST extensibility, external links can be added to responses. These links also remain as absolute URIs.

4.8 Feed pagination

Paged feeds can be useful when the number of entries is very large or indeterminate. To save bandwidth, clients navigate through the feed and only access a subset of the feed's entries as necessary.

Foundation REST API utilizes the following query parameters for feed pagination (For detailed information, see “[Common HTTP headers](#)” on page 9):

- page
- items-per-page
- include-total

If all results can fit on one page, the Response is not paged. In this case, the Response does not contain the page and items-per-page fields even if they are specified in the Request. Also, the pagination link relations, such as first, last, previous and next, are not available.

For a paged feed, the Response must have at least one of the following link relations:

- first
- This link relation is always available.
- last
- This link relation is available when the size of the entire feed is known. This condition is met in the following scenarios:
 - The items-per-page query parameter specified is larger than the number of entries in the Response for current page.
 - The client sets the include-total query parameter to calculate the number of entries of the feed.
 - The current page is the last page of the feed.
- previous
- This link relation is available when the current page is not the first page.
- next
- This link relation is available when the current page is not the last page or when the size of the feed is unknown.

For detailed information about these link relations, see [Appendix A, Link relations](#) on page 389.

4.9 Full text query in collection resources

When you send a GET request to one of the following collection resources, you can use the full text query parameter q to filter entries in a result set according to specific criterion.

- Folder Child Documents
- Folder Child Objects
- Checked Out Objects
- All Versions

For more information about these resources, see *OpenText Documentum Content Management - Foundation REST API Reference Guide (EDCPKRST250400-PRE)*.

The value of the full text query parameter q is a string that follows the [simple search language](#) syntax.



Note: The parentheses which can be used in search, are not supported in the parameter q when being appended to the preceding collection resources.

4.10 Simple search language

The simple search language is used in the Search resource and the [full text query](#) parameter to specify search criterion.

Search criterion in the simple search language must follow this syntax:

```
searchExpression: orExpression
orExpression: andExpression (<OR> andExpression)*
andExpression: operandExpression (<AND> operandExpression)*
operandExpression: ( parenthesis | implicitExpression )
parenthesis : <PARENTHESIS_START> orExpression <PARENTHESIS_END>
implicitExpression: ( term )+
term: ( positiveTerm | <NOT> positiveTerm)
positiveTerm: ( <WORD> | <PHRASE> )
<DEFAULT> SKIP :
{
<SPACE: ( [ " ", "\t", "\n", "\r" ] )+ >
}
<DEFAULT> TOKEN :
{
<AND: "and" >
| <OR: "or" >
| <PARENTHESIS_START: "(" >
| <PARENTHESIS_END: ")" >
| <NOT: "not" >
| <WORD: ( ~["(", ")", "\'", "\\"", "\t", "\n", "\r"] )+ >
| <PHRASE: "\'" ( ~["\'"] )+ "\'" >
}
```

<term> can be a [word](#), or a [phrase](#). It can also be a list of words or phrases, or both, separated by spaces and quoted appropriately. When used in the Search resource, <term> can also be enclosed in parentheses to escape [boolean operators](#).

The full text simple language supports:

- Words
- Phrases
- Implicit AND
- boolean Operators
- Wildcards

4.10.1 Words

A word is a set of characters with the following exceptions:

- (
-)
- \"
- \t
- \n
- \r

Multiple words enclosed in double quotes are treated as a [phrase](#), such as "foo bar". When the words are not enclosed, [implicit and](#) is applied.

4.10.2 Phrases

A phrase, which contains more than one word separated by spaces, is enclosed in double quotes. For example:

"foo bar" returns objects that contain the phrase foo bar.

The single quote character (') can be included as a part of a phrase without any change. For example: "foo ' bar"

Note that \" is not supported even when it is enclosed in double quotes. For example, "foo \" bar" is invalid.

4.10.3 Implicit AND

A blank space separating two terms is interpreted as the and boolean operator. For example:

foo bar

This expression, which equals to foo and bar, returns objects that contain both foo and bar. To search for objects that contain either foo or bar, include or in the expression explicitly:

foo or bar

4.10.4 Boolean operators

The following boolean operators are supported in the simple search language.

- not
- and
- or

These operators follow the standard logical semantics. They are listed in order of precedence, with the highest- precedence one at the top.

In the Search resource, you can use both parentheses and double quotes to escape boolean operators. In the [full text query](#) parameter, you can only use double-quotes to escape boolean operators.

To include any of the boolean operators as a string in search criterion, enclose the operators in double quotes. And thus, the system treats them as phrases. For example: "and" or "or"

4.10.5 Wildcards

The following wildcard characters are supported in the simple search language.

Wildcard	Description	Example
*	Matches any number of characters. You can use the asterisk (*) anywhere in a character string.	wh* finds what, white, and why, but not awhile or watch.
?	Matches a single alphabet in a specific position.	b?ll finds ball, bell, and bill, but not buell.

4.11 Facet search

When xPlore is configured for the Documentum CM Server repositories, Foundation REST API supports full-text search by facets, which enables you to explore a collection of search results categorized into specific dimensions called facets. By default, the following properties of a SysObject are facet-enabled:

- r_modifier
- keywords
- r_modify_date
- r_full_content_size
- a_application_type
- r_object_type

- owner_name

To enable the use of facets on other properties, you must modify your xPlore configuration and re-index.

When you perform a facet search, a facet section appears in the Response feed of the search result at the same level as the entry element.

4.11.1 Facet search with URL parameters

More information on how to use facet requests with URL parameters can be found in the *OpenText Documentum Content Management - Foundation REST API Reference Guide (EDCPKRST250400-PRE)*. Here is a code sample that shows you facet results:

```
<feed>
  ...
  <entry>
    ...
    </entry>
    ...
    <entry>
      ...
      </entry>
      <dm:facet>
        <dm:facet-id>facet_r_modify_date</dm:facet-id>
        <dm:facet-label>Modify Date</dm:facet-label>
        <dm:facet-value>
          <dm:facet-value-id>LAST_YEAR</dm:facet-value-id>
          <dm:facet-value-count>23</dm:facet-value-count>
          <dm:facet-id>facet_r_modify_date</dm:facet-id>
          <dm:facet-value-constraint>
            2013-01-01T00:00:00.000\+0000/2014-01-01T00:00:00.000\+0000
          </dm:facet-value-constraint>
          <link rel="search"
            href="http://localhost:8080/dctm-rest/repositories/acme01/
              search?q=emc&facet=r_modify_date&
              facet-value-constraints=2013-01-01T00:00:00.000%5C%2B0000/
              2014-01-01T00:00:00.000%5C%2B0000" />
        </dm:facet-value>
        <dm:facet-value>
          <dm:facet-value-id>LAST_MONTH</dm:facet-value-id>
          <dm:facet-value-count>3</dm:facet-value-count>
          <dm:facet-id>facet_r_modify_date</dm:facet-id>
          <dm:facet-value-constraint>
            2014-04-01T00:00:00.000\+0000/2014-05-01T00:00:00.000\+0000
          </dm:facet-value-constraint>
          <link rel="search"
            href="http://localhost:8080/dctm-rest/repositories/acme01/
              search?q=emc&facet=r_modify_date&
              facet-value-constraints=2014-04-01T00:00:00.000%5C%2B0000/
              2014-05-01T00:00:00.000%5C%2B0000" />
        </dm:facet-value>
      </dm:facet>
    </entry>
  </feed>
```

The following table describes the properties in the facet section in detail:

Property	Description
facet-id	Indicates the property to be used as the facet. When being used in <code>facet-id</code> , a property is prefixed with <code>facet_</code> . In the example, the <code>r_modify_date</code> property is used as the facet.
facet-label	Label of the property used as the facet. Labels of properties are defined in Foundation Java API interfaces. For example, the label of <code>r_modify_date</code> is <code>Modify Date</code> .
facet-value	Each <code>facet-value</code> section contains data of the results belonging to one group. In the example, two <code>facet-value</code> sections appear, one for instances whose <code>r_modify_date</code> falls in last year's date range and the other for instances whose <code>r_modify_date</code> falls in last month's date range.
facet-value-id	Indicates the value of property that is used as the facet
facet-value-count	Indicates the number of results belonging to a certain group
facet-value-constraint	Indicates the property constraints expression.
search link	Points to corresponding results of a certain group. For example, you can navigate to instances whose <code>r_modify_date</code> falls in last year's date range by accessing this link: 1 <code>http://localhost:8080/dctm-rest/repositories/acme01/search?</code> 2 <code>q=emc&facet=r_modify_date&facet-value-</code> <code>constraints=2013-01-01T00:00:00.000%5C</code> <code>3 %2B0000/2014-01-01T00:00:00.000%5C%2B0000</code>

When a repeating property is used as the facet, entries in the result set may also contain a facet block that is comprised of multiple facet value groups, each of which represents a combination of repeating values (`<value_a> + <value_b>`). This enables you to navigate to the results whose repeating property contains both `<value_a>` and `<value_b>` by accessing the corresponding search link.



Note: The plus sign (+) is encoded as %2B in the XML representation.

Consider the following scenario:

- You use the repeating property keywords as the facet.
- The result set contains three objects:
 - `<Object1>`, which contains the keywords `foobar` and `V1`.
 - `<Object2>`, which contains the keywords `foobar` and `V2`.
 - `<Object3>`, which contains the keywords `foobar`, `V1`, and `V2`.

In this scenario, when you click the search link `http://host:port/dctm-rest/repositories/documentum1/search?q=HelloWorld&facet=keywords&facet-value-constraints= foobar` to navigate to the entries whose keywords contains `foobar`, all entries in the feed contain a facet block that has multiple facet value groups. Each of these groups represents a value combination (`foobar + x`). For example, when you

access the search link `http://host:port /dctm-rest/repositories/documentum1/search?q=HelloWorld&facet=keywords&facet-value-constraints=foobar%2BV1` in the first entry, `<Object1>` and `<Object3>` are returned because their keywords contain both `foobar` and `V1`. Similarly, the link `http://host:port /dctm-rest/repositories/documentum1/search?q=HelloWorld&facet=keywords&facet-value-constraints= V1%2BV2` returns two results, `<Object1>` and `<Object3>`.

In addition to the AND operator (`+`), you can also use the OR operator (`|`), which is encoded as `%7C` in URLs, in `facet-value-constraints`.



Note: These '`+`' and '`|`' operators have the same precedence in `facet-value-constraints`, and the expression is evaluated from right to left. For example, the expression `a+b|c+d` is treated as `a+(b|(c+d))`.

4.11.2 Facet search with AQL

Here is a code sample that shows the facet request that is used when you want to navigate into the facet groups:

Example 4-37: Navigate into a facet group

```
<search>
  <expression-set>
    <expressions>
      <property-list name="owner_name" operator="IN">
        <values>
          <value>dmadmin</value>
        </values>
      </property-list>
    </expressions>
  </expression-set>
  <facet-definitions>
    <facet-definition id="facet_type">
      <attributes>
        <attribute>r_object_type</attribute>
      </attributes>
    </facet-definition>
  </facet-definitions>
</search>
```

The following are the facet results. The elements in this set of facet results are similar to the results from the “[Facet search with URL parameters](#)” on page 63 section. However, the URL for the link relation search has a new element called `facet-id-constraints` that contains the key value pairs for each facet id and its constraint. The id should correspond to the facet id in the AQL. Here is a code sample that shows you facet results and the `facet-id-constraints` parameter:

Example 4-38: Facet results and constraints

```
<feed>
  <entry>
    ...
  </entry>
  <dm:facet
    xmlns:dm="http://identifiers.emc.com/vocab/documentum">
```

```

<dm:facet-id>facet_type</dm:facet-id>
<dm:facet-label>Type</dm:facet-label>
<dm:facet-value>
    <dm:facet-id>facet_type</dm:facet-id>
    <dm:facet-value-id>dm_document</dm:facet-value-id>
    <dm:facet-value-count>30</dm:facet-value-count>
    <dm:facet-value-constraint>dm_document</dm:facet-value-constraint>
    <link rel="search" href="http://localhost:8080/dctm-rest/repositories/
        REPO/search?facet-id-constraints=facet_type%3Ddm_document"/>
</dm:facet-value>
<dm:facet-value>
    <dm:facet-id>facet_type</dm:facet-id>
    <dm:facet-value-id>dm_cabinet</dm:facet-value-id>
    <dm:facet-value-count>5</dm:facet-value-count>
    <dm:facet-value-constraint>dm_cabinet</dm:facet-value-constraint>
    <link rel="search" href="http://localhost:8080/dctm-rest/repositories/
        REPO/search?facet-id-constraints=facet_type%3Ddm_cabinet"/>
</dm:facet-value>
</dm:facet>
</feed>

```



As you can see in the preceding code sample, the `<cabinet>` type has five results. You can navigate into this facet group by using the POST HTTP method with the original request to the URL in the facet result `http://localhost:8080/dctm-rest/repositories/REPO/search.xml?facet-id-constraints=facet_type%3Ddm_cabinet`.

For more information on using Facets requests with AQL, see *OpenText Documentum Content Management - Foundation REST API Reference Guide (EDCPKRST250400-PRE)* for version 7.3 or later.

4.12 Lightweight and shareable objects

A lightweight object type is optimized to reduce the storage space needed in a database. When a lightweight SysObject is created, it references a shareable `<supertype>` object. Additional lightweight SysObjects that are created also reference the same shareable object.

The shareable `<supertype>` object is called the lightweight `<SysObject parent>`. The lightweight SysObject that references the `<supertype>` is called the child.

Each lightweight SysObject shares the information of its shareable parent object. Instead of having multiple, nearly identical, rows in the SysObject database tables to support all instances of the lightweight type, a single parent object exists for multiple lightweight SysObjects.

Shareable Type

Shareable type instances can share their property values with lightweight types. Multiple lightweight objects can share the property values of one shareable object.

Lightweight Type

Lightweight object type is a child of shareable object type. It is used to reduce the space required to store it in a database.

4.12.1 Get lightweight and shareable types

Lightweight and shareable types can be retrieved by an existing Types resource.

To retrieve a lightweight and shareable type:

1. Use the filter attribute *type_category* on the Types resource to get lightweight object types. Here is an example of the URI:

```
/repositories/<{repositoryName}>/types?filter=type_category=4
```



Note: You must include your repository name in the URI. Also notice the parameter *filter=type_category=4*, which is used for Lightweight object types.

2. Use the filter attribute *type_category* on the Types resource to get sharable object types. Here is an example of the URI:

```
/repositories/<{repositoryName}>/types?filter=type_category=2
```



Note: You must include your repository name in the URI. Also notice the parameter *filter=type_category=2*, which is used for shareable object types.

3. The shareable object types have a *lightweight-types* link relation that points to their relative child lightweight object type.

Similarly, the lightweight object types have a link relation *parent-shareable-type* that points to their parent shareable object type. The link relation *<parent>* points to the super type.

4. The *category* property has been added to the Type resource version 7.3 and later to show the type of the category, and the *shared-parent* property has also been added to the Type resource version 7.3 and later to show the *shared-parent* of the lightweight type. Here's a code sample that shows you how to use these two properties:

```
<type label="MyLightweight" name="my_lightweight"
      parent="http://localhost:8080/dctm-rest/repositories/REPO/types/my_shareable"
      shared-parent="http://localhost:8080/dctm-rest/repositories/REPO/types/
      my_shareable"
      category="shareable"
      xmlns="http://identifiers.emc.com/vocab/documentum"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <properties></properties>
  <links>
    <link
      href="http://localhost:8080/dctm-rest/repositories/REPO/types/
      my_lightweight"
      rel="self"/>
    <link
      href="http://localhost:8080/dctm-rest/repositories/REPO/types?
      parent-shareable-type=my_shareable" rel="http://identifiers.emc.com/
      linkrel/types"/>
    <link
      href="http://localhost:8080/dctm-rest/repositories/REPO/types/
```

```
my_shareable"
      rel="parent" />
  </links>
</type>
```

4.13 Configuring DQL resource access modes

The *dql.disallowed.types* property provides a facility to define the DQL resource access modes. This property supports the following DQL access modes:

- **admin:** This is the default mode where only the users having superuser or administrator privileges can access the DQL resource.
- **group:** In this mode, the set of users that are part of the specific groups (the dm_groups created at Documentum CM Server) can access the DQL resource. These specific groups can be configured using the *rest.dql.access.groups* property. This property can take multiple group names as comma-separated values.
- **all:** This mode provides the backward compatibility with the DQL resource where any user can access the DQL resource. The *dql.disallowed.types* property is valid only when the access mode is configured to all.

The property, *dql.disallowed.types*, in the *rest-api-runtime.properties* file restricts the DQL execution for non-sysobjects on which ACL cannot be defined. This property provides a facility to configure OpenText Documentum CM types on which a DQL execution can be restricted for a user whose basic privileges are less than eight or not an administrator or superuser. OpenText recommends that you use this property to avoid DQL-related security concerns either on OpenText Documentum CM configuration objects (such as docbase config, server config, bocs config, and so on) or the types on which ACL cannot be defined (such as dm_user, dm_group, dm_content, and so on).

By default, the Foundation REST API server does not restrict any user to execute a DQL on any type. The applications can choose to set either all the type names or some of them, and then provide them as values to the types such as dm_user, dm_group, dm_content, and so on. The type names must be provided as comma-separated values.

DQL execution can be restricted for the following types:

dmr_content, dm_relation, dm_acs_config, dm_cont_transfer_config,
dm_network_location_map, dm_cache_config, dm_cache_config,
dmc_mq_config, dm_bocs_config, dm_dms_config, dm_docbase_config,
dm_server_config, dm_user, dm_acl, dm_role, dm_group, dmr_containment,
dm_assembly, dm_tasks_all, dmi_package, dmi_queue_item, dm_format,
dm_mount_point, dm_store, dm_type, dm_process, dm_workflow,
dmi_auditAttrs, dm_java, dm_basic, dm_public_key_certificate, dmi_type_info,
dmc_java_library, dmi_change_record, dm_func_expr, dm_location,
dmi_dd_attr_info, dm_relation_ssa_policy, dm_aggr_domain,
dmi_change_record, dm_validation_descriptor, dm_client_registration,
dm_replica_catalog, dmi_wf_attachment, dmi_registry, dmc_aspect_type,
dm_fulltext_index, dm_lifecycle, and dmc_java_library.

For additional information about DQL ingestion, see “[Additional restrictions for all access mode](#)” on page 69.



Note: The `rest.dql.access.mode` property provides a facility to configure the set of groups as comma-separated values and is valid only when the value is set to group. The users that are part of the groups provided to this property are only allowed to access the DQL resource.

Detect and Prevent unusual query patterns for all access modes admin, group, and all

The DQL query execution is restricted, if the following query patterns exists:

- True predicates such as `1 = 1`, `TRUE = TRUE`, `TRUE = 1`.
- False predicates such as `FALSE = FALSE`, `FALSE = 0`.
- Numerical comparisons such as `10 < 20` or `5 > 1`.

4.13.1 Additional restrictions for all access mode

To enforce additional threat validations, configure the `rest.dql.restricted.validation.pattern` property in the `rest-api-runtime.properties` file with the required patterns.

select_all

Prevents the use of the asterisk (*) in SELECT queries, requiring users to specify the required attributes explicitly.

For example: Use

```
SELECT object_name, r_creation_date FROM dm_document
```

instead of

```
SELECT * from dm_document
```

missing_where,, wildcard

Prevents bulk data access by detecting and restricting it using the WHERE clause in SELECT queries.

For example: Use

```
SELECT object_name, r_creation_date FROM dm_document WHERE object_name LIKE '%doc%'
```

instead of

```
SELECT object_name, r_creation_date FROM dm_document WHERE object_name = 'doc123'
```

keywords

Prevents the use of IN or UNION or JOIN in DQL queries to prevent access to multiple tables while using sub queries.

all

Enables all the preceding DQL threat validations.

4.14 Generating link relation in DQL results

To facilitate content consumption, Foundation REST API generates link relations, including thumbnail links, in DQL results. Whether to generate links and what links to generate is based on the criterion you specify in a DQL query request, such as the properties to select and types of resources to select from.

Generated links are presented at the entry level. By accessing these links, a client application is able to navigate to the corresponding resources. Typically, if the DQL query result item has a dedicated resource, the result entry contains an `editlink` relation pointing to that resource.

 **Example 4-39: XML Representation of query results with links generated**

```
<entry>
<id>...</id>
<title>090007c2800001dc</title>
<updated>...</updated>
<content>...</content>
<links>
<!!-- all object related links for this query result is added here -->
<links>
</entry>
```



 **Example 4-40: JSON Representation of query results with links generated**

```
{
  "id": "...",
  "title": "...",
  "updated": "...",
  "content": {...},
  "links": [## all object related links for this query result is added here ##]
}
```



4.14.1 Additional information about generating links

When a client application submits a request that contains a DQL statement, the Foundation REST API server does not make any change to the DQL. The query resource performs a simple parse to obtain the types needed for link generation from the DQL.

DQL query syntax:

```
SELECT [FOR base_permit_level][ALL|DISTINCT] value [AS name] {,value [AS name]}
FROM [PUBLIC] source_list
[WITHIN PARTITION (partition_id{,partition_id})
| IN DOCUMENT clause
| IN ASSEMBLY clause]
[SEARCH [FIRST|LAST]fulltext_search_condition
[IN FTINDEX index_name{,index_name}]]
[WHERE qualification]
[GROUP BY value_list]
[HAVING qualification]
[UNION dql_subselect]
```

```
[ORDER BY value_list]
[ENABLE (hint_list)]
```

How and what links are generated is delegated to each resources (mostly the types you specify in the FROM clause, such as a document resource).

Note that not all DQL statements are eligible to generate links in query results as a DQL query may not return information needed to generate links. For example, properties such as `r_object_id`, or `user_name`, which can be used to identify resources, are not selected in the DQL statement. The following query does not generate any link because neither `object_name` nor `r_modify_date` is able to identify a resource.

```
select object_name, r_modify_date from dm_document
```

Additionally, if you query a type that pertains to no existing resource in Foundation REST API, no link is generated. Similarly, if you query multiple types that are not relevant with one another, no link is generated. The following query does not generate any link:

```
select * from dm_document, dm_user
```

4.14.2 Thumbnail support

When the query result contains the `thumbnail_url` attribute, the system uses the value of this attribute to generate a link to the [Thumbnail](#).

Example 4-41: Query results with thumbnail links generated in XML

Sample request:

```
GET http://localhost:8080/dctm-rest/repositories/REPO?dql=select
r_object_id,object_name,r_object_type,thumbnail_url from dm_sysobject
```

Sample Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns="http://www.w3.org/2005/Atom"
      xmlns:dm="http://identifiers.emc.com/vocab/documentum"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <id>http://localhost:8080/dctm-rest/repositories/REPO</id>
  <title>query results</title>
  <updated>2014-05-20T15:09:18.473+08:00</updated>
  <author>
    <name>OpenText Documentum</name>
  </author>
  <dm:page>1</dm:page>
  <dm:items-per-page>100</dm:items-per-page>
  <link
      href="http://localhost:8080/dctm-rest/repositories/REPO?
            select+r_object_id,object_name,r_object_type,
            thumbnail_url+from+dm_sysobject"
      rel="self"/>
  <link
      href="http://localhost:8080/dctm-rest/repositories/REPO?
            select+r_object_id,object_name,r_object_type,
            thumbnail_url+from+dm_sysobject&items-per-page=100&page=2"
      rel="next"/>
  <link
```

```

    href="http://localhost:8080/dctm-rest/repositories/REPO?
    select+r_object_id,object_name,r_object_type,
    thumbnail_url+from+dm_sysobject&items-per-page=100&page=1"
    rel="first"/>
<entry>
<id>http://localhost:8080/dctm-rest/repositories/REPO?
    select+r_object_id,object_name,r_object_type,
    thumbnail_url+from+dm_sysobject&index=0</id>
<title>080020808000013c</title>
<updated>2014-05-20T15:09:18.763+08:00</updated>
<content>
<dm:query-result
    definition="http://localhost:8080/dctm-rest/repositories/REPO/types/
        dm_cryptographic_key"
    xsi:type="dm:dm_cryptographic_key">
<dm:properties xsi:type="dm:dm_cryptographic_key-properties">
<dm:r_object_id>080020808000013c</dm:r_object_id>
<dm:object_name/>
<dm:r_object_type>dm_cryptographic_key</dm:r_object_type>
<dm:thumbnail_url>http://CS71P01:8081/thumbsrv/getThumbnail?
    object_type=dm_cryptographic_key&
    format=&
    is_vdm=false&
    repository=8320
</dm:thumbnail_url>
</dm:properties>
</dm:query-result>
</content>
<link
    href="http://localhost:8080/dctm-rest/repositories/REPO/objects/
        080020808000013c"
    rel="edit"/>
<link
    href="http://localhost:8081/thumbsrv/getThumbnail?
        object_type=dm_cryptographic_key&
        format=&
        is_vdm=false&
        repository=8320"
    rel="icon"/>
</entry>
</feed>
```



Example 4-42: Query results with thumbnail links generated in JSON

Sample request:

```
http://localhost:8080/dctm-rest/repositories/documentum1?dql=
select r_object_id,object_name,r_object_type,thumbnail_url from dm_sysobject
```

Sample Response:

```
{
  id: "http://localhost:8080/dctm-rest/repositories/documentum1"
  title: "DQL query results"
  updated: "2014-07-04T15:44:00.896+08:00"
  author:
  {
    name: "OpenText Documentum"
  }
  page: 1
  items-per-page: 2
  links:
  {
    rel: "self"
    href: "http://localhost:8080/dctm-rest/repositories/documentum1?"
```

```

        dql=select r_object_id,object_name,
        r_object_type,
        thumbnail_url from dm_sysobject"
    }
{
    rel: "next"
    href: "http://localhost:8080/dctm-rest/repositories/documentum1?
        dql=select r_object_id,object_name,
        r_object_type,
        thumbnail_url from dm_sysobject"
}
{
    rel: "first"
    href: "http://localhost:8080/dctm-rest/repositories/documentum1?
        dql=select r_object_id,object_name,
        r_object_type,
        thumbnail_url from dm_sysobject"
}
entries: [
{
    id: "http://localhost:8080/dctm-rest/repositories/documentum1?
        dql=select r_object_id,object_name,
        r_object_type,
        thumbnail_url from dm_sysobject"
    title: "080f42418000013c"
    updated: "2014-07-04T15:44:00.896+08:00"
    content:
    {
        name: "query-result"
        type: "dm_cryptographic_key"
        definition: "http://localhost:8080/dctm-rest/repositories/
                    documentum1/types/dm_cryptographic_key"
        properties:
        {
            r_object_id: "080f42418000013c"
            object_name: ""
            r_object_type: "dm_cryptographic_key"
            thumbnail_url: "http://localhost:8081/thumbsrv/getThumbnail?
                            object_type=dm_cryptographic_key&
                            format=&
                            is_vdm=false&
                            repository=1000001"
        },
        links: [...]
    }
    links: [
    {
        rel: "edit"
        href: "http://localhost:8080/dctm-rest/repositories/documentum1/
                objects/080f42418000013c"
    }
    {
        rel: "icon"
        href: "http://localhost:8081/thumbsrv/getThumbnail?
                object_type=dm_cryptographic_key&
                format=&
                is_vdm=false&
                repository=1000001"
    }
    ]
},
{
    id: "http://localhost:8080/dctm-rest/repositories/documentum1?
        dql=select r_object_id,object_name,r_object_type,
        thumbnail_url from dm_sysobject"
    title: "080f42418000013d"
    updated: "2014-07-04T15:44:00.896+08:00"
    content:
    {
        name: "query-result"
    }
}
]

```

```

        type: "dm_public_key_certificate"
        definition: "http://localhost:8080/dctm-rest/repositories/
                      documentum1/types/dm_public_key_certificate"
        properties:
        {
            r_object_id: "080f42418000013d"
            object_name: ""
            r_object_type: "dm_public_key_certificate"
            thumbnail_url: "http://localhost:8081/thumbsrv/getThumbnail?
                            object_type=dm_public_key_certificate&
                            format=&
                            is_vdm=false&
                            repository=1000001"
        }
        links: [...]
    }
    links: [
        {
            rel: "edit"
            href: "http://localhost:8080/dctm-rest/repositories/
                    documentum1/objects/080f42418000013d"
        },
        {
            rel: "icon"
            href: "http://localhost:8081/thumbsrv/getThumbnail?
                    object_type=dm_public_key_certificate&
                    format=&
                    is_vdm=false&
                    repository=1000001"
        }
    ]
}

```



4.15 Location of persistent data

Some resources found in Foundation REST API release 7.3 and later, such as the Saved Search resource and the User Preference resource, must create internal persistent data in the repository. The correct configuration of a persistent data folder is required to prevent data access errors.

Foundation REST API decides where to store persistent data depending on your runtime environment setting, the user accessing the persistent data, and the resource generating the data. This section discusses how Foundation REST API resolves locations of persistent data.

The path of the folder storing persistent data consists of two parts, a parent folder path followed by a sub folder path.

- Parent folder path: This folder path can point to a global location specified in the `rest-api-runtime.properties` file, the user's default folder, or the temp folder.

For more information, see [Resolving Parent Folder](#).

- Sub folder path: The path of a sub folder follows this pattern: `${user_name}_ ${resource_name}`.

For more information, see [Resolving Sub Folder](#).

▶ **Example 4-43: Parent folder and sub-folder paths**

Here are some examples of persistent data folder paths. The parent folder path, which is shown in italic, followed by the sub-folder path:

- */System/Applications/rest-persistence/admin_saved_search*
- */testuser_default/testuser_saved_search*
- */temp/myuser_myresource*



4.15.1 Resolving a parent folder

When resolving the parent folder of persistence data, Foundation REST API has the following options. These options are listed in the order of precedence, with the highest-precedence at the top.

- Global location specified in `rest-api-runtime.properties`
- User's default folder
- `/Temp`

If none of the options is enabled, the system returns a 403 error code to the client that attempts to generate persistent data.

Global location

The global location is the root folder for all persistence data. With this location specified, operations generating persistent data create sub folders under the root to store persistence.

The runtime property `rest.persistence.folder` specifies the global location of persistence data. For example:

```
rest.persistence.folder=/System/Applications/rest-persistence
```

If you specify an invalid path, the system returns a 403 error code. In this case, the system does not try the two lower precedence options (user default folder or `/Temp` folder).

Make sure that you grant users Write permission on this folder as they have to create sub folders there. To do this, set the basic permission of `dm_world` to 6 for the global location folder as shown in the following snippet.

```
<permission-set
    xmlns="http://identifiers.emc.com/vocab/documentum"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <permitted>
        <permission accessor="dm_world" basic-permission="6"/>
        <permission accessor="dm_owner" basic-permission="7"
            extend-permissions="EXECUTE_PROC,CHANGE_LOCATION"/>
    </permitted>
    <links>
```

```
<link rel="self" href="http://localhost:8080/" />
<link rel="edit" href="http://localhost:8080/" />
<link rel="http://identifiers.emc.com/linkrel/acl"
      href="http://localhost:8080/repositories/REPO/acls/45024c518000110c" />
</links>
</permission-set>
```

For security reasons, you may want to hide implementation details. Marking the global location folder hidden serves the purpose:

```
<folder
  xmlns="http://identifiers.emc.com/vocab/documentum"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:type="dm_folder"
  definition="http://localhost:8080/repositories/REPO/types/dm_folder">
  <properties>
    <object_name>rest-persistence</object_name>
    <r_object_type>dm_folder</r_object_type>
    ...
    <a_is_hidden>true</a_is_hidden>
    ...
  </properties>
</folder>
```

User's default folder

If a global location for persistent data does not exist, Foundation REST API tries the user' default folder (specified by `default_folder`) as an alternative.

Temp folder

Non-administrative users may not have a default folder. In this case, Foundation REST API tries the `/Temp` folder. Because the `/Temp` folder can be used as the storage for persistent data, be cautious when you delete this folder or remove data from the folder.

4.15.2 Resolving sub-folder

Once the parent folder is resolved, Foundation REST API checks whether the parent folder has a sub-folder `${user_name} ${resource_name}` to save persistent data. If the sub folder exists, Foundation REST API returns the path `parenet_folder\sub_folder` to the client for persistent data storage. Note that the user initiating the operation must be granted Write permission on the sub folder. If the sub folder does not exist, the client creates the sub folder on behalf of the user. If the user does not have the writer permission on the parent folder, the system throws an exception.

4.15.3 Global location change

When you change the runtime property `rest.persistence.folder`, only persistent data generated after the change is stored in the new location. Old persistent data stays in the original global location folder. However, when you update a resource, such as the `<Saved search>` or the `<User preference>` resource that generates persistent data after a global location change, the data store location does not change.

4.16 MailApp support

The Foundation REST API *MailApp* feature requires a little set up before you can use it in your server side code. Perform the following tasks in your Foundation REST API server side code to set up the MailApp feature:

- Your Foundation REST API clients must send email content as multi part form data to the Foundation REST API server.
- Your Foundation REST API clients must send content to your Foundation REST API server as an *HTTP query* parameter with the *format* value set to either `msg` or `eml` as shown in this example:

```
Ex: http://localhost:8090/dctm-rest/repositories/documentum1/folders/
0c0004d280000107/documents?content-count=1&content_length=359424&
format=msg
```

- Set the `rest.should.parse.emails` property in the `rest-api-runtime` file to `true` in your Foundation REST API server side code. This default value for this property is `false`.

When the preceding items have been set, the Folder child documents collection resource and Folder child objects collection resource will use the Foundation Java API to invoke the *MailApp* logic in the Foundation REST API server side code. The mail app logic parses the attributes of the email and updates the email *SysObject* that must be imported. Once the email import operation is finished, you can find the email attributes, such as `subject`, `to`, `from`, and so on in the *SysObject* of the mail object that is being imported.

MailApp processing depends on the values used in the following properties in the `mail-app.properties` file.

- `shouldSeparateAttachments=true`: When this flag is `true`, attachments are separated from mail. Otherwise not.
- `objectTypeForAttachments=dm_document`: The object type of attachments must be `dm_document` or a subtype of `dm_document`, except for `dm_message_archive` and its subtypes.
- `shouldParseMsgFile=true`: When this flag is `true`, then `msg` files are parsed by the mail app. Otherwise not.
- `shouldSkipDuplicateCheck=true`: When this flag is `true`, then duplicate mails (MSG) can be imported.

4.17 Constraints validation of objects

Constraints validation of objects are done using Foundation Java API `object validator` derived from `SysObject`. This validation is a mechanism to verify the following constraints or rules that are defined while creating or updating a type:

- Attribute constraints: Constraints or rules defined on an attribute of a type while creating or updating the type. The constraints can be categorized as not null, list, expression and so on.
- Object constraints: Constraints or rules defined at the type level while creating or updating the type.

In the `object validator` method, the validation is performed while creating or updating an object of a type. The values of an object's attribute are validated with the respective attributes and objects constraints when creating or updating the object.

Foundation REST API supports the following types of criteria:

1. Validate the constraints or rules defined on attributes, irrespective of whether the values of these attributes are changed.
2. Validate the constraints or rules defines on attributes whose values have changed or modified.
3. Validate the object or type level constraints.
4. Combination of criteria [1 on page 78](#) and [3 on page 78](#).
5. Combination of criteria [2 on page 78](#) and [3 on page 78](#).

The preceding validation criteria can be applied at Foundation REST API, based on the values chosen from one of the following list of validation policy values:

1. `attrs`: Criterion [1 on page 78](#) is applied.
2. `modified_attrs`: Criterion [2 on page 78](#) is applied.
3. `obj`: Criterion [3 on page 78](#) is applied.
4. `obj_and_attrs`: Criterion [4 on page 78](#) is applied.
5. `obj_and_modified_attrs`: Criterion [5 on page 78](#) is applied.
6. `not_validate`: Does not validate `SysObject` attribute values. This is the default value.

The two types of configuration for validation are:

- Static:

The validation is performed based on the valid validation policy value chosen from [1 on page 78](#) through [6 on page 78](#) assigned for the `rest-api-runtime` property named `rest.constraints.validation.method`. This validation helps

the existing Foundation REST API client or customer applications to leverage the validation feature without changing their application code.

As the configuration is static, the Foundation REST API client does not know either to choose or not to choose the validation for a specific object. However, the Foundation REST API client can limit the validation to the set of types specified for the `rest.constraints.validation.allowed.types` property in the `rest-api-runtime` properties file.

You can provide multiple types as comma-separated values. The default value is `all` and is considered only when a valid value is chosen for the `rest.constraints.validation.method` property.

- Dynamic:

The validation is performed based on the valid validation policy value provided for the `constraint-validation-policy` query parameter while creating or updating the object.



Note: The dynamic configuration always takes the precedence over the static configuration.

The Foundation REST API controllers that support the validation are:

- FolderChildrenDocumentsController
- FolderChildrenObjectsController
- FolderChildrenFolderController
- SysobjectController
- CabinetsController
- CabinetController
- FolderController
- DocumentController

4.18 Adding a document with content to Foundation REST API

There are two ways in which you can add a document, with content, to Foundation REST API. The first approach is to create a document object and add content to the new document object using two separate requests. The second approach is to use a single request to create the document and its primary content.

4.18.1 Creating a document and adding its content

Follow these steps to create a document and add content to it:

1. Navigate to a specific document *Cabinet* or *Folder* resource. You can do this using the following *GET* Request:

```
GET http://localhost/dctm-rest/repositories/documentum1/cabinets/0c00031280000105
```

2. Create a *Document* object in the selected *Cabinet* or *Folder* with the following *POST* request:

```
POST  
http://localhost:8080/dctm-rest/repositories/documentum1/folders/  
0c00031280000105/documents
```

This is your *POST* request body:

```
{ "properties":{ "object_name": "newTestDocument" }}
```

3. Add content to the new *Document* object with the following *POST* request:

```
POST: http://localhost:8080/dctm-rest/repositories/documentum1/objects/  
0900031280002f60/contents
```

```
Content-Disposition: form-data; name=content  
Content-Type: text/plain This sentence is the text content of your Document object
```

4.18.2 Creating a document and its content in a single request

To create a *Document* object and its primary text content in a single request, create and send the following POST Request:

```
POST: http://localhost/dctm-rest/repositories/documentum1/folders/  
0c00031280000105/documents  
Content-Disposition: form-data;  
name=properties  
Content-Type: text/plain  
  
{  
    "properties": {  
        "object_name": "newTestDocument1"  
    }  
}  
name=content  
Content-Type: text/plain This sentence is the text content of your Document object
```

4.18.3 Code samples

Example 4-44: Creating a Document object and its content with two requests

The following code sample shows you how to create a *dm_document* (*dmObject*) object and set its *object_name*. Next, we show you how to create a *binaryContentObject*, which is added to your new *dm_document* object as content.

```
ObjectViewBuilder builder = (ObjectViewBuilder) params[1];  
String targetUrl = "http://localhost:8080/dctm-rest/repositories/documentum1/  
folders/0c00031280000105/documents";  
Object binaryContentObject = "This is used as binary content for your new object  
    <script>alert('hello world');</script> 1";  
String objectName = "test_doc";
```

```

Object dmObject = appendDocBasicAttr(builder, objectName).build();

private ObjectViewBuilder appendDocBasicAttr(ObjectViewBuilder builder,
                                             String objectName)
{
    return builder
        .root("document")
        .type("dm_document")
        .attribute("object_name", objectName)
        .attribute("r_object_type", "dm_document");
}

```



Example 4-45: Creating a Document object and its content with one request

The following code sample shows you how to create a *Document* object and its content with one request:

```

Item createdObjectWithContent = client.createDocument(targetUrl, dmObject,
    binaryContentObject, "text/plain", "format", "crtext");

public RestObject createDocument(String url, Object objectToCreate,
    Object content, String contentMediaType) // More parameters are acceptable
{
    return post(url, new JSONObject(objectToCreate), content, contentMediaType,
        JSONObject.class, params);
}

protected <T> T post(String uri, T object, Object content,
    String contentMediaType, Class<? extends T>
    responseBodyClass) // More params are acceptable
{
    MultiValueMap<String, Object> parts = new LinkedMultiValueMap<String, Object>();
    parts.add("metadata", new HttpEntity<Object>(object, isXml()?
        XML_CONTENT:JSON_CONTENT));
    parts.add("binary", contentMediaType == null?content:new HttpEntity<Object>
        (content, new Headers().contentType(contentMediaType).toHttpHeaders()));
    T t = sendRequest(uri, POST, isXml()?ACCEPT_XML_HEADERS_WITH_MULTIPART_CONTENT:
        ACCEPT_JSON_HEADERS_WITH_MULTIPART_CONTENT, parts,
        responseBodyClass, params);
}

protected <T> T sendRequest(String uri, HttpMethod httpMethod,
    HttpHeaders headers, Object requestBody,
    Class<T> responseBodyClass)// More params are acceptable
{
    HttpEntity<Object> requestEntity = newRequestEntity(headers, requestBody);
    ResponseEntity<T> entity = processor.process(requestUri, httpMethod,
        requestEntity, responseBodyClass);
}

```

To complete this task, we show you how to use the *Spring RestTemplate* to create the single *POST* request that is used to send everything to the Foundation REST API server:

```

public <T> ResponseEntity<T> process(String url, HttpMethod method,
    HttpEntity<?> requestEntity, Class<T> responseType)
{
    return restTemplate.exchange(url, method, requestEntity, responseType);
}

```



Chapter 5

Authentication

Foundation REST API supports the following authentication schemes:

- HTTP basic authentication for inline users
- OAuth 2.0 authentication
- Pre-authenticated authentication for Web Access Management solutions

 **Note:** In addition to these out-of-the-box authentication schemes, Foundation REST API also allows you to create your own custom authentication schemes using the extensibility library.

For more information, see “[Authentication extensibility](#)” on page 313.

5.1 End user tracking

Foundation REST API supports the end user tracking feature from release 21.2. As part of end user tracking, Foundation REST API provides the following information to the Documentum CM Server using Foundation Java API.

- End user machine IP address.
- End user location (this can be an exact location or geographic location), that the Foundation REST API clients pass to the Foundation REST API server as a value in the X-CLIENT-LOCATION custom header.

 **Note:** The Client_Location custom header is deprecated from Foundation REST API 21.3 release onwards and is replaced with X-CLIENT-LOCATION.

- The application used by the end user to log in to the Foundation REST API server.
- The Documentum REST client applications can provide their application name using the header the X-CLIENT-APPLICATION-NAME. The header value is considered as the OpenText Documentum CM client application name using which a user tries to login. If a value is not provided to this header, the application name is taken from the Documentum REST application context path.

The end user information is logged in the dm_audittrail table when the events, dm_connect and dm_logon_failure are enabled on Documentum CM Server. These events are triggered by Documentum CM Server when a user tries to get a session from Documentum CM Server. The audit event for dm_connect is not enabled by default. However, the dm_logon_failure event is enabled by default. The Foundation REST API server sends the end user information to Foundation Java API

if the `dfc.client.should_use_enduserinfo` Foundation Java API property is set to true.

5.2 HTTP basic authentication

HTTP basic authentication sends usernames and passwords to the Foundation REST API server for authentication.



Note: We strongly recommend that you use HTTPS, which requires a secure connection, when using HTTP basic authentication because HTTP basic authentication transmits the username and password in BASE64 encoded plain text.

HTTP basic authentication is typically used in the following scenario:

Scenario A (inline users):

- In your production environment, you do not have an identity provider (IdP).
- In your production environment, it is acceptable to store user credentials in OpenText Documentum CM repositories.

When using HTTP basic authentication, a request carries the username/password pair in the Authorization Header with the following pattern:

```
Authorization: Basic BASE64(${username}:${password})
```



Example 5-1: Username/password pair for dmadmin/password

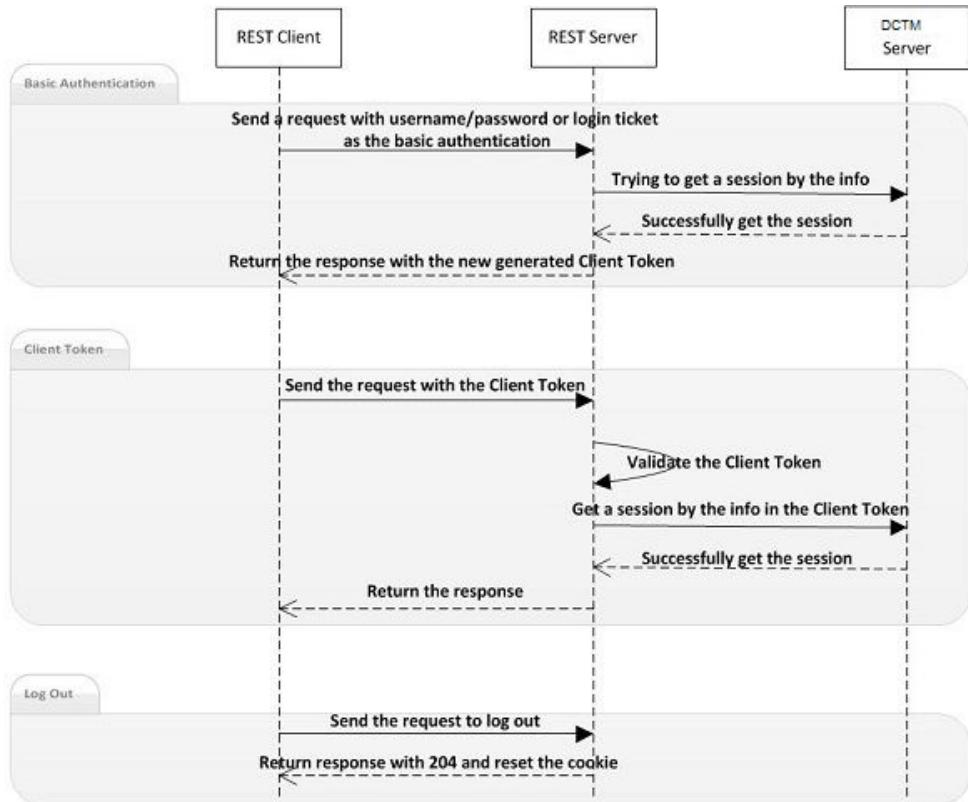
```
GET /${resource-url} HTTP/1.1  
Authorization: Basic ZG1hZG1pbjpwYXNzd29yZA==
```



If the authentication fails, the Foundation REST API server responds with the HTTP 401 Unauthorized status code.

Foundation REST API release 7.2 and later allow you to improve the efficiency of HTTP Basic authentication by using [client tokens](#). This improvement allows an authenticated Foundation REST API client to access the Foundation REST API server without having to negotiate a new session ticket in a specified period of time.

The following diagram illustrates the workflow of HTTP Basic authentication with client tokens:



1. A client sends the Request with HTTP Basic authentication credentials.

```

GET /dctm-rest/repositories/REPO HTTP/1.1
Host: localhost:8080
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.3.1 (java 1.5)
Authorization: Basic ZG1hZG1pbjpwYXNzd29yZA==
  
```

2. The Foundation REST API server tries to retrieve a session from Documentum CM Server with the credential the client provides. If the credential is valid, the Foundation REST API server creates a client token in the DOCUMENTUM-CLIENT-TOKEN cookie of the Response and sends it back to the client.

```

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: DOCUMENTUM-CLIENT-TOKEN= <encrypted_client_token>
  
```

Otherwise, the Foundation REST API server rejects the Request with an HTTP 401 error.

3. When sending subsequent requests, the client includes the client token in the cookie.

```

GET /dctm-rest/repositories/REPO HTTP/1.1
Host: localhost:8080
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.3.1 (java 1.5)
Cookie: DOCUMENTUM-CLIENT-TOKEN= <encrypted_client_token>
  
```

4. The Foundation REST API server validates the client token. If the client token is valid, the server returns the Response. If the client token expires or the token is invalid, the Foundation REST API server rejects the Request with HTTP 401
5. The client sends a request to log out:

```
GET /dctm-rest/logout HTTP/1.1
Host: localhost:8080
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.3.1 (java 1.5)
Cookie: DOCUMENTUM-CLIENT-TOKEN= <encrypted_client_token>
```

For more information, see [Explicit Logoff](#).

6. The Foundation REST API server reset the client token and returns HTTP 204:

```
HTTP/1.1 204 No Content
Server: Apache-Coyote/1.1
Set-Cookie: DOCUMENTUM-CLIENT-TOKEN="" ; Expires=Thu,
01-Jan-1970 00:00:10 GMT; Path=/dctm-rest; HttpOnly
Date: Thu, 06 Nov 2014 06:35:33 GMT
```



Note: If an authenticated client provides both HTTP basic authentication credentials and a client token, the Foundation REST API server ignores the client token, validating the credentials only. If the credentials are valid, a new client token is generated.

5.2.1 Enabling HTTP basic authentication

HTTP Basic is the default authentication scheme in Foundation REST API, and thus it is enabled out of the box. If you want to enable client tokens in HTTP Basic Authentication, set the `rest.security.auth.mode` property to `basic-ct` in `<dctm-rest>\WEB-INF\classes\rest-api-runtime.properties`.



Note: `<dctm-rest>` represents the directory where you extract the `dctm-rest.war` file.

5.2.2 User credential mapping

In scenario A, user credentials are mapped as follows:

```
 ${username} = ${dm_user.user_login_name}
 ${password} = ${dm_user.user_password}
```

OpenText recommends that you use the following mapping rule consistently for both Documentum CM Server domain-required and non-domain-required modes.

```
 ${username} = ${dm_user.user_login_domain}\${dm_user.user_login_name}
 ${password} = ${user password}
```

If your Documentum CM Server repository is configured with the domain-required authentication model, it is possible to have multiple users with the same login name if each user is in a different domain. Therefore, under domain-required authentication model, it is required to put the domain name prefix in the `username` variable. If domain-required authentication model is not enabled (default setting), the domain name prefix is optional in the `username` variable.



Note: In Documentum CM Server, you can create a user with a login name that contains the backslash sign ('\\'), for example user alpha\\beta, where alpha is not a domain name. To make such an authentication succeed, the Foundation REST API client must insert an empty domain name to the user login name, for example, \\alpha\\beta. This is a known limitation in Documentum CM Server.

5.2.3 Known limitations

According to RFC2617, section-2, user names for HTTP basic authentication cannot contain the colon character (':').

5.3 OAuth 2.0 authentication

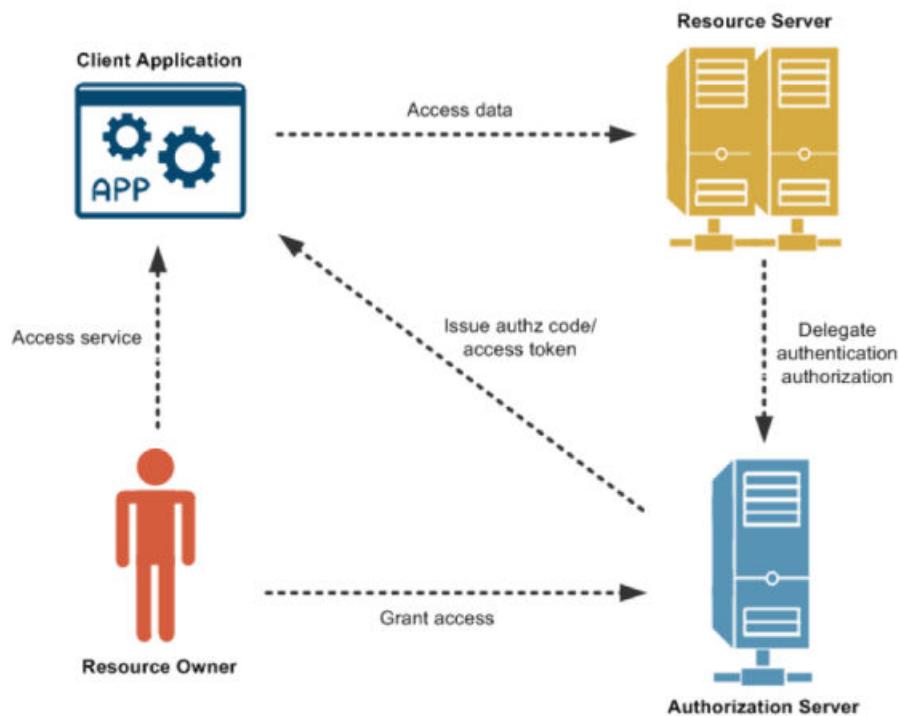
The OAuth 2.0 authorization framework is the industry-standard protocol for authorization. It enables a third-party application to obtain limited access to an HTTP service. Foundation REST API supports OAuth 2.0 Authorization Framework and can be integrated with popular authorization servers, such as ADFS, Azure, OTDS, and so on.

Four main participants in the Foundation REST API OAuth 2.0 scenario:

- OAuth Server—The server issues access tokens to the client after successfully authenticating the resource owner and obtaining authorization.
- Documentum Client Application—The client application that requests to access protected resources on behalf of the resource owner.
- Foundation REST API—The protected resources exposed in REST API.
- Documentum CM Server—The back-end resource server for protected resource.

In addition to OAuth 2.0, OpenId Connect (OIDC) 1.0 is a simple identity layer on top of OAuth 2.0 protocol, which allows the relying parties to exchange and verify user identities along with OAuth 2.0 tokens. Foundation REST API also support to verify OAuth access token by OIDC protocol.

Besides these two protocols, DOCUMENTUM-CLIENT-TOKEN has been integrated with the authentication flow, which simplifies the authentication and authorization for authenticated Requests.



5.3.1 OpenText Directory Services integration with Documentum CM Server for REST

5.3.1.1 Overview

OpenText Directory Services (OTDS) is an authentication server that supports the standard OAuth 2.0 authorization framework, which is the industry-standard protocol for authorization. OAuth 2.0 enables a third-party application to obtain controlled access to an HTTP service.

To support OTDS integration with Documentum CM Server in REST, the following should be completed:

- Configure OTDS integration with Documentum CM Server.
- Configure Documentum CM Server to sync with OTDS.
- Configure the Foundation REST API server to work with OTDS and Documentum CM Server.

5.3.1.2 Prerequisites

5.3.1.2.1 Architecture flow

The architecture flow is as follows:

1. The Documentum CM Server combines with OTDS and shares users and groups.
2. The Foundation REST API server uses the Documentum CM Server as it's authentication and resource server.
3. A Foundation REST API client application sends a token to the Foundation REST API server who then passes it to the Documentum CM Server for authentication.
4. The Documentum CM Server authenticates the token in cooperation with OTDS.
5. REST gets the authentication response from Documentum CM Server.
 - When the token cannot be authenticated, Documentum CM Server throws an exception and the flow stops here.
 - When the token is authenticated, the Foundation REST API client is given access to the protected resource.

5.3.1.2.2 Configuration prerequisite

Before you can proceed, you must have the following in place:

- Documentum CM Server 16.4 P01 or later



Warning

Avoid using Documentum CM Server 16.4 P07 and 16.4 P08 for integration with the Foundation REST API server.

- Active Directory (AD) installed, preferably on a separate machine
- OTDS 16.2.3 or later installed with the `otds-admin` and the `otdsws` services started
- REST 16.4 P01 or later

5.3.1.3 Configure OTDS integration with Documentum CM Server

The following sections are the steps in the process to configure OTDS integration with Documentum CM Server. We recommend that you follow the steps shown here in the sequence in which they are presented.

5.3.1.3.1 The otds-admin service

Follow these steps to use the `otds-admin` service as the UI to configure the properties of OpenText Directory Services (OTDS):

1. Open a browser and enter `https://<otds-server>:8080/otds-admin` to open the UI used to configure the properties of OTDS.
2. Log in to OTDS with the administrative account, such as the default Administrator `otadmin@otds.admin` account.

The *OpenText Directory Services* documentation contains detailed instructions on how to configure all of the properties of OTDS.

5.3.1.3.2 Enable HTTP



Note: This step only applies to the simplified internal testing environment. When the environment must use HTTPS/SSL, as such is the case with OpenText Cloud, then omit this step and do not enable HTTP. Proceed directly from here to [Partitions](#).

When you want to use OTDS with HTTP, then HTTP must be enabled. Otherwise, you can skip this step:

1. Go to the **System Config** section in OTDS and click **Add Attribute**.
2. Enable HTTP by setting the `directory.auth.EnforceSSL` property to `false`.

5.3.1.3.3 Partitions

A custom partition containing the Active Directory (AD) server details, user and group filters, monitor protocol, and AD attributes that map to OTDS, must be created.

OpenText Directory Services documentation contains detailed information about partition.

Follow these steps to create a new partition in OTDS:

1. In OTDS, click on **SETUP** in the left-hand navigation to open the slide down menu.
2. In the left-hand slide down menu, click on the **Partitions** tab to go to the **Partitions** screen.
3. In the toolbar at the top of the **Partitions** screen, click on the **Add** button and select **New User Synchronized Partition** from the flyout menu.
4. Click the **Connection Information** tab. Enter the **Host name or address** and **Port** of your AD.
5. Click **Test Connection** and make sure that **Connected Successfully** is displayed in the status area above the button.

6. Click on the **Authentication** tab to view the authentication properties of the **67-ContentServer164** partition. Fill in the **User name** and **Password** of the Administrator account for your AD.
7. Click the **Test Authentication** button to make sure that this Administrator account is working as expected.
8. Click the **General** tab and enter a value in the **Name** and **Description** fields of this new partition.



Warning

Once the name is set, it cannot be changed.

9. Click the **Server Setting** tab and set the **Server Type** and **Naming context** for the AD.
10. By default, all groups and users are automatically included in a new partition. By using the **Group Locations** and **User Locations** tabs, you can decide which groups or users are included in this partition.
Set the groups or users to include in the new partition.
11. Your new OTDS partition has been created. Click the **Save** button to save the partition after the configuration settings have been made.



Note: Any tabs that are not explicitly mentioned here can be left with their default values intact. If an update to these default values must be made, refer to the OTDS team for detailed configuration instructions.

The users and groups that are imported from the AD can be found by clicking **Actions** then selecting **View Members**.

For more information on how to create a synchronized user partition, see the *OpenText Directory Services Installation and Administration Guide*.

5.3.1.4 Resource

Resources can be any ECM server, such as Documentum CM Server, in this case. They contain configuration information about the server URL, credentials, OTDS to docbase attributes mapping of dmotsrest service in Documentum CM Server.



Note: The dmotsrest service is deployed using the war file found here:
<Documentum CM Server root folder>\wildfly<version number>\server\ DctmServer_MethodServer\deployments\ServerApps.ear\dmotsrest.war.

Follow these steps to add a new resource:

1. Go to the **Resources** screen and click the **Add** button to add the dmotsrest.war file as a new resource.
2. Enter a value for **Resource Name** for this new resource.

 **Warning**

Once the value for **Resource Name** has been set, it cannot be changed.

3. Click on the **Synchronization** tab to go to the **Synchronization** screen.
4. In the **Synchronization** screen, select the **User and group synchronization** check box. This enables a number of other check boxes.
5. In the **Synchronization** screen, select **Delete users and groups** from the check box.
6. While still in the **Synchronization** screen, in the **Synchronization connector** box, select **REST (Generic)** from the list.
7. Click the **Connection Information** tab. Enter a value for each of the **Base URL**, **Username**, and **Password** fields of the `dmotdsrest` service and its Administrator.
8. Click the **Test Connection** button and make sure that **Connected Successfully** is displayed in the status area above the button.
9. *Optional Step:* Click the **User Attribute Mappings** tab to view the OTDS to AD attributes.

These attributes can be modified according to your specific needs or usage requirements. `oTExternalD3` is used as the value for `user_login_name` and `cn` is used as the value for `user_name`. For example, `otadmin@otds.admin` is used as the value for `user_login_name` and `otadmin` is used as the value for `user_name` in Documentum CM Server.

10. *Optional Step:* The values of the resource attributes can be set for the attributes shown on the **Group Attribute Mappings** tab.
11. Click **Save** to save this resource after you have completed its configuration.
12. An access role, which is named after the *Resource*, is automatically created in **Access Roles**.

For more information on how to create a synchronized resource, see the *OpenText Directory Services Installation and Administration Guide*.

5.3.1.5 Access role

Once a resource is created, an **Access Role**, with the same name as the resource, is automatically created in **Access Roles**.

Follow these steps to configure the access role:

1. Click **Actions** and select **Include Groups** to enable a group synchronization.
2. Click **Actions** and select **View Access Role Details** to go to the details configuration screen.
3. Click **Add** to add a partition to this access role. Select the box for the **67-ContentServer164** partition and click **Add Selected Items to Access Role**.
4. Click the **Users** tab and make sure that `otadmin@otds.admin` is not selected to avoid synchronization.

If **otadmin@otds.admin** is present, then select it and click **Delete**.

5. Click **Save** to save the **Access Roles**.

For more information on access role, see the *OpenText Directory Services Installation and Administration Guide*.

5.3.1.6 Sync users and groups

There are two ways to synchronize Users and Groups:

- Using a full synchronization.
- Using an Incremental synchronization.

5.3.1.6.1 Full sync

1. Select **Resources** from the left hand navigation.
2. Select the resource that was created.
3. Click **Actions** and select **Consolidate**.

5.3.1.6.2 Incremental sync

1. Select **Partitions** from the left hand navigation.
2. Select the partition that was created.
3. Click **Actions** and select **Restart**.

5.3.1.7 Configure Documentum CM Server

Perform the following steps to configure Documentum CM Server:

5.3.1.7.1 Enable OTDSAuthentication service

This section discusses how to set the OTDSAuthentication service in Documentum CM Server's configuration in order to enable authentication with OTDS.

1. At the command prompt, type **iapi**.
2. Enter the docubase (this is the repository name).
3. Enter the Administrator account username.
4. Enter the Administrator account password.
5. Enter the following command:

```
API> retrieve,c,dm_server_config  
API> append,c,1,app_server_name  
SET> OTDSAuthentication  
  
API> append,c,1,app_server_uri  
SET> http://localhost:9080/OTDSAuthentication/servlet/authenticate
```

```
API> save,c,1
```

6. If you enter `retrieve,c,dm_server_config` again and then enter `dump,c,1`, the following section should be displayed:

```
server_os_codepage : ISO_8859-1
app_server_name : do_method
[1]: do_mail
[2]: OTDSAuthentication
[3]: http://localhost:7080/DmMethods/servlet/DoMe
[4]: http://localhost:9080/DmMail/servlet/DoMail
[5]: http://localhost:9080/OTDSAuthentication/serv
thoa
vlet/authenticate
```

7. If the preceding information is displayed, then the configuration of Documentum CM Server is finished.

Restart the **docbroker** and **repositories** so that the settings will take effect.

5.3.1.7.2 Configure OTDSAuthentication service

5.3.1.7.2.1 Get a Certificate from OTDS

1. Go to `http://<otds-server>:8080/otdsws/rest/systemconfig/certificate_content` to get the certificate content of OTDS.
2. Copy all of the certificate information between the `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` markers.
3. On the Documentum CM Server machine, find the `otdsauth.properties` file, which is typically found at `<OpenText Documentum CM Root Folder>/wildfly<Version>/server/DctmServer_MethodServer/deployments/ServerApps.ear/OTDSAuthentication.war/Web-INF/classes`.
4. Open the `otdsauth.properties` file and paste the certificate information after `certificate=`.



Note: Erase all spaces and line returns within the certificate content that you copied to make sure that it is a continuous String.

5. Set `http://<otds-server>:8080/otdsws/rest/authentication/credentials` after `otds_rest_credential_url=`.
6. Save this file and restart the JMS.



Note: Using IAPI, make certain that Documentum CM Server and OTDS are completely configured and integrated before using Foundation REST API.

- To make sure that token mode is ready to be worked with, use the following command when `otds_token`, `ct-otds_token` is used as the authentication mode in runtime properties:

```
API> connect,<REPO_NAME>,null,dm_otds_ticket=<ACCESS_TOKEN_FROM_OTDS>
```

- To make sure that password mode is ready to be worked with, use the following command when `otds_password`, `ct-otds_password` is used as the authentication mode in runtime properties.

```
API> connect,<REPO_NAME>,<USER_LOGIN_NAME>,dm_otds_password=<PASSWORD_ON_OTDS>
```

If there is something wrong, see the logs at <Documentum CM Server root folder>\wildfly<version number>\server\DtcmServer_MethodServer\logs for more information and refer to Documentum CM Server team for help.

Make sure the proper command, according to your authentication mode, can display information without any exceptions before you continue.

For more information, including detailed instructions, see *OpenText Documentum Content Management - Server Administration and Configuration Guide (EDCCS250400-AGD)*.

5.3.1.8 Configuration for REST

5.3.1.8.1 Configure OAuth clients on OTDS

1. Enter the following URL to configure OAuth clients on OTDS and log in with the default administrator account, **otadmin@otds.admin**.

`https://<otds_ip>:<otds_port>/otds-admin`
2. On OTDS, navigate to OAuth clients and click **Add**.
3. Enter the *Client ID* and ensure that the **confidential** checkbox is not selected so that the Foundation REST API client is able to use implicit flow of OAuth 2.0.
4. Select **Redirect URLs**, add the REST URL, and save your changes.

5.3.1.9 Configure runtime properties

These authentication modes are supported in REST:

- **otds_token**: Supports OTDS integration with Documentum CM Server using token-based authentication.
- **ct-otds_token**: Supports OTDS integration with Documentum CM Server using token-based authentication with a client token cookie.
- **otds_password**: Supports OTDS integration with Documentum CM Server using password-based authentication.
- **ct-otds_password**: Supports OTDS integration with Documentum CM Server using password-based authentication with a client token cookie.
- **otds_token-basic**: Supports both HTTP basic authentication and **otds_token** schemes.
- **ct-otds_token-basic**: Supports both HTTP basic authentication and **otds_token** schemes where a client token cookie is supported for both.

➡ Example 5-2: Configuring REST api runtime properties

In this example we use ct-otds_token authentication, which is the most commonly used authentication mode, to show you how to configure the rest-api-runtime.properties file:

1. Go to <web application root folder>\webapps\dctm-rest\WEB-INF\classes and find the rest-api-runtime.properties file.
2. Add these lines to the rest-api-runtime.properties file:

```
rest.security.auth.mode=ct-otds_token
rest.security.realm.name=com.documentum.rest
rest.security.otds.login.url=http://<otds-server>:8080/otdsws/login
```



5.3.1.10 Authentication sample

The following sample uses a REST tool called **Postman** to perform the following:

5.3.1.10.1 Get an access token from OTDS

Using the implicit flow of **OAuth 2.0**, get an access token from OTDS. This sample shows you how to get that access token.

1. Send the following Request to retrieve the access token from OTDS:

```
POST http://otds-server:8080/otdsws/login HTTP/1.1
Content-Type: application/x-www-form-urlencoded; charset=utf-8

grant_type=password&client_id=<OAuth Client name in OTDS>&
username=<User Login Name>&password=<User Password>
```

2. The access token is then returned in the Response body.

➡ Example 5-3: A Response body

```
{
  "access_token": "eyJraWQiOiiwM2ZiNzE40DcwNDk2NjY5MzY0MjI2NzU5MWNjZDNjZDAwZ
DBjN2UxIiwiIjoiSlldUIwiYVxnIjoiU1MyNTYifQ.eyJzdWIoiI
ZZWY0YTc3Mi1MTdkLTQ2NjUtYmJmNS04M2EyZTjkNjVmYjIiLCJzY3zY
01tdLCjkbaXai0nsiT1REU19IQVnfUEFTU1dPUkQiOiJmYWxZSj9LCJzY
XQi0jE1NTE20DgyMDIsIm1zcyI6Imh0dHA6Ly8xMC4yNDUuMzkuMjMy0j
gwODAiLCjnQioiJQQVNT09SRCIsInR5cI6ImFjY2Vzc190b2tbiI
sInBpZCI6IjEwljIjONS40MC40MSIsInjPZC16eyJ1NjdzKG15ZS0xOTQ1
LTQ0MjUtOTNhOS1hOGYyODE2YjBjMjgiOjvdGRzLXR1c3QiLCIxNWI40
TcyOC1jZWQ3LTQzYjYtYc1zs05YzcmMm1zODM2YjEi0jvdGRzLXR1c3
QifSwic2lkIjoiMzVknDEzYjYtMDU3S00ZTVkLWE40GQtMjhNTU5M2I
1NjbjIiwiDWlkIjoiB3RkcyI0ZXN0QGRjdG0ucmVzdC5jb20iLCU1bm0i
0iJvdGRzLXR1c3QiLCJuYw1IjoiB3RkcyI0ZXNOIiwiZXhwIjoxNTUxN
jkxODAyLCjpxXQ10jE1NTE20DgyMDIsImpoASt6IjU4Y2E5NzA2LWRjNz
UtNGEyOC04MGRjLTc3MzY1MTNKZDE4ZiIsImNpZC16IkNvbnR1bnRTZxJ
2ZXItUmVzdCj9_LPqrg2tHXzZ92g-Ue1xxuk87ANB1x01FXM7rhuYtDQR
s-SCRZhVVdSvyzbXMoF3RJbSHJmBofcxjvhfozOKdMUHNqltLH44YaX
ZBdYMHFe6we1dsefSyDTYxzI1zrELz0oXAJ13GHMp5Z8NPzderVm5mI3
5LEpOTWK6o-2FLMM_S11LMQ-QGfrbJXjtAsg1dyAHC3K7H9ItEx_efag
zojYKEzYVIjgcuuf3RaQRHnViP_GgWXaLoeZQ_YazP6GEvKKH5iyXU7H
Lj8mL-hHjxs5jUw-3NI34NdfW7I8uLAcfi6Fc16ZtJyjCx2vIqQIey4Jg
ywFLz19vdq7Mw",
  "refresh_token": "eyJraWQiOiiwM2ZiNzE40DcwNDk2NjY5MzY0MjI2NzU5MWNjZDNjZDAwZ"
```

```

DBjN2UxIiwidHlwIjoisIdUiawiYWxnIjoiU1MyNTYifQ.eyJzdWIiOiI
zZwYOYTc3Mi1lMTdkLTQ2NjUtYmJmNS04M2EyZTJkNjVmYjIiLCJzY3Ai
0ltdLCJkbXaiOnsiT1REU191QVNfUEFTU1dPUKQiOijmYwxZS9LCJzY
XQiOjE1NTe20DgyMDIsImlzcyI6Imh0dHA6Ly8xMC4yNDUuMzkuMjMyOj
gwODAilCJncnQioiJQQVNTV09SRCIsInR5cCI6InJ1ZnJ1c2hfdG9rZW4
iLCJwaWQiOixMc4yNDUuNDauNDEiLCJzaWQiOizNWQOMTNiN10wNTd1
LTr1NWQtYTg4ZC0yOWE1NTkYZju2MGMiLCJ1aWQiOjJvdGRzLXRlc3RAZ
GNobS5yZXN0LmNbVSIsInVubSI6Im90ZHtdGVzdCIsIm5hbWUiOjJvdG
RzLXR1c3QiLCJ1eHAiOm51bGwsImhdCI6MTU1MTY40DIwMiwanRpIjo
iMWQNTc2YjQtODk0MCO0MDUzLTK3ZmQtODE5MmYwZyY1MjQ4IiwiY21k
IjoQ29udGVudFNlcnZlci1SZXNOIno.U25sXPRZ8-yddkq-a9A82Etzw
djkx-o69XM5Kmb6qwhExe03RpxJojtASif08W1W3V1uPrBAoyYgZS8SHm
aKOiQS_SXG5_iq9ASX0wQSvfbKINoAVKL7GnfOuYWffJp3crozEjeTxuS
D4g6vr94nGwNOHLFLnzp_UC_1zVCH1Is4Sxsg1X-8Kc5DakbRoXVAst7iR
5bDtImK4K1BQ712N8bfNsNDPvxs1FvjLJ0Z07B2GhLckQ-vCZ2r3vpTqHj
TFo9LvKTItCTheXECf50Cw3GIv3-HRyjIMELXHqr3mbbBoj2ZhVzdXV2P
hLaUemnf0Z1hIqMD4Ujfjh99nkY5AQmw",
    "token_type": "Bearer",
    "expires_in": 3600
}

```



5.3.1.10.2 REST Request with an access token

Follow these steps to create and send a REST Request that contains an access token:

Copy an existing access token from a former response and use it in *one* of the two ways shown here:

- Add the prefix *Bearer* to the access token and paste it into the Header Authorization property of your HTTP request.
- Using the GET HTTP method, paste the access token after the URL to use it as a query parameter value.

```
GET http://<otds-server>:<otds-port>/dctm-rest/repositories/
REPO?access_token=<paste_access_token_value_here>
```

The result is that access to the requested URL is given to the Foundation REST API client.

5.3.2 Authentication workflow

The following diagram illustrates the OAuth 2.0 authentication process for OpenText Documentum CM applications. The steps in the authentication process are as follows:

- The *User Agent* sends the request to the OAuth2 server to obtain an access token.
- The *OAuth2 server* forwards the request for user authentication and, if authenticated, sends back an access token.
- The *User Agent* sends a REST request to the Foundation REST API server along with the access token.
- The *Foundation REST API server* validates the access token, authenticates the user with Documentum CM Server, and then returns a DOCUMENTUM-CLIENT-TOKEN cookie.

The Foundation REST API server does not care how the User Agent goes about getting the OAuth access token from the OAuth2 server. The Foundation REST API server starts to handle the OAuth 2.0 authentication by examining the access token from the Request that it receives from the User Agent.

OAuth authentication workflow

1. The User Agent makes a request to the Foundation REST API server for a protected resource.
2. The Foundation REST API server responds with a 401 Unauthorized if the OAuth access token or client token is not found within the Request.

```
HTTP/1.1 401 Unauthorized
Content-Type: application/json; charset=UTF-8
WWW-Authenticate: Bearer acme.com
Location: https://oauth-server.acme.com/login
```

The WWW-Authenticate Response Header indicates that the *Bearer* authentication schema should be used to issue further Requests. The Location Response Header points to the OAuth Server login URL where it can retrieve the OAuth access tokens. To customize the Response Headers, set the following configuration parameters in the Foundation REST API server.

Add to WEB-INF\classes\rest-api-runtime.properties

```
rest.security.realm.name=acme.com
rest.security.oauth2.login.url=https://oauth-server.acme.com/login
```

3. The User Agent attempts to make an OAuth authentication. The user client application should handle the URL redirection correctly.
4. The OAuth2 server requests that the User Agent provide it with credentials.
5. The User Agent provides the requested credentials to the OAuth2 server. The preceding two sequences (marked in red) indicate that the Request and the Response are out of the scope of the OAuth2 server.
6. The OAuth2 server tries to verify the credentials by communicating with the backend identity provider.
7. The OAuth2 server Requests for authorization now grant scope. The scope can limit the authorization range of the User Agent.
8. The User Agent grants scopes for the protected resource on behalf of the resource owner and asks for the authorization grant.

➡ Example 5-4: Request sent by User Agent

The full Request sent by User Agent is shown here:

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded
```

```
grant_type=authorization_code&code=Spxl0BeZQQYbYS6WxSbIA
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcbs
```

In this example, the OAuth2 server is assumed to accept *Basic* login. So *Basic* information is sent in the Authorization Request Header. The parameter '*grant_type*' is set to '*authorization_code*', which indicates that the Client wants to do the authorization grant in the Authorization Code type. The following '*code*' parameter specifies the authorization code to retrieve from the Auth Server. The '*redirect_uri*' is telling the Auth Server where to redirect the client after granting authorization.



9. The OAuth2 server grants the access token to the User Agent.

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Cache-Control: no-store
Pragma: no-cache
{
    "access_token": "2YotnFZFEjr1zCsicMWpAA",
    "token_type": "example",
    "expires_in": 3600,
    "refresh_token": "tGzv3J0kF0XG5Qx2T1KWIA",
    "example_parameter": "example_value"
}
```

10. The User Agent requests the protected resource from the Foundation REST API server.

```
GET localhost:8080/repositories/REPO HTTP/1.1
Content-Type: application/json; charset=UTF-8
Authorization: Bearer 2YotnFZFEjr1zCsicMWpAA
```

11. The Foundation REST API server sends the access token to Documentum CM Server.
12. Documentum CM Server validates the access token and if the validation is successful, then Documentum CM Server provides a session to Foundation REST API server.
13. The Foundation REST API server gets the Foundation Java API session.
14. The Foundation REST API server confirms login success and saves the security context.
15. The Foundation REST API server represents the protected resource for the User Agent. When REST is configured to support a client token, then REST creates a new DOCUMENTUM-CLIENT-TOKEN and sends it to the User Agent. Then the User Agent can directly use this client token to request other REST resources without going through the OAuth 2.0 authorization process again.

5.3.3 Configuration on the Foundation REST API server

You must configure Foundation REST API server to perform OAuth 2.0 authentication on Foundation REST API.

Edit `rest-api-runtime.properties` in the `dctm-rest.war` as follows:

```
# oauth2
#   - Support OAuth 2.0 and configurable support OpenId Connect.
# ct-oauth2
#   - Support OAuth 2.0 and configurable support OpenId Connect with Client
#   Token cookie supported.
# The default mode is OTDS. For more information, refer to the
#   Reference Guide for details.
#
rest.security.auth.mode=

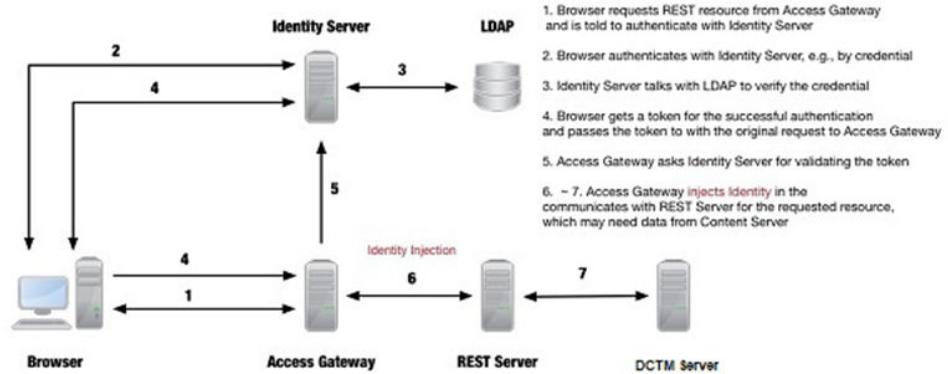
# The url of login entry point for OAuth 2.0 to indicate user where to send request.
# This value is used as a hint when oauth2 error occurs.
# The default value is empty.
rest.security.oauth2.login.url=
```

5.4 Pre-authenticated authentication

You may have deployed your own claim-based authentication infrastructure prior to deploying Foundation REST API. The customer authentication scheme is not supported by REST or Documentum CM Server out of the box. The customer authentication scheme performs the authentication ahead of Foundation REST API and returns an HTTP Header or cookie along with the HTTP Request to the REST servlet. The HTTP Header or cookie serves as the plain-text principal (without password) authorization of the authenticated user. The pre-authenticated authentication scheme can be used to integrate Foundation REST API with Tivoli WebSEAL, NetIQ Access Manager or other similar Web Access Management (WAM) systems.

This security requirement can be fulfilled using a WAM solution, which protects networks, applications, and data while simultaneously providing quick and efficient access to an increasing number of authorized users.

The following is the typical topology of a WAM architecture, which describes a typical scenario:



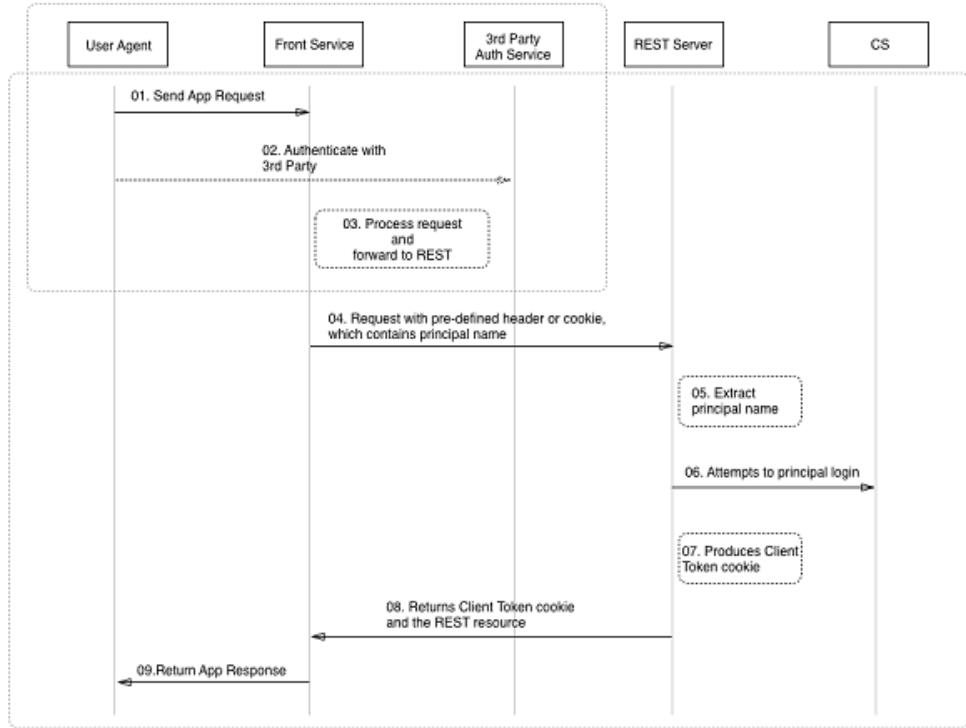
Foundation REST API can provide you with a pre-authenticated authentication scheme, where customers only need to specify the Header name and the regular-expression to parse the Header value. Foundation REST API uses Privileged DFC (principal mode) or *trust.properties* to get Documentum CM Server sessions for the user operation.

 **Note:** Details of the Privileged DFC instance and configuration of *trust.properties* can be found in the *OpenText Documentum Content Management - Server Fundamentals Guide (EDCCS250400-GGD)*.

5.4.1 Workflow

The following is a diagram that shows the workflow of a pre-authenticated login. The following are constraints for this authentication extension:

1. The User Agent cannot communicate with the Foundation REST API server. Instead it only communicates with the Front Service.
2. The Front Service authenticates the User Agent using a 3rd Party Authentication Service. The Front Service does not authenticate the User Agent with the Foundation REST API server or the Documentum CM Server.
3. The Front Service communicates with the Foundation REST API server according to a predefined schema, such as http Headers or a cookie.
4. The Foundation REST API server extracts the principal name based on a predefined pattern, and uses the principal name to attempt a principal login to Documentum CM Server.



5.4.2 Feature Highlights

Runtime Configuration for Pre-authenticated Login

```

#####
##      6. General Security Configuration      ##
#####
# .....
# .....

# preauth
#   - Support pre-authentication without Client Token cookie supported
# ct-preauth
#   - Support pre-authentication with Client Token cookie supported
# The default mode is basic.
# For more information, see OpenText Documentum REST Services Reference Guide.
#
rest.security.auth.mode=

#####
##          Security Configuration for Pre-authenticated Login           ##
#####
# This section contains the security configuration for Pre-authenticated Login to parse
# the principal from the Request by the front service. Configuration in this section
# work
# only when preauth scheme is set in the General Security Configuration section.
# More information is found in this guide, including details on the configuration steps.
#
#
# Specifies the names of headers which contain the principal name sent by the front
# service.
# The value must be a set of comma-separated header names without spaces. The value is

```

```

case
# sensitive. The REST server iterates through the headers of the Request and retrieves
the
# first one that is contained in the set of cookie names defined here. For example,
assuming
# [FrontServiceToken1,FrontServiceToken2] is specified here. The HTTP Request has header
# "FrontServiceToken1=principal_name". Therefore "principal_name" is extracted to attempt
# the DFC principal login. When the value is not specified, the REST server won't
extract the
# principal name from the headers. With regards to extracting the principal name, the
priority
# of headers is higher than the priority of settings within cookies.

rest.security.prauth.header.names=
# Specifies the names of cookies which contain the principal name sent by the front
service.
# The value must be comma-separated cookie names without spaces. The value is case
sensitive.
# The REST server iterates through the cookies of the Request and retrieves the first one
# that is contained in the set of cookie names defined here. For example,
# [FrontServiceToken1,FrontServiceToken2] is specified here. HTTP request has cookie
# "FrontServiceToken2=principal_name". The "principal_name" is extracted to attempt DFC
# principal login. When the value is not specified, the REST server won't extract
principal
# name from cookie. With regards to extracting the principal name, the priority
# of headers is higher than the priority of cookies.
#
rest.security.prauth.cookie.names=
# Specified the regular expression patterns to extract principal name from the
distinguished
# name by the front service. The value must be dual-pound (##) separated regular
expressions
# without spaces. The value is case sensitive.
# For example, "uid=(.*?),ou=ecd,dc=emc,dc=com##(.*)" is specified here. If distinguished
# name by the front service is "uid=joe,ou=ecd,dc=emc,dc=com", first regular expression
will
# work and "joe" is parsed out. When the distinguished name by the front service is
"joe", the
# second regular expression works and "joe" is extracted.
# By default, the regular expression is "(.*)" to match the whole string.
#
rest.security.prauth.principal.patterns=

```

5.4.3 Set Documentum Client Token cookie or not

The DOCUMENTUM-CLIENT-TOKEN is one of the authentication schemes that are supported by Foundation REST API. When the DOCUMENTUM-CLIENT-TOKEN cookie is set, the Foundation REST API server can serve the Front Service by using the Client Token authentication schema. When the DCTM Client Token is not sent back, the Foundation REST API server has to get the session by using the principal login for all subsequent Requests.

REST API Runtime Properties

```
#Client Token won't be sent back
rest.security.auth.mode=prauth
```

5.4.4 Principal in headers or cookie

The Front Service sends the distinguished name that contains the principal by Header or cookie. The Foundation REST API server processes the Header first, then it processes the cookie. The names are case sensitive.

5.4.5 Principal pattern

Foundation REST API applies the patterns specified to the distinguished name sent by the Front Service and extracts the principal. Here are the two patterns:

```
# Two patterns separated by double pound ## characters
rest.security.prauth.principal.patterns=uid=(.*?),
ou=ecd,dc=emc,dc=com##cn=(.*?),ou=pc,dc=dell,dc=com
```

5.5 Reverse proxy configuration

If the Foundation REST API server is placed behind a reverse proxy server, the following configurations are required on the proxy server. These configurations make the Foundation REST API server generate correct links in a response.

- The proxy server must retain the original the HOST Header when forwarding a request to the Foundation REST API server, instead of modifying it.
 - If you deploy the proxy server on an Apache server, set `ProxyPreserveHost` to on.
 - If you deploy the proxy server on an Nginx server, use the following directive:

```
proxy_set_header Host $host;
```
 - If you deploy the proxy server on an IIS server, you must use the URL Rewrite module to rewrite the URLs the Foundation REST API server generates to the original HOST value.

For more information about how to set this Header on other application servers, refer to the documentation of the application server.

- If the communication between the client and proxy server and that between the proxy server and Foundation REST API server uses different protocols (for example, the client uses HTTPS to communicate with the proxy server while the proxy server uses HTTP to communicate with the Foundation REST API server), unify the communication protocol by using the `X-Forwarded-Proto` Header on the proxy server.

On Nginx, use the following directive:

```
proxy_set_header X-Forwarded-Proto <protocol>
```

For example, the following directive forces the proxy server to use HTTPS to communicate with the Foundation REST API server:

```
proxy_set_header X-Forwarded-Proto https
```

On Apache, add the following line in the virtual host configuration:

```
RequestHeader set X-Forwarded-Proto <protocol>
```

For more information about how to set this Header on other application servers, refer to the documentation of the application server.

- In most cases, the port is included in the HOST Header. In this case, you do not need to specify the port on the proxy server. Otherwise, unify the port by using the X-Forwarded-Port Header on the proxy server if the proxy server and the Foundation REST API server use different ports.

On Nginx, use the following directive:

```
proxy_set_header X-Forwarded-Port <Port>
```

On Apache, add the following line in the virtual host configuration:

```
RequestHeader set X-Forwarded-Port <Port>
```

For more information about how to set this Header on other application servers, refer to the documentation of the application server.

- Performance may degrade when many clients send requests concurrently. To improve concurrent access performance, we recommend that you modify the multi-processing module of the proxy server according to the related documentation. For example, if you use an Apache-based proxy server on a Windows machine, you may need to increase the value of ThreadsPerChild and MaxRequestsPerChild, or even remove the limitation on the number of requests that an individual child server handles by setting MaxRequestsPerChild to 0.

5.5.1 Configuration Samples

The following sample shows how to configure an Nginx-based reverse proxy server to generate correct links:

Example 5-5: Reverse Proxy Configuration on Nginx to Generate Correct Links

```
location / {
    proxy_pass http://rest_servers;
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-Proto https;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Port 443;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_redirect off;
}
```

The following sample shows how to improve concurrent access performance for an Apache-based proxy server on a Windows machine:

Example 5-6: Reverse Proxy Configuration on Apache to improve concurrent access performance

```
<IfModule mpm_winnt_module>
ThreadsPerChild 512
```

```
MaxRequestsPerChild 0  
</IfModule>
```



5.6 Client token

A DOCUMENTUM-CLIENT-TOKEN is an encrypted proof token that represents the identity of an authenticated user who can access Foundation REST API. It is not a standalone authentication scheme, it is integrated with other authentication schemes and used together with those authentication schemes to authenticate REST requests. Typically, a Client Token is transferred between the Foundation REST API client and the Foundation REST API server as HTTP cookies. These HTTP cookies have the cookie name DOCUMENTUM-CLIENT-TOKEN.

When Client Token is enabled, the Foundation REST API server generates a Client Token cookie after a successful authentication, and sends it back to the Foundation REST API client in the Response. The purpose of this client token is to provide an authenticated Foundation REST API client with a temporary token, that expires, and can be used to access the Foundation REST API server without having to validate user credentials, or negotiate SSO tokens each and every time. The Client Token cookie has no session state stored on the Foundation REST API server. Therefore, it can work in clustered environments.

A client token cookie is encrypted and validated by the Foundation REST API server. The Foundation REST API client is not responsible for saving or decrypting the token. A validation failure of the client token cookie leads to an authentication failure. In this case, the Basic token negotiation must be repeated.

The Client Token can be used with the following existing authentication schemes:

- HTTP Basic authentication
- Pre-authentication authentication

For more information on how to develop your own authentication scheme and integrate it with the Client Token, see “[Authentication extensibility](#)” on page 313.



Note: On Chrome 80 and later, the default behavior for sending cookies in the first- and third-party contexts, has changed as follows:

- Cookies that do not specify a SameSite attribute are treated as if they specified SameSite=Lax. They are restricted to first-party or same-site contexts by default.
- Cookies that are intended for third-party or cross-site contexts must specify as SameSite=None;Secure for secure or HTTPS connections.

For the HTTP connection, specify as SameSite=None.

- Adding SameSite=None allows your cookie to be sent in third-party contexts where the partitioned attribute is not supported, as long as third-party

cookies are allowed in browser settings. If the third-party cookies are not allowed in browser settings, adding `SameSite=None` does not allow your cookie to be shared in third-party contexts or applications. To allow cookies to be exchanged between the third-party contexts or applications where a top level site and an embedded iframe third-party application inside the top level, then you must set the `None; Secure; Partition` as a value to the `rest-api-runtime` property, `rest.security.client.token.cookie.samesite`.

Since the Foundation REST API server sets the DOCUMENTUM-CLIENT-TOKEN cookie, it must be configurable for the users to determine if they want to enable the first-party or third-party context to submit cookies.

Add or update the following property in the `rest-api-runtime.properties` file to set the `SameSite` attribute in the DOCUMENTUM-CLIENT-TOKEN cookie. It is used to control access to the DOCUMENTUM-CLIENT-TOKEN cookie from third-party sites.

The possible values are Strict, Lax, and None. The default value is blank, which means that the browser will determine the third-party access behavior. For example:

```
rest.security.client.token.cookie.samesite=
```

Enabling Client Tokens

To see if a Client Token can be enabled in an authentication scheme, refer to your chosen authentication scheme, or to see `<dctm-rest.war>/WEB-INF/classes/rest-api-runtime.properties.template`.

Foundation REST API supports the following expiration policies for the client token cookie:

Policy	Description
com.emc.documentum.rest.security.ticket.impl.HardTimeoutExpirationPolicy	<p>The client token expires after a specified duration.</p> <p>If the Foundation REST API client sends a request before the duration, the Foundation REST API server accepts the client token. If the Foundation REST API client sends a request after the duration, the Foundation REST API server rejects the client token, and the client has to authenticate again.</p>

Policy	Description
com.emc.documentum.rest.security.ticket.impl.TolerantTimeoutExpirationPolicy (default)	<p>The client token expires after two times of the specified duration. The Foundation REST API server issues new client tokens under certain conditions. For details, see the following:</p> <ul style="list-style-type: none"> • If the Foundation REST API client sends a request before the duration, the Foundation REST API server accepts the client token. • If the Foundation REST API client sends a request after the duration and before two times of the duration, the Foundation REST API server accepts the client token and issues another client token with the same duration to the client for subsequent requests. • If the Foundation REST API client sends a request after two times of the duration comes to an end, the Foundation REST API server rejects the client token, and the client has to authenticate again.
com.emc.documentum.rest.security.ticket.impl.TouchedTimeoutExpirationPolicy	<p>The client token expires after a specified duration. The Foundation REST API server issues new client tokens under certain conditions. For details, see the following:</p> <ul style="list-style-type: none"> • If the Foundation REST API client sends a request before the duration, the Foundation REST API server accepts the client token, and issues another client token with the same duration to the client for subsequent requests. • If the Foundation REST API client sends a request after the duration, the Foundation REST API server rejects the client token, and the client has to authenticate again.

5.6.1 Startup Script Modification

For Linux operating systems, if client tokens are used in your deployment, add the following option to the startup script of the application server where Foundation REST API is deployed to achieve better performance:

```
JAVA_OPTS="$JAVA_OPTS -Djava.security.egd=file:/dev/.urandom"
```

5.6.2 Explicit Logoff

Starting from release 7.2, all authentication schemes utilizing client tokens enable client applications to explicitly log off. When a client application obtains a client token, the link relation `http://identifiers.emc.com/linkrel/logoff` is generated in the Repository resource. The client application can log off by using this link relation, which notifies the Foundation REST API server to invalidate the client token. After that, the client application must perform authentication again when submitting new requests.

5.6.3 Logoff Success Handling

By default, the sign-out process returns a 204 Response without a response body. In a real world case, the sign out usually must redirect the user to another page. In 7.3, another runtime property is exposed for users to configure the logout success URL. The redirection happens only when the logout succeeds. The following runtime property allows to customize logout success URL.

```
# Specifies the URL to redirect to after the successful logout.
# This property can be used when a Client Token cookie is used# for authentication.
# It can be a complete URL or a relative URL. When it is a relative URL, the server
# inserts the Request base and context path to the URL path.
# When it is not specified, the server returns no content for a successful logout.
# Examples:
#   - http://another-server/goodbye
#   - /services
rest.security.logout.success.url=
```

By convention, only the POST HTTP method is typically supported for signing out of most web applications. Allowing for the integration with other authentication schemes, such as OTDS, by using GET to communicate with Foundation REST API for the process of logout. Another runtime property is used to specify which HTTP methods are supported for sing out. When the Request method is not supported, the status code 405 is returned with an error message in the Response. The following runtime property allows you to customize logout HTTP method.

```
#Specifies supported HTTP methods for sign out. Multiple values separated by comma,
# are supported as below.
#Examples:
#rest.security.logout.supported.methods=GET,POST
#Note: the HTTP methods specified in String format are not case sensitive,
# meaning 'get,POST', 'GET,post' or 'Get,Post' all make sense.
#when not specified, the default values GET,POST would be applied.
rest.security.logout.supported.methods=
```

5.6.4 Client Token Encryption and Decryption

Client tokens can be encrypted and decrypted using various cryptography algorithms. Follow the instructions in `rest-api-runtime.properties` to configure the cryptography algorithm-related parameters.



Notes

- The default cryptographic algorithms that are used for client token encryption and decryption are strong enough in most cases. Therefore, we recommend that you keep the original settings unless you have special business requirements.
- When you use a Crypto provider other than **JsafeJCE** (RSA provider) or **BC** (Bouncy Castle provider), you must set the `rest.security.crypto.provider.class` parameter after modifying the `rest.security.crypto.provider`.
- When you deploy Foundation REST API on IBM WebSphere and use the HTTP Basic with Client Token combination of authentication schemes.

The following configurations are required in the `rest-api-runtime.properties` file:

- `rest.security.crypto.provider=IBMJCE`
- `rest.security.crypto.provider.class=com.ibm.crypto.provider.IBMJCE`
- `rest.security.random.algorithm=IBMSecureRandom`
- For a multi-node deployment of Foundation REST API servers, you must set the `rest.security.crypto.key.salt` parameter consistently across all Foundation REST API servers.
- Some web applications may contain security providers that share the same names with the security providers in Foundation REST API. The `rest.security.crypto.provider.force.replace` property determines whether to replace a security provider in web applications with the one in Foundation REST API when the two providers share the same name. The default value is `false`, meaning that Foundation REST API uses the security provider registered in the web application. The default setting is recommended.
- Foundation REST API client token encryption is supported by the **Java 7** cryptographic cipher.
- Foundation REST API also supports third-party encryption providers such as Java Sun service provider module and **Bouncy Castle**.

5.6.4.1 Algorithms with a Key Size Larger than 128 bits

The default version of Java Cryptography Extension (JCE) policy files bundled in the JDK environment limits the key size of cryptography algorithms to 128 bits. To remove this restriction, download Unlimited Strength Jurisdiction Policy Files from the Oracle web site.

5.7 Auditing the Foundation REST API client events

The Foundation REST API provides a `rest.security.audit.events` property in the `rest-api-runtime.properties` file as a comma separated string of values to audit the REST client application events, the Foundation REST API supports the authentication events such as login (`dm_login`) and logout (`dm_logout`).

The Foundation REST API client application must:

- Set the `rest.security.audit.events` property to `auth` to enable auditing for authentication events.
- Use client token-based authentication to audit the login and logout events. The initial request that generates the `DOCUMENTUM-CLIENT-TOKEN` cookie is considered as a login event. A request that clears this cookie is considered as a logout event.

To log additional critical information along with the existing auth events, include that information as part of the HTTP headers. Then, add these headers to the `rest.security.audit.headers` property as a comma separated values. This additional HTTP headers information is added to the `attribute_list` of `dm_audittrail` table.

For example: `header1=value1;header2=value2;....`

The following are OpenText Documentum CM audit event records:

dm_audittrail field	dm_audittrail value	sample values	Description
<code>event_name</code>	<code>dm_login</code> or <code>dm_logout</code>	<code>dm_login</code> or <code>dm_logout</code>	A specific value for <code>dm_login</code> or <code>dm_logout</code> event.

dm_audittrail field	dm_audittrail value	sample values	Description
string _1	<product name>	dctm-rest	The Foundation REST API identifies the client application name using the following order (from highest to lowest): <ol style="list-style-type: none"> <i>Request Attribute:</i> If the request contains the X-CLIENT-APPLICATION-NAME attribute, its value is used. <i>Request Header:</i> If the attribute is not present, the system checks for the header X-CLIENT-APPLICATION-NAME in the HTTP request. <i>Context Path:</i> If neither the attribute nor the header is available, the context path, which is the WAR file name is used as the fallback identifier.
string _2	Not applicable	Not applicable	Empty.
string _3	<remote IP address>	10.193.70.198	The Foundation REST API resolves the remote IP address using the following order (from highest to lowest): <ol style="list-style-type: none"> <i>X-Forwarded-For Header:</i> If present, the value of the X-Forwarded-For header is used. <i>Tomcat Server IP Address:</i> If <code>request.getRemoteAddr()</code> returns 127.0.0.1 or ::1. This indicates a loopback address, the IP address of the running Tomcat server is used as a fallback. <i>request.getRemoteAddr():</i> If none of the above conditions are met, the IP address returned by <code>request.getRemoteAddr()</code> is used.
string _4	<remote host name>	abcd.companynamen.net	The Foundation REST API attempts to resolve the remote hostname of the incoming HTTP request using the following logic: <ul style="list-style-type: none"> In <code>server.xml</code>, if the Tomcat connector is configured with <code>enableLookups="true"</code> the hostname is resolved using reverse DNS lookup. If <code>enableLookups</code> is not enabled, Tomcat defaults to returning the IP address using <code>request.getRemoteAddr()</code>. This is part of Tomcat's internal implementation.
string _5	<user geo location coordinate>		If present, the value of the X-CLIENT-LOCATION header is used to identify the geographical or logical location of the client application. For example: <code>request.getHeader("X-CLIENT-LOCATION")</code>

dm_audittrail field	dm_audittrail value	sample values	Description
attribute_list	<request_header1=value1:request_header2=value2>	X-NETWORK-LOCATION = Bangalore eBOCS	<p>Foundation REST API supports capturing specific request headers for auditing purposes. These headers are appended to the attribute_list field in the audit log entry.</p> <p>The headers to be captured must be explicitly listed in the property: rest.security.audit.headers.extra.</p> <p>For each incoming request, Foundation REST API checks for the following:</p> <ul style="list-style-type: none"> If values are appended to the attribute_list field, separated by semicolons (;). If none of the specified headers are present in the request, no values are added to attribute_list.

5.8 Bypassing browser login forms upon HTTP 401 unauthorized

When a user provides invalid credentials, an HTTP 401Unauthorized status code is returned, which contains a *WWW-Authenticate* Header indicating the authentication scheme.

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic <My Realm>
```

This behavior may trigger some web browsers to display a login dialog box prompt for re-authentication. However, if you design your own login screen, it could be blocked by the one the web browser prompts. To bypass a login dialog box, the Foundation REST API server can append a suffix to the authentication scheme in the *WWW-Authenticate* Header. This makes web browsers not recognize the scheme and so that no login dialog boxes are prompted.

To enable this feature, client applications must set the custom Header DOCUMENTUM-CUSTOM-UNAUTH-SCHEME to true in requests. Optionally, on the Foundation REST API server, you can customize the *WWW-Authenticate* Response Header for HTTP status 401 with the *rest.api.unauthorized.Response.scheme.suffix* in the *rest-api-runtime.properties* runtime property setting.

➤ Example 5-7: Request

```
GET /${resource-url} HTTP/1.1
Authorization: Basic ZG1hZG1pbjpwYXNzd29yZ
DOCUMENTUM-CUSTOM-UNAUTH-SCHEME: true
```



► **Example 5-8: Runtime Property Setting**

```
rest.api.unauthorized.Response.scheme.suffix=MyScheme
```



► **Example 5-9: Response**

```
HTTP/1.1 401 Unauthorized  
WWW-Authenticate: Basic_MyScheme <My Realm>
```



Chapter 6

Using Kubernetes

OpenText Documentum Content Management - Cloud Deployment Guide (EDCSYCD250400-IGD) provides more details about using Foundation REST API in a Kubernetes environment.

Chapter 7

Resource extensibility

7.1 Overview

Resource extensibility is an API infrastructure that enables you to extend Foundation REST API by composing, customizing, and creating new REST resources. These newly-created resources are finally discoverable from the Home Document, existing core REST resources, or new custom resources by using HATEOS relations. By wielding the power of resource extensibility, you can tailor Foundation REST API to your needs with limited amount of coding.

The following diagram illustrates the lifecycle of custom resource development:

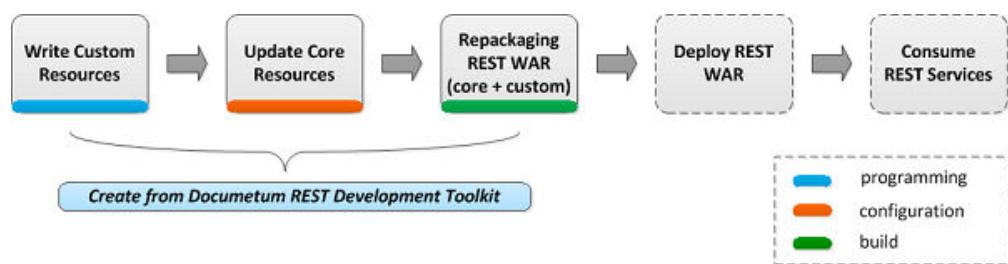


Figure 7-1: Lifecycle of Building Custom Resources

1. Design and implement custom resources.

Foundation REST API extensibility features all you to add custom resources to your REST applications. Your custom resources can be used with the out-of-box Core resources to help you develop a customized web application. The Foundation REST API SDK has a set of toolkits and libraries that you can use to development custom resources.

Here are the toolkits and libraries that are available in the Foundation REST API SDK:

- A set of Core REST Java libraries for custom resource development that enhances the Foundation REST API development of the Core REST services. Java docs and code samples for the shared library are also provided.
- A **Marshalling Framework** to handle XML or JSON message marshalling and unmarshalling of Core REST resources.
- A **Maven-based toolkit** to manage the build process of custom resource projects. An archetype is included and you can use it to create a sample project for your organization.

- An **Ant-based toolkit** to manage the build process of custom resource projects.
A lot of code samples for custom resource development are available.
2. Customize Core REST resources.

Typically, the newly developed resources need to interact with Core resources via link relations or other representations. Foundation REST API provides you with a number of features to customize Core resources.

 - Add topmost custom resources to the Home document. Topmost resources are those resources that do not link from other resources.
 - Add new link relations to Core resources. The newly developed resource can be a forwarded state of a Core resource, so that the Core resource must give a link relation to the new resource.
 3. Repackage the REST WAR file.
 4. Foundation REST API provides the build toolkits and scripts to build custom resources and Core resources into a single WAR file.
 5. Deploy the WAR file to an application server. The custom REST Services can be deployed into the same type of application servers supported by Foundation REST API, as long as the custom REST Services does not bring any library conflict to the application servers.
 6. Consume the extended Foundation REST API. When the custom resources are developed in the same pattern as Core resources, the client consumes the Core resources and custom resources in the same pattern, too. And both of them share the same authentication scheme.

7.2 Deprecated Java APIs

The following JAVA APIs are deprecated:

- The `<com.emc.documentum.rest.wire.xml.AnnotatedXmlMessageWriter.setSuggestedNamespaces()>` method has been deprecated. Manually suggesting namespaces does not produce a well-formatted XML representation. Moving forward, the `AnnotatedXmlMessageWriter` will automatically examine the XML root type to determine the default namespace.
- The `<com.emc.documentum.rest.binding.SerializableField#xmlListitemName>` has been deprecated. It has been replaced with a new annotation called `<com.emc.documentum.rest.binding.SerializableField4XmlList>` that defines a Java List or Array field for custom marshaling and unmarshalling.
- The `<com.emc.documentum.rest.binding.SerializableField#xmlListUnwrap>` has been deprecated. It has been replace by a new annotation called `<com.emc.documentum.rest.binding.SerializableField4XmlList>` that defines a Java List or Array field for custom marshaling and unmarshalling.
- The `<com.emc.documentum.rest.binding.SerializableEntry>` has been deprecated. This annotation was used to Marshall to and Unmarshall from a key-value pair. Since

`java.util.Map` is supported as a single serializable field
`<com.emc.documentum.rest.binding.SerializableEntry>` is no longer needed.

- The `<com.emc.documentum.http.UriFactory>` has been deprecated. It has been replaced by `<com.emc.documentum.rest.context.ResourceUriBuilder>`, which provides a typical pattern that can be used to build a URI.
- The `<com.emc.documentum.rest.http.SupportedMediaTypes>` has been deprecated.
- The `<com.emc.documentum.rest.view.FeedableView.getUriFactory()>` has been deprecated.

It has been replaced by

`<com.emc.documentum.rest.context.ResourceUriBuilder(String)>`

- The `<com.emc.documentum.rest.view.LegacyLinkableView>` has been deprecated. It has been replaced by `<com.emc.documentum.rest.context.ResourceUriBuilder>`.
- `<LinkableView.getUriFactory()>` has been deprecated and the method has been moved to a new parent class called `<LegacyLinkableView>`.
- The `<com.emc.documentum.rest.dfc.ContentManager.setPrimaryContent(String, InputStream, String, String, int, boolean, CheckinPolicy, String)>` has been deprecated. New parameters are added, use `<setPrimaryContent(String, InputStream, long, String, String, String, int, boolean, CheckinPolicy, String)>` instead.
- The `<com.emc.documentum.rest.dfc.ContentManager.setRendition(String, InputStream, String, String, int, String, boolean)>` has been deprecated. New parameters are added, use `<setRendition(String, InputStream, String, String, String, int, String, boolean)>` instead.
- The `<com.emc.documentum.rest.dfc.ContextSessionManager.executeWithinTheContextTransaction(Callable<T>)>` has been deprecated. It has been replaced by `<executeWithinTheContextTransaction(SessionCallable)>`.

7.3 Get started with the development kit

Foundation REST API SDK provides Maven and Ant development kits to build up the custom REST project for users. Follow SDK instructions to set up the first custom REST project.

7.3.1 Maven Toolkit

Since the 7.2 release, a Maven toolkit is available in the SDK of Foundation REST API. The toolkit makes the build process of custom resource projects much easier. Furthermore, the kit introduces a set of deliverables to improve the productivity of custom resource development.



Note: Maven is the recommended build tool in custom resource development. However, it is not required. You can use other tools to build your custom resource projects.

The Maven kit introduces the following deliverables:

- A core REST Java library and dependencies for REST extensibility development.
- Maven pom files that describe the dependencies of the Core REST Java library.

The pom files describe the internal and external library dependencies of Core REST JAR files. You can install these Core REST JAR files into your local Maven repository as third-party JAR files.
- A Maven archetype project for custom resource development.

The archetype project can be used as a template project for custom resource development. This project can be installed in both local and remote repositories.
- A script to install the Maven archetype and dependencies into a local Maven repository.
- A script to create a sample project from the Maven archetype.

7.3.1.1 Creating a Custom Resource Project from the Maven Archetype

Instead of requiring you to create a custom resource project from scratch, Foundation REST API SDK provides you with a Maven archetype to reduce your efforts in project building and coding. You can generate a new project from the archetype and start custom resource development from there.

The Maven archetype helps you create a multi-module project that looks like the following:

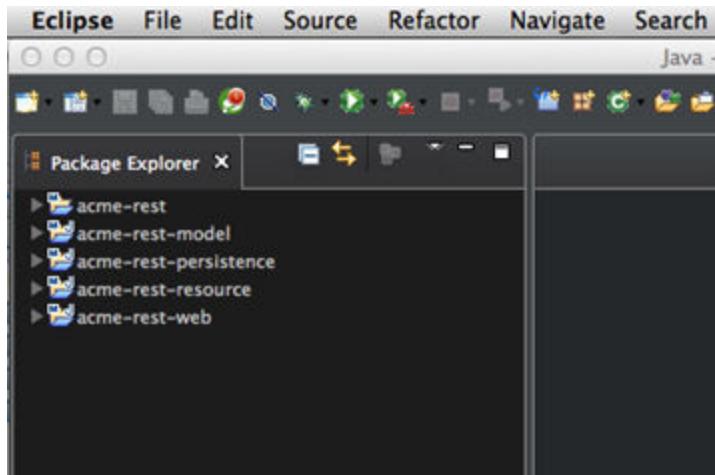


Figure 7-2: Maven Archetype Project Structure

System Requirements:

- Java 7 or Java 8 must be installed, and the path location of the Java installation must be added to the classpath Environment System Variable.
- Maven 3 must be installed and the location of which must be added to the classpath system variable.

Creating a project from the Maven archetype consists of three stages:

1. [Installing the Maven archetype and dependencies to your local repository](#)
2. [Creating a project based on the archetype on page 121](#)

Installing the Maven Archetype and Dependencies

1. Extract the `documentum-rest-sdk-<version-number>.zip` (`tar`) package to your local drive.
2. Navigate to the `documentum-rest-sdk-<version-number>/maven-kit`.
3. View the following Readme files: `archetype-install-guide.md` and `dependencies-install-guide.md`. These two ReadMe files guide you through the installation process of the local Maven archetype and dependencies.
4. The SDK has a script that you can run a to the Maven artifacts. To run this script, follow these instructions:

Open a command line interface

Run the command that applies to your operating system:

- Windows

`dctm-maven-offline-install.bat`

- Linux or Mac

`bash dctm-maven-offline-install.sh`

After the task completes, the core REST archetype is installed to your local Maven repository. By default, the archetype is under the following directory:

`<user_profile>/.m2/repository/com/emc/documentum/rest/extension`

Creating a Project from the Maven Archetype

After the Maven archetype for Foundation REST API is installed, a custom REST project can be created either in an [IDE on page 121](#) or a [command-line prompt](#).

The Eclipse IDE has support for Maven archetype projects. In this section, we use Eclipse to demonstrate how to create a project from the Maven archetype in an IDE.

Creating a Project from the Maven Archetype in an IDE

1. Click **File > New > Project**, select **Maven Project**, and then click **Next**. The New Maven Project wizard appears.
2. In the Select project name and location phase, leave **Create a simple project (skip archetype selection)** unchecked. Click **Next** to proceed to the Select an Archetype phase.

3. In Select an Archetype, click **Configure > Add Local Catalog**. The Local Archetype Catalog box dialog appears.
4. In the **Catalog File** field, click **Browse** to navigate to your .m2 folder and then select the archetype-catalog.xml file. In the **Description** field, enter the description of the archetype, and then click **OK**.
5. Click **OK** to go back to Select an Archetype. In the **Catalog** field, select the catalog you specified in [step 4](#), and then select the **Include snapshot archetypes** check box. The <documentum-rest-extension-archetype> artifact appears. Select this artifact and click **Next**.
6. Enter the information about *group ID*, *artifact ID*, *version*, and *package*, and click **Finish**. Eclipse starts to build a project from the Maven archetype.

If the working IDE environment is not the Eclipse, you can alternatively set up the archetype project with command lines. Maven and Java are required to be set into the system path.

Creating a Project from the Maven Archetype in a Common-Line Prompt

1. Navigate to the documentum-rest-sdk-version-number/maven-kit directory.
2. Run the following script command that applies to your operating system:
 - Windows
`dctm-rest-getstarted.bat`
 - Linux or Mac
`bash dctm-rest-getstarted.sh`

After the task completes, a new project is created in the directory that is generated.

7.3.1.1.1 Verifying the Project

Out-of-the-box, two custom resources – alias-sets and alias-set are embedded in the Maven archetype. Follow these steps to build your project, deploy the WAR file to an application server, and then access the alias-sets resource to verify the project.

1. Enter the directory <*artifact_id*>-web/src/main/resources of your project and create the dfc.properties file according to your Documentum CM Server installation. For more information, see *OpenText Documentum Content Management - Server and Server Extensions Installation Guide (EDCSY250400-IGD)*.
2. Navigate to the project folder (the <*artifact_id*> directory) and then run the following Maven task to build the project:

```
mvn install
```

The <*artifact_id*>-web-1.0.war file is created under the <*artifact_id*>/<*artifact_id*>-web/target directory.

3. Deploy the WAR file to an application server. For more information, see *OpenText Documentum Content Management - Server and Server Extensions Installation Guide (EDCSY250400-IGD)*.
4. Access the alias-sets with a GET request to a URL that looks like the following:

```
http://<localhost:port>/acme-rest/repositories/<repositoryName>/alias-sets
```

Upon a successful deployment, the operation returns a collection of alias sets in the repository.

```
<feed xmlns="http://www.w3.org/2005/Atom"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <id>http://localhost:8080/dctm-rest/repositories/dctm72/alias-sets</id>
  <title>Alias Sets</title>
  <author>
    <name>OpenText Documentum</name>
  </author>
  <updated>2014-08-20T17:00:36.197+08:00</updated>
  <link rel="self"
        href="http://localhost:8080/dctm-rest/repositories/dctm72/alias-sets"/>
  <entry>
    <id>http://localhost:8080/dctm-rest/repositories/dctm72/alias-sets/
6600000b80000105</id>
    <title>AdminAccess</title>
    <author>
      <name>Administrator</name>
      <uri>
        http://localhost:8080/dctm-rest/repositories/dctm72/users/Administrator
      </uri>
    </author>
    <updated>2014-08-20T17:00:36.197+08:00</updated>
    <published>2014-08-20T17:00:36.197+08:00</published>
    <content src="http://localhost:8080/dctm-rest/repositories/dctm72/alias-sets/
6600000b80000105"/>
    <link rel="self" href="http://localhost:8080/dctm-rest/repositories/dctm72/
alias-sets/6600000b80000105"/>
  </entry>
  <entry>
    <id>
      http://localhost:8080/dctm-rest/repositories/dctm72/alias-sets/6600000b8000050a
    </id>
    <title>auxiliary_alias_set</title>
    <author>
      <name>dctm72</name>
      <uri>
        http://localhost:8080/dctm-rest/repositories/dctm72/users/dctm72
      </uri>
    </author>
    <updated>2014-08-20T17:00:36.197+08:00</updated>
    <published>2014-08-20T17:00:36.197+08:00</published>
    <content src="http://localhost:8080/dctm-rest/repositories/dctm72/alias-sets/
6600000b8000050a"/>
    <link rel="self" href="http://localhost:8080/dctm-rest/repositories/dctm72/
alias-sets/6600000b8000050a"/>
  </entry>
  <entry>...</entry>
  ...
  <entry>...</entry>
</feed>
```

7.3.2 Ant Toolkit

The Foundation REST API SDK contains an Ant-based toolkit that helps Apache Ant users manage the build process of custom resource projects. A folder named `ant-kit` is located under the root directory of the SDK, which contains data needed for the Ant build creation.

Creating a Custom Resource Build with Ant

1. Navigate to the `ant-kit/war/web-information/classes` folder and then edit the `dfc.properties` file according to your Documentum CM Server configuration.

For more information about how to configure `dfc.properties`, see *OpenText Documentum Content Management - Server and Server Extensions Installation Guide (EDCSY250400-IGD)*.

2. Navigate to the `ant-kit` folder where the `build.xml` file is located and then run the `ant all` command to create the WAR file.

Enter the application name and version for your web archive when you are prompted to.

When the command completes, a multi-module project is created together with a WAR file under the `ant-kit/dist` directory.

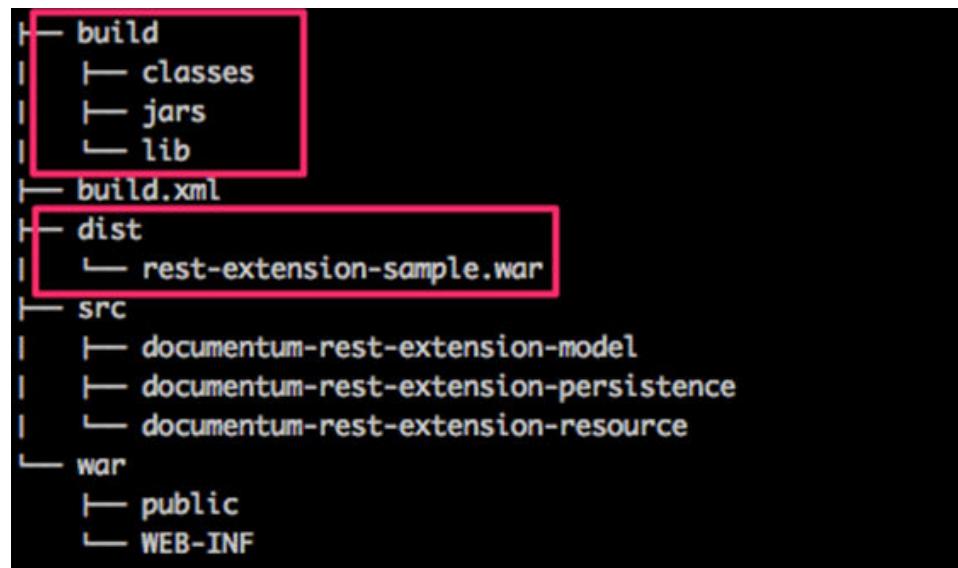


Figure 7-3: ANT Project Structure

Out-of-the-box, a custom resource alias-sets is embedded in the Ant-based toolkit. After you deploy the WAR file, access alias-sets with a GET request to a URL that looks similar to:

`http://host:port/acme-rest/repositories/MyRepository/alias-sets`

If both the build creation and WAR file deployment are successful, the operation returns a collection of alias sets in the repository.

If you want to add more custom models, controllers, and resources, follow the instructions in `ant-kit/readme.txt`.

7.4 Architecture of extensible Foundation REST API

The extensibility feature of Foundation REST API allows you to create custom resources in the same RESTful pattern as Core resources, with the objective to reuse the Core REST infrastructure as much as possible. To understand where the extension points are within the whole infrastructure, we need to have an overview of the architecture of Foundation REST API. The following diagram illustrates the overall architecture of Foundation REST API.

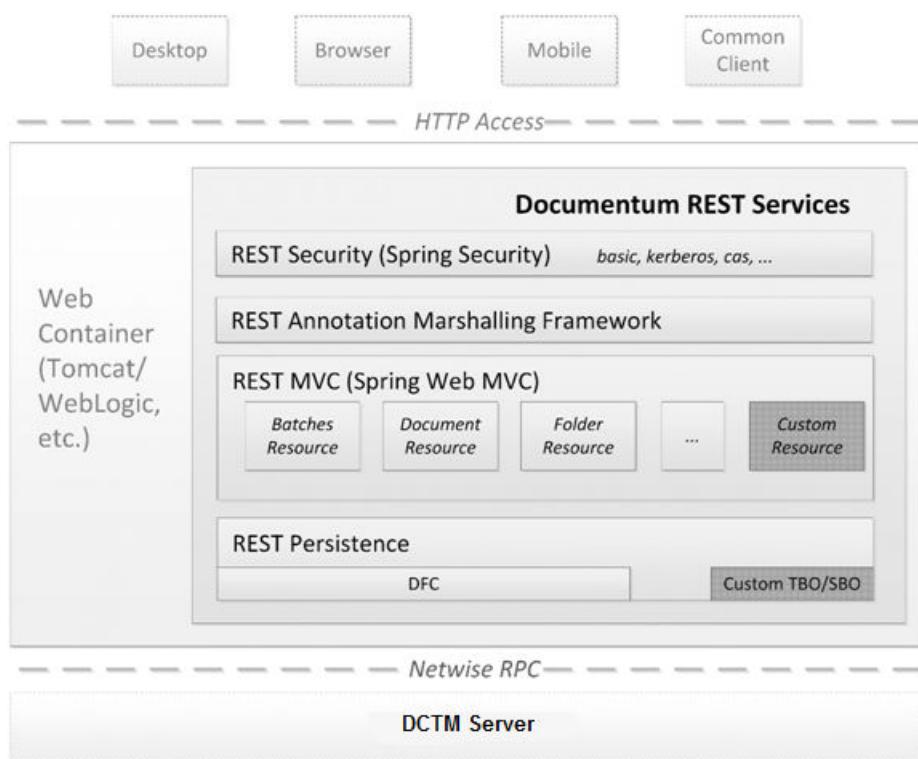


Figure 7-4: Foundation REST API Architecture

The whole Foundation REST API is built as a single WAR application, including both Core resources and custom resources.

Foundation REST API leverages Spring Security to provide various authentication schemes. Any authentication scheme configured in a deployment applies to both Core resources and custom resources. In the current release, the security component is not exposed for customization.

Foundation REST API leverages Spring Web MVC Framework to build all REST resources. The Spring Web MVC sets clear separations of roles and makes implementations of components pluggable. By taking advantages of Spring Web MVC, Foundation REST API provides various means of extensibility features, enabling you to add custom resources or customize Core resources with flexibility and efficiency. We call this as **Foundation REST API MVC**.

On the top of the Foundation REST API MVC, Foundation REST API provides a unique REST annotation framework which helps to Marshall REST Java resource models into JSON and/or XML representations, and Unmarshall the JSON and/or XML representations into resource models. When developing custom resources, you just need to focus on the resource model design. The marshalling and unmarshalling are done by Foundation REST API at the framework level.

Under the bottom of the Foundation REST API MVC, persistence APIs communicate with Documentum CM Server repositories. Foundation REST API provides a lot of common APIs to manipulate persistent data in the Documentum CM Server repositories. Besides, Foundation REST API provides the hands-on session APIs that integrate to the Security component for different authentication schemes and exposes them as the uniform interfaces in the persistence component.

7.4.1 Foundation REST API Security

Custom resources often use the same authentication scheme as Core resources within the same WAR deployment. More precisely, Foundation REST API uses a set of Spring security filters to apply specific authentication schemes, and each security filter determines which resource URI pattern(s) are included in the specified authentication scheme.

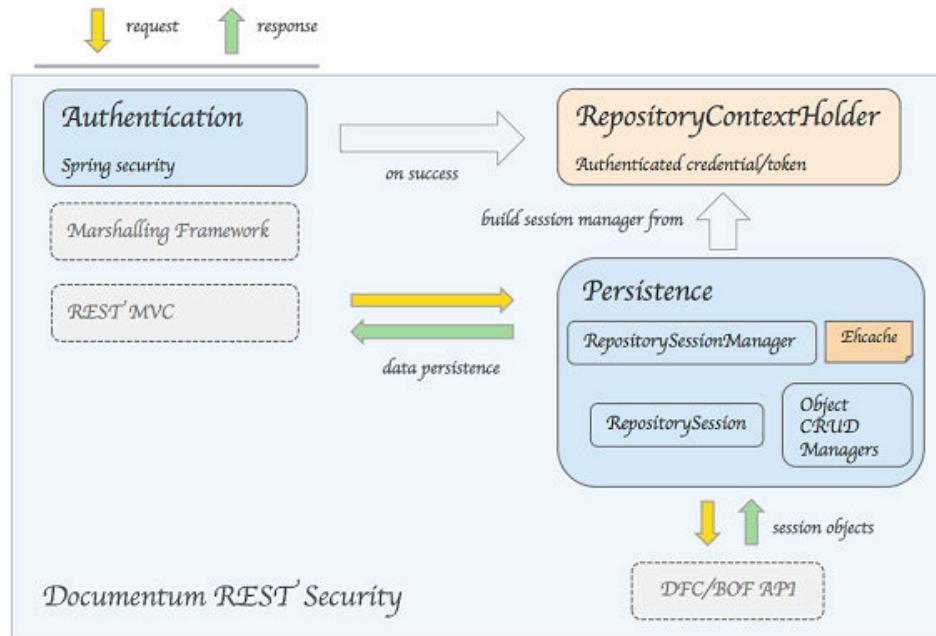
Foundation REST API applies many authentication schemes by using the resource URI pattern `/repositories/{repositoryName}` and all of its subsequent path segments. Therefore, when a custom resource's URI is created with the prefix `/repositories/{repositoryName}/`, such as `/repositories/{repositoryName}/alias-sets`, access to that resource is authenticated by the configured authentication scheme(s). When this is not the case, such as in `/help`, access to the resource is anonymous.

For more information, see [Custom Authentication Development](#).

The security filters actually perform the authentication for access to Documentum CM Server repositories or third-party Single-sign-On entities. When the authentication procedure is successful, the Persistence layer of the resource implementation retrieves the authenticated Foundation Java API session manager. It retrieves the Foundation Java API session manager by using the `com.emc.documentum.rest.dfc.RepositorySession` and `com.emc.documentum.rest.dfc.RepositorySessionManager` APIs. A custom persistence API can extend the `com.emc.documentum.rest.dfc.SessionAwareAbstractManager` API to get the repository session.

For more information, see [Persistence: Session Management](#).

The following diagram illustrates the relationship between REST authentication and the session related APIs:



Here is an explanation on what is happening in the diagram:

- When authentication succeeds, it stores the authenticated user credential or Single Sign-On token in the `com.emc.documentum.rest.config.RepositoryContextHolder` Core REST class.
- Resource controllers that are used in REST MVC call persistence object managers to perform any persistent object operations.
- Persistent object managers are auto-wired with a `com.emc.documentum.rest.dfc.RepositorySession` Java bean.
- The `RepositorySession` Java bean is auto-wired using a Singleton of the `com.emc.documentum.rest.dfc.RepositorySessionManager` Java bean.
- The Singleton of the `RepositorySessionManager` Java bean retrieves the Foundation Java API session manager according to the user principle from the `RepositoryContextHolder` Java bean for the current Request.
- Ehcache is used to store the Foundation Java API session manager, and the cache key is generated from the `RepositoryContextHolder` Java bean.
- The `RepositoryContextHolder` bean is cleared after each Request exits, however the Ehcache for the Foundation Java API session manager remains for a period of time.

7.4.2 Foundation REST API MVC

Foundation REST API MVC performs customizations on Spring Web MVC to facilitate REST resource development. A typical resource implementation contains three layers, the model, the controller, and the view.

- The model layer is the data holder for a resource where Foundation REST API annotations are defined for marshalling and unmarshalling.
- The controller layer is the center of the resource implementation where it defines the Request and Response mappings, as well as calls the persistence to manipulate the data. The controller implementation mainly uses Spring annotation `@Controller` to implement the resource. The controller method accepts and returns the model instances.
- The view layer is a wrapper of the model where representation-related information, such as links and atom attributes, are resolved. The output of the view is still the model instance since the model is the only entity for marshalling and unmarshalling. The view implementation binds to specific models and controllers.

The following diagram illustrates the relationship of the three layers:

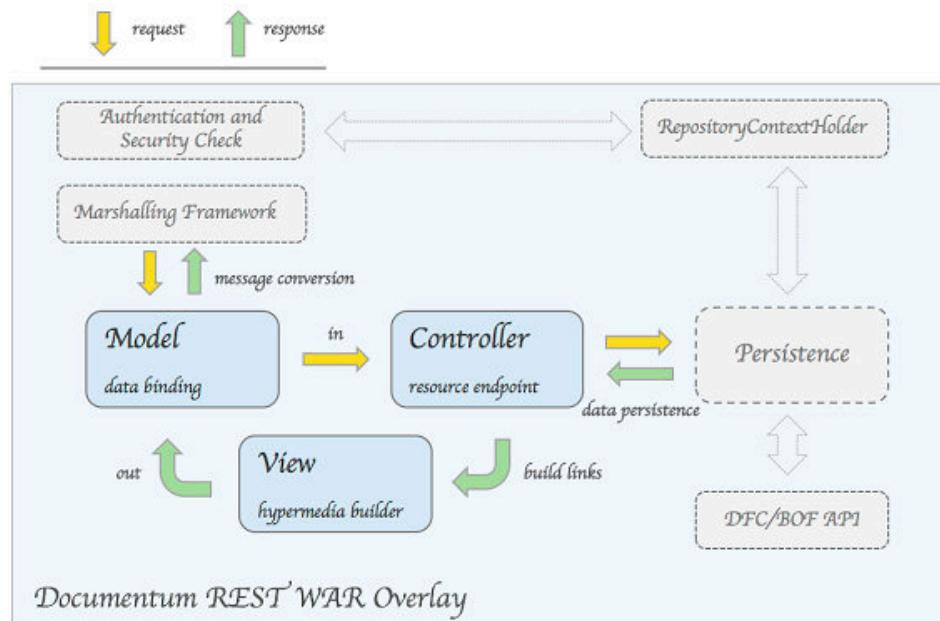


Figure 7-5: Foundation REST API MVC

A resource controller can be bound to one or several view definitions. Each view definition renders a specific model with regard to links and other customizations. The input of output of a controller method is the model class which is annotated

with Foundation REST API annotations. [Developing Custom Resources](#) discusses details of how [Foundation REST API MVC](#) is used in custom resource development.

7.4.3 Foundation REST API Persistence

Foundation REST API provides a number of APIs to manipulate persistent data in Documentum CM Server repositories, such as folder, document, content, and so on. Persistence APIs are managed as Spring beans, and loaded by resource implementations with Spring annotation `@Autowired`. These APIs are mainly in the `com.emc.documentum.rest.dfc` Java package and its sub packages.

Custom resources may also need to write new persistence APIs or integrate their existing type-based objects (TBOs) or service-based objects (SBOs) into the REST implementation. The Custom persistence APIs must be written using the same pattern as other Core persistence APIs, and they must be loaded with Spring bean configurations.

Typically, you can develop a new persistence API with following procedure:

1. Write a Java interface.

```
public interface UserManager {
    UserObject createUser(String name, String password);
}
```

2. Write a Java class to implement this interface.

```
public class UserManagerImpl extends SessionAwareAbstractManager
implements UserManager {
    public UserObject createUser(String name, String password) {
        ...
    }
}
```



Note: For more information on the usage of Foundation Java API sessions in the object manager, see ["Persistence: Session Management"](#) on page 204.

3. Implement it as a Java bean.

There are two ways to create a Java bean:

- Spring Java code configuration
- Using an XML namespace



Example 7-1: Create a Java Bean Using Spring Code Configuration

```
@Configuration
@ComponentScan(
    basePackages = { "com.acme" },
    excludeFilters = {
        @ComponentScan.Filter(
            type = FilterType.CUSTOM,
            classes = {
                com.emc.documentum.rest.context.ComponentScanExcludeFilter.class
            })
    }
}
```

```

    }

    public class CustomContextConfig {
        @Bean(name="customUserManager")
        public UserManager customUserManager(){
            return new UserManagerImpl();
        }
    }
}

```

You must ensure that the package where you created your custom configuration class is specified in the `rest.context.config.location` property within the `rest-api-runtime.properties` file. For example, when the `CustomContextConfig` class is under the `com.acme.context.config` package, the `rest.context.config.location` property should be defined in runtime properties file as shown:

```
rest.context.config.location=com.acme.context.config
```

When you use the Spring `@ComponentScan` annotation in your custom defined configuration class, the `com.emc.documentum.rest.context.ComponentScanExcludeFilter` exclude filter is mandatory. This exclude filter ensures that the Spring framework loads all of the resources that you have defined.



Example 7-2: Create a Java Bean Using Spring XML Namespace

Define the bean in class path file: `/META-INF/spring/custom.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" ...>
    <bean id="customUserManager" class="com.acme.UserManagerImpl"/>
</beans>
```



4. Reference the persistence in resources.

```
public CustomUserController extends AbstractController {
    @Autowired UserManager customUserManager;
    ...
}
```

For more information, see [persistence programming](#) for the details of implementing a persistence API.

7.5 Foundation REST API marshalling framework

7.5.1 Overview

Foundation REST API provides a simple yet powerful marshalling framework to facilitate object to object communication. The Foundation REST API marshalling framework introduces a set of Java annotations that bind models directly to your XML or JSON representations. This data binding eliminates the need to write message converters for custom data models.

By default, core and custom resources use the annotation marshalling framework to convert XML and JSON messages. However, because Foundation REST API also supports Spring message conversions, these messages can be altered by users who understand how to work with Spring message conversions.

7.5.1.1 Features of the REST Marshalling Framework

The REST Marshalling Framework has the following features:

- Out-of-the-box message marshalling

The REST Marshalling Framework handles the serialization of Java objects into HTTP messages, and the deserialization of HTTP messages into Java object under [Foundation REST API MVC](#).

- Out-of-the-box REST data models

A set of annotated Java classes for the common REST data models are packaged within the Core REST library and can be reused or even extended.

- Supports both XML and JSON

The default XML and JSON message converters are implemented within the Core REST library. A common set of Java annotations are used for both XML and JSON representations. Therefore, when a Java class is annotated, both XML and JSON representation are supported.

The resource controller returns the annotated class instance directly, leaving all the message conversion work to [Foundation REST API MVC](#) and the REST Marshalling Framework.

```
@Controller
public class MyDaoController extends AbstractController {
    @RequestMapping(value = "/my-dao/{id}", method = RequestMethod.GET)
    @ResponseBody
    public MyDao get(@PathVariable("id") String id, @RequestUri final UriInfo uriInfo) {
        MyDao myDao = dfcObjectManager.get(id);
        return myDao;
    }
    ...
}
```

7.5.2 Annotations

This marshalling framework introduces the following annotations:

- `@SerializableType`
Used on a Java class to serialize an object of the class to a REST structural representation.
- `@SerializableField`
Used on a Java class field to serialize the Java object field to a sub element/property of a REST representation.
- `@SerializableField4XmlList`
Used on a Java class field to serialize the Java List type field to a sub element/property of a REST representation.
- `@SerializableField4XmlMap`
Used on a Java class field to serialize the Java Map type field to a sub element/property of a REST representation.

All annotations work for REST message marshalling and unmarshalling.

7.5.2.1 `@SerializableType`

The annotation `@SerializableType` under Java package `com.emc.documentum.rest.binding` indicates that a Java object is intended to be serialized to a REST representation. This annotation provides a number of attributes enabling you to customize the marshalling and unmarshalling behavior.

Table 7-1: SerializableType Attributes

Attributes	Description	Supported Formats	Marshall / Unmarshall	Default
value	Specifies the serializable name	JSON/XML	in and out	not set

Attributes	Description	Supported Formats	Marshall / Unmarshall	Default
fieldVisibility	<p>Specifies whether to serialize all or some fields according to the access modifiers at the class level:</p> <ul style="list-style-type: none"> • ALL - all fields are intended to be serialized but static fields are excluded. Static fields are serializable only when they are explicitly annotated with the <i><@Serializable Field></i> annotation. • PUBLIC - public fields are intended to be serialized but public static fields are excluded. Public static fields are serializable only when they are explicitly annotated with the <i><@Serializable Field></i> annotation. • NONE - no fields are intended to be serialized. <p>Settings of the annotation</p>	JSON/XML	in and out	ALL

Attributes	Description	Supported Formats	Marshall / Unmarshall	Default
	<p>@Serializable Field take precedence over this attribute. For example, when fieldVisibility is set to none, properties with the annotation @Serializable Field are still intended to be serialized.</p> <p>For more information, see Example A on page 138 and Example B on page 138.</p>			
fieldOrder	<p>Specifies the order of fields in the REST representation for a serialized object.</p> <p>For fields that can be serialized but not appearing in this list, they are presented at the end of the REST representation.</p> <p>For fields appearing in the list, but not serialized, they are ignored.</p> <p>For more information, see Example C on page 139.</p>	JSON/XML	out	not set

Attributes	Description	Supported Formats	Marshall / Unmarshall	Default
ignoreNullFields	<p>Specifies whether to ignore fields with null values when marshalling.</p> <p>For more information, see Example D on page 139.</p>	JSON/XML	out	true
inlineField	<p>Indicates that a non-primitive type field is intended to be moved to an upper level in the REST representation.</p> <p>Note that when you set <code>inlineField</code> to a certain field, all other fields in the class are not serialized.</p> <p>For more information, see Example E on page 140.</p>	JSON/XML	out	not set
xmlValueField	<p>Specifies a field from which the value will be taken as the XML element value of this type.</p> <p>For more information, see Example F on page 140.</p>	XML	in and out	not set

Attributes	Description	Supported Formats	Marshall / Unmarshall	Default
jsonWriteRootAsField	<p>For JSON marshalling, specifies whether to write the root name (specified by the value attribute) into a field specified by the annotation <code>jsonRootField</code>.</p> <p>For more information, see Example G on page 140.</p> <p>If <code>jsonWriteRootAsField</code> is <code>false</code> and the annotated type has a super class annotated as well, the JSON root is marshaled so that the type information exists in JSON representation which is useful during unmarshalling.</p>	JSON	out	false
jsonRootField	Specifies the field where the JSON root name is displayed.	JSON	in and out	json-root
xmlNS	<p>Specifies the namespace for XML representation.</p> <p>For more information, see Example H on page 141.</p>	XML	out	not set inherited from the parent class

Attributes	Description	Supported Formats	Marshall / Unmarshall	Default
xmlNSPrefix	<p>Specifies the namespace prefix for XML representation.</p> <p>For more information, see Example H on page 141.</p>	XML	out	not set inherited from the parent class
inheritValue	<p>Specifies whether to reuse the serializable name of the super type.</p> <ul style="list-style-type: none"> • true: the serializable name of this type is set to the serializable name of the super type. Only types that do not need to be unmarshalled can have this attribute set to <i>true</i>. • false: the serializable name of this type depends on the value on page 132 attribute. 	JSON/XML	in and out	false

Attributes	Description	Supported Formats	Marshall / Unmarshall	Default
unknownFieldDeserializers	You can specify any AnnotatedFieldDeserializer implementation to process an unknown field. However, the unknownFieldDeserializers annotation field is only used when deserializing.	JSON/XML	out	Annotated Exceptional UnknownField Deserializer

7.5.2.1.1 Examples

This section lists several examples explaining the detailed usage of the attributes that are available in the @SerializableType annotation.

Example 7-3: Public Field Visibility

In the following example, the non-public field `id` is not serialized.

Java Definition	XML	JSON
<pre>@SerializableType (value = "business-object", fieldVisibility = SerializableType.FieldVisibility.PUBLIC) public class BusinessObject { private String id; public Integer vstamp; public BusinessObject(String id, Integer vstamp) { this.id = id; this.vstamp = vstamp; } // Field visibility for public fields public static void main(String[] args) { BusinessObject bo = new BusinessObject("09ae2d", 6); } }</pre>	<pre><?xml version='1.0' encoding="UTF-8'?> <business-object> <vstamp>6</vstamp> </business-object></pre>	<pre>{ "vstamp":6 }</pre>



Example 7-4: None Field Visibility

In the following example, only the field `active` annotated by `@SerializableField` is serialized.

Java Definition	XML	JSON
<pre>@SerializableType (value = "business-object", fieldVisibility = SerializableType.FieldVisibility.NONE) public class BusinessObject { private String id; private Integer vstamp; @SerializableField private Boolean active; public BusinessObject(String id, Integer vstamp, Boolean active) { this.id = id; this.vstamp = vstamp; this.active = active; } // field visibility for none fields public static void main(String[] args) { BusinessObject bo = new BusinessObject("00ae2d", 6, true); } }</pre>	<pre><?xml version='1.0' encoding='UTF-8'?> <business-object> <active>true</active> </business-object></pre>	<pre>{ "active":true }</pre>



Example 7-5: Field Order

In the following example, fields are serialized according to the order specified in `fieldOrder`.

Java Definition	XML	JSON
<pre>@SerializableType (value = "business-object", fieldOrder = {"vstamp", "id"}) public class BusinessObject { private String id; private Integer vstamp; public BusinessObject(String id, Integer vstamp) { this.id = id; this.vstamp = vstamp; } // field order public static void main(String[] args) { BusinessObject bo = new BusinessObject("00ae2d", 6); } }</pre>	<pre><?xml version='1.0' encoding='UTF-8'?> <business-object> <vstamp>6</vstamp> <id>00ae2d</id> </business-object></pre>	<pre>{ "vstamp":6, "id":"00ae2d" }</pre>



Example 7-6: Ignore Null Fields

In the following example, the null field `id` is not serialized.

Java Definition	XML	JSON
<pre>@SerializableType (value = "business-object", ignoreNullFields = true) public class BusinessObject { private String id; private Integer vstamp; public BusinessObject(String id, Integer vstamp) { this.id = id; this.vstamp = vstamp; } // ignore null fields public static void main(String[] args) { BusinessObject bo = new BusinessObject(null, 6); } }</pre>	<pre><?xml version='1.0' encoding='UTF-8'?> <business-object> <vstamp>6</vstamp> </business-object></pre>	<pre>{ "vstamp":6 }</pre>



➡ Example 7-7: Inline Field

In the following example, the inline filed event is moved to an upper level in the REST representation and only fields in event are serialized.

Java Definition	XML	JSON
<pre>@SerializableType {value = "business-object", inlineField = "event"} public class BusinessObject { private String id; private Integer vstamp; private Event event; public BusinessObject(String id, Integer vstamp, Event event) { this.id = id; this.vstamp = vstamp; this.event = event; } @SerializableType public static class Event { private String eventId; private Boolean active; public Event (String eventId, Boolean active) { this.eventId = eventId; this.active = active; } } // inline public static void main(String[] args) { BusinessObject bo = new BusinessObject("09ae2d", 6, new Event("775200", true)); } }</pre>	<pre><?xml version='1.0' encoding='UTF-8'?> <event> <eventId>775200</eventId> <active>true</active> </event></pre>	<pre>{ "eventId": "775200", "active": true }</pre>



➡ Example 7-8: XML Value Field

In the following example, vstamp is serialized as the value of the business-object element and id is serialized as an attribute.

Java Definition	XML	JSON
<pre>@SerializableType {value = "business-object", xmlValueField = "vstamp"} public class BusinessObject { @SerializableField(xmlAsAttribute = true) private String id; private Integer vstamp; public BusinessObject(String id, Integer vstamp) { this.id = id; this.vstamp = vstamp; } // xml value field public static void main(String[] args) { BusinessObject bo = new BusinessObject("09ae2d", 6); } }</pre>	<pre><?xml version='1.0' encoding='UTF-8'?> <business-object id="09ae2d">6</business-object></pre>	<pre>{ "id": "09ae2d", "vstamp": 6 }</pre>



➡ Example 7-9: JSON Write Root

In the following example, business-object, which is specified in value is added to name, which is the default value of jsonRootField.

Java Definition	XML	JSON
<pre>@SerializableType (value = "business-object", jsonWriteRootAsField = true) public class BusinessObject { private String id; private Integer vstamp; public BusinessObject(String id, Integer vstamp) { this.id = id; this.vstamp = vstamp; } // json write root public static void main(String[] args) { BusinessObject bo = new BusinessObject("09ae2d", 6); } }</pre>	<pre><?xml version='1.0' encoding='UTF-8'?> > <business-object> <id>09ae2d</id> <vstamp>6</vstamp> </business-object></pre>	<pre>{ "name": "business-object", "id": "09ae2d", "vstamp": 6 }</pre>



Example 7-10: Namespace

In the following example, XML namespaces and prefixes are added.

Java Definition	XML	JSON
<pre>@SerializableType (value = "business-object", xmlNSPrefix = "dm", xmlNS = "http://documentum.emc.com") public class BusinessObject { private String id; private Integer vstamp; private Event event; public BusinessObject(String id, Integer vstamp, Event event) { this.id = id; this.vstamp = vstamp; this.event = event; } @SerializableType (value = "event", xmlNSPrefix = "acme", xmlNS = "http://acme.org") public static class Event { private Boolean active; public Event (Boolean active) { this.active = active; } } // xml namespace public static void main(String[] args) { BusinessObject bo = new BusinessObject("09ae2d", 6, new Event(true)); } }</pre>	<pre><?xml version='1.0' encoding='UTF-8'?> <business-object xmlns="http://documentum.emc.com" xmlns:acme="http://acme.org"> <id>09ae2d</id> <vstamp>6</vstamp> <acme:events> <acme:active>true</acme:active> </acme:events> </business-object></pre>	<pre>{ "id": "09ae2d", "vstamp": 6, "event": { "active": true } }</pre>



7.5.2.1.2 Unmarshalling with Undefined Attributes

A new property called *unknownFieldDeserializers* has been added to the annotation *SerializableType*. You can specify any *AnnotatedFieldDeserializer* implementation to process an unknown field. However, the *unknownFieldDeserializers* annotation field is only used when deserializing. The default value of the *unknownFieldDeserializers* property is *AnnotatedExceptionalUnknownFieldDeserializer* implementation, which throws *AnnotationParseException* exception when finding any unknown element.

- For backward compatibility Unknown elements parsing (deserializing) will have the *AnnotationParseException* exception added in Foundation REST API version 7.3, which may be correctly parsed by 7.2.

The solutions for the preceding issue are as follows:

- Avoid sending elements that do not exist

- Customize the *unknownFieldDeserializers*
- When you use *unknownFieldDeserializers*, you will want to ignore all unknown elements.

In some cases, unknown elements are useless, or don't need to be processed. Although in these cases, it is recommended to throw the exception to let the client know that the server can also ignore these unknown elements.

In addition to throwing an exception with *AnnotatedExceptionalUnknownFieldDeserializer*, there is another default implementation, *AnnotatedIgnoreableUnknownFieldDeserializer*, that is also provided. This implementation ignores the unknown elements directly. To use it, just set it to *unknownFieldDeserializers* of the *SerializableType*.

Here is an example that shows you how to do this:

```
@SerializableType(value="model",
    unknownFieldDeserializers={AnnotatedIgnoreableUnknownFieldDeserializer.class})
public class SomeModel2 {
    private String fieldA;
    private String fieldB;
}
```

The unknown elements are ignored during deserialization process. Ignoring the unknown elements allows the existing elements to be correctly populated. Here is an example that illustrates this point:

```
<model>
    <fieldA>a</fieldA>
    <fieldB>b</fieldB>
    <unknownField>x</unknownField>
</model>

<model unknownAttribute="x">
    <fieldA>a</fieldA>
    <fieldB>b</fieldB>
</model>

{
    "fieldA": "a",
    "fieldB": "b",
    "unknownField": "x"
}
```

- Specify empty *unknownFieldDeserializers*

An empty *unknownFieldDeserializers* explicitly sets *unknownFieldDeserializers* to {} for a class. Here's a example that illustrates this technique:

```
@SerializableType(value="model", unknownFieldDeserializers={})
public class SomeModel3 {
    private String fieldA;
    private String fieldB;
}
```

Setting the empty *unknownFieldDeserializers* property causes the binding framework to determine whether to process the unknown field or ignore it. This is useful because some 'unknown' elements may be meaningful and may need to be processed by the system.

The typically used for inheritance. You declare the field as a parent class, but the field instance is actually a child of the parent. The server tries to deserialize the

representation with the parent while the attributes belonging to the child are unknown to the parent. The following example helps to understand this use case:

```
@SerializableType(value="parent",unknownFieldDeserializers= {})
public abstract class AbstractParent {
    private String name;
}

@SerializableType(value="child-a", jsonWriteRootAsField=true,
jsonRootField="parent")
public class ChildA extends AbstractParent {
    @SerializableField(xmlAsAttribute=true)
    private int age;
}

@SerializableType(value="child-b", jsonWriteRootAsField=true,
jsonRootField="parent")
public class ChildB extends AbstractParent {
    private boolean graduated;
}

@SerializableType(value="group")
public class Group {
    List<AbstractParent> children;
}
```

There are three instances, one parent and two children. Each child has its own properties. The parent has the empty *unknownFieldDeserializers* property defined. The fourth instance has a List of *AbstractParent* data type. The output representation may be:

```
<?xml version='1.0' encoding='UTF-8'?>
<group>
    <children>
        <child-a age="10">
            <name>child a</name>
        </child-a>
        <child-b>
            <graduated>true</graduated>
            <name>child b</name>
        </child-b>
    </children>
</group>

[{"children": [
    {"parent":"child-a","age":10,"name":"child a"}, 
    {"parent":"child-b","graduated":true,"name":"child b"}]
```

When deserializing the representation for the *Group* element, the system tries to use *AbstractParent* to parse the child representation. The child contains the 'unknown' elements 'age' and 'graduated' for *AbstractParent*. With empty *unknownFieldDeserializers*, the binding system processes it correctly.

The server throws an exception when you don't set *unknownFieldDeserializers* to empty and the default *AnnotatedExceptionalUnknownFieldDeserializer* is used.

When you set the *AnnotatedIgnoreableUnknownFieldDeserializer*, unknown elements are ignored. The server creates *AbstractParent* with its own elements. When *AbstractParent* is not abstract, the instance can be created. When it is abstract, which is the case here, the exception is thrown.

The difference between *AnnotatedIgnoreableUnknownFieldDeserializer* and empty *unknownFieldDeserializers* is that

AnnotatedIgnoreableUnknownFieldDeserializer ignores the elements, whereas empty *unknownFieldDeserializers* lets the server determine how to deserialize it. The server may determine to deserialize it or ignore the unknown elements.



Note: When using inheritance, the *unknownFieldDeserializers* must be set to empty.

- Implement customized deserializer

The customized deserializer should implement *AnnotatedFieldDeserializer* directly. You can extend *AbstractFieldJsonDeserializer* or *AbstractFieldXmlDeserializer*.

```
@SerializableType(value="model", fieldVisibility=FieldVisibility.NONE,
    unknownFieldDeserializers= {MyFieldJsonDeserializer.class,
                                MyFieldXmlDeserializer.class})
public class SomeModel4 {
    @SerializableField
    private String name;
    private String myfoo;

    //get/set methods

    public static class MyFieldXmlDeserializer extends AbstractFieldXmlDeserializer {
        @Override
        public Object deserialize(Object parser, Object object, String name,
                                 SerializableFieldMeta fieldNode, Map<String, Object>
infoMap) {
            if(!"foo".equals(name)) {
                throw new AnnotationParseException("E_UNDEFINED_FIELD_NAME", name);
            }
            if(parser instanceof StartElement) {
                return readAttribute((SomeModel4)object, (StartElement)parser, name);
            } else {
                return readElement((SomeModel4)object, asEventReader(parser), name);
            }
        }
        @Override
        public boolean deserializable(Object next, SerializableFieldMeta fieldNode) {
            return true;
        }
        private String readElement(SomeModel4 childD, XMLEventReader reader, String
name) {
            try {
                XMLEvent event = null;
                while((event=reader.nextEvent())!=null) {
                    if(event.isCharacters()) {
                        childD.setMyfoo(event.asCharacters().getData());
                    } else if(event.isEndElement()) {
                        EndElement ee = event.asEndElement();
                        if(ee.getName().getLocalPart().equals(name)) {
                            break;
                        }
                    }
                }
            } catch (XMLStreamException e) {
                throw new AnnotationParseException(e, "E_PARSE_XML_FIELD_ERROR", name,
this);
            }
            return null;
        }

        private String readAttribute(SomeModel4 childD, StartElement startElement,
                                     String
name) {
            Iterator<?> i = startElement.getAttributes();
            while(i.hasNext()) {
```

```

        Attribute attr = (Attribute)i.next();
        if(name.equals(attr.getName().getLocalPart())) {
            childD.setMyfoo(attr.getValue());
            break;
        }
    }
    return null;
}

public static class MyFieldJsonDeserializer extends
AbstractFieldJsonDeserializer {
    @Override
    public Object deserialize(Object parser, Object object, String name,
        SerializableFieldMeta fieldNode, Map<String, Object>
infoMap) {
        if("foo".equals(name)) {
            JsonParser jParser = asJacksonParser(parser);
            try {
                JsonToken next = jParser.nextToken();
                if(next == JsonToken.VALUE_STRING) {
                    ((SomeModel4)object).setMyfoo(jParser.getText());
                }
            } catch (IOException e) {
                throw new AnnotationParseException(e, "E_PARSE_JSON_FIELD_ERROR",
                    name, this);
            }
            return null;
        } else {
            throw new AnnotationParseException("E_UNDEFINED_FIELD_NAME", name);
        }
    }
    @Override
    public boolean deserializable(Object next, SerializableFieldMeta fieldNode) {
        return true;
    }
}

```

The `<SomeModel4>` class defines two *unknownFieldDeserializers* classes; one for JSON and one for XML. Use one of the classes (for JSON or XML) to parse all raw representation according your requirements.

Note the following items in the example:

1. `<MyFieldXmlDeserializer>` processes the raw XML representation, and only the `<foo>` element is able to be processed.
2. `<StartElement>` is used to parse the XML attributes.
3. `<XMLEventReader>` is used for parsing the XML elements.
4. `<MyFieldJsonDeserializer>` reads the raw JSON representation, and only the `<foo>` element is able to be processed.
5. `<JsonParser>` is used to parse fields.

7.5.2.2 @SerializableField

The annotation `@SerializableField` indicates that a Java object field is intended to be serialized to a sub element or attribute of a REST representation. This annotation provides a number of attributes enabling you to customize the marshalling and unmarshalling behavior.



Note: Care must be taken before applying the `<@SerializableField>` annotation to final or static fields. Unexpected behavior may occur when manipulating these final or static fields.

Table 7-2: @SerializableField Attributes

Attribute	Description	Supported Formats	Marshal/Unmarshal	Default
value	Specifies the node name of the serializable type on one field of the class, for example, the name of an element in XML. For more information, see Example I on page 150 .	XML/JSON	in and out	not set
required	Specifies whether a field is required to be non-null for marshalling. When the field is required but its value is null, an exception is thrown during marshalling.	XML/JSON	out	false
override	Specifies whether to override the representation of parent's field with same serializable name.	XML/JSON	in and out	false

Attribute	Description	Supported Formats	Marshal/Unmarshal	Default
xmlWriteTypeRoot	<p>This setting applies to a field of complex type.</p> <p>A complex type refers to a custom type annotated with <code>SerializableType</code>.</p> <p>Specifies whether to write <code>SerializableType.value</code> instead of <code>SerializableField.value</code> as the direct XML child element for a complex type.</p> <ul style="list-style-type: none"> • true - write <code>SerializableType.value</code> as the direct XML child element for a complex type • false - write <code>SerializableField.value</code> as the direct XML child element for a complex type 	XML	in and out	false
defaultImpl	Specifies the default implementation class of a field for unmarshalling when the field implements an interface or extends an abstract class.	XML/JSON	in	DEFAULT.class

Attribute	Description	Supported Formats	Marshal/ Unmarshal	Default
xmlAsAttribute	<p>Specifies whether the field is represented as an attribute in the XML element attribute or a child element.</p> <p>For more information, see Example J on page 150.</p>	XML	in and out	false
 Cau tion <i>Deprec ated</i>	<p>Specifies whether to unwrap the items from a <code>java.lang.List</code> type field as the direct member of the serialized object.</p> <ul style="list-style-type: none"> When being unwrapped, the collection members of this field are moved to an upper level to be direct members of the serialized object in REST representation When not being unwrapping, this field is marshalled and unmarshalled as usual <p>For more information, see Example K on page 151.</p>	XML	in and out	false

Attribute	Description	Supported Formats	Marshal/Unmarshal	Default
 Caution <i>Deprecated</i>	<p>Specifies the XML element name for the value list of a list data type. This attribute takes effect only when the field to serialize is composed of a list of simple data types, such as <code>List<String></code>. If the field is a list of custom classes, such as <code>List<MyType></code>, the attribute does not work and the default value <code>item</code> is used as the element name for the list.</p> <p>For more information, see Example L on page 151.</p>	XML	in and out	item
<code>xmlNS</code>	<p>Specifies the namespace for a certain field in XML representation. If the annotated field is an instance of a certain class that has a different namespace specified, the class namespace overrides this setting.</p> <p>For more information, see Example M on page 152.</p>	XML	out	not set inherited from the parent class

Attribute	Description	Supported Formats	Marshal/ Unmarshal	Default
xmlNSPrefix	<p>Specifies the namespace prefix for a certain field in XML representation. If the annotated field is an instance of a certain class that has a different namespace prefix specified, the class namespace prefix overrides this setting.</p> <p>For more information, see Example M on page 152.</p>	XML	out	not set inherited from the parent class

7.5.2.2.1 Examples

This section lists several examples explaining the detail usage of the attributes in the `@SerializableField` annotation.

➡ Example 7-11: Field Serializable Name

In the following example, the field `vstamp` is serialized as `version-stamp`.

Java Definition	XML	JSON
<pre> @SerializableType ("business-object") public class BusinessObject { private String id; @SerializableField("version-stamp") private Integer vstamp; public BusinessObject(String id, Integer vstamp) { this.id = id; this.vstamp = vstamp; } // customize field name public static void main(String[] args) { BusinessObject bo = new BusinessObject("09ae2d", 6); } } </pre>	<pre> <?xml version='1.0' encoding='UTF-8'?> <business-object> <id>09ae2d</id> <version-stamp>6</version-stamp> </business-object> </pre>	<pre> { "id": "09ae2d", "version-stamp": 6 } </pre>



➡ Example 7-12: Field as XML Attribute

In the following example, the field `vstamp` is serialized as an attribute.

Java Definition	XML	JSON
<pre>@SerializableType ("business-object") public class BusinessObject { private String id; @SerializableField(xmlAsAttribute = true) private Integer vstamp; public BusinessObject(String id, Integer vstamp) { this.id = id; this.vstamp = vstamp; } // field as xml attribute public static void main(String[] args) { BusinessObject bo = new BusinessObject("09ae2d", 6); } }</pre>	<pre><?xml version='1.0' encoding='UTF-8'?> <business-object vstamp="6"> <id>09ae2d</id> </business-object></pre>	<pre>{ "id": "09ae2d", "vstamp": 6 }</pre>



Example 7-13: Field as XML Unwrapped List

In the following example, the list of outEvents are unwrapped.

<pre>@SerializableType ("business-object") public class BusinessObject { private String id; private Integer vstamp; @SerializableField(value = "in-events", xmlListUnwrap = false) private List<Event> inEvents; @SerializableField(value = "out-events", xmlListUnwrap = true) private List<Event> outEvents; public BusinessObject(String id, Integer vstamp, Event[] in, Event[] out) { this.id = id; this.vstamp = vstamp; this.inEvents = in == null ? new ArrayList<Event>() : Arrays.asList(in); this.outEvents = out == null ? new ArrayList<Event>() : Arrays.asList(out); } @SerializableType("event") public static class Event { @SerializableField("eid") private Integer eid; public Event (Integer eid) { this.eid = eid; } } // field as xml unwrapped list public static void main(String[] args) { BusinessObject bo = new BusinessObject("09ae2d", 6, new Event[]{new Event(1), new Event(2)}, new Event[]{new Event(3), new Event(4)}); } }</pre>	<pre><?xml version='1.0' encoding='UTF-8'?> <business-object> <id>09ae2d</id> <vstamp>6</vstamp> <in-events> <event> <eid>1</eid> </event> <event> <eid>2</eid> </event> </in-events> <out-events> <event> <eid>3</eid> </event> <event> <eid>4</eid> </event> </out-events> </business-object></pre>	<pre>{ "id": "09ae2d", "vstamp": 6, "in-events": [{ "eid": 1 }, { "eid": 2 }], "out-events": [{ "eid": 3 }, { "eid": 4 }] }</pre>
---	---	---



Example 7-14: Field List XML Item Name

In the following example, the inEvent list uses the default name item for list items while the outEvent uses out-event.

Java Definition	XML	JSON
<pre>@SerializableType ("business-object") public class BusinessObject { private String id; private Integer vstamp; @SerializableField(value = "in-events") private List<String> inEvents; @SerializableField(value = "out-events", xmlListItemName = "out-event") private List<String> outEvents; public BusinessObject(String id, Integer vstamp, String[] in, String[] out) { this.id = id; this.vstamp = vstamp; this.inEvents = in == null ? new ArrayList<String>() : Arrays.asList(in); this.outEvents = out == null ? new ArrayList<String>() : Arrays.asList(out); } // Field xml list item name public static void main(String[] args) { BusinessObject bo = new BusinessObject("09ae2d", 6, new String[]{"e-1", "e-2"}, new String[]{"e-3", "e-4"}); } }</pre>	<pre><?xml version='1.0' encoding='UTF-8'?> <business-object> <id>09ae2d</id> <vstamp>6</vstamp> <in-events> <item>e-1</item> <item>e-2</item> </in-events> <out-events> <out-event>e-3</out-event> <out-event>e-4</out-event> </out-events> </business-object></pre>	<pre>{ "id": "09ae2d", "vstamp": 6, "in-events": ["e-1", "e-2"], "out-events": ["e-3", "e-4"] }</pre>



Example 7-15: Field XML Namespace

In the following example, XML namespaces and prefixes are added.

Java Definition	XML	JSON
<pre>@SerializableType (value = "business-object", xmlnsPrefix = "dm", xmlns = "http://documentum.emc.com") public class BusinessObject { private String id; @SerializableField(xmlAsAttribute = true, xmlnsPrefix = "xpc", xmlns = "http://xpc.emc.com") private Integer vstamp; @SerializableField(xmlNSPrefix = "d2", xmlns = "http://d2.emc.com") private String config; public BusinessObject(String id, Integer vstamp, String config) { this.id = id; this.vstamp = vstamp; this.config = config; } // Field xml namespace public static void main(String[] args) { BusinessObject bo = new BusinessObject("09ae2d", 6, "sample"); } }</pre>	<pre><?xml version='1.0' encoding='UTF-8'?> <business-object xmlns="http://documentum.emc.com" xmlns:dm="http://xpc.emc.com" xmlns:d2="http://d2.emc.com" xmlns:xpc="http://d2.emc.com" vstamp="6"> <id>09ae2d</id> <d2:config>sample</d2:config> </business-object></pre>	<pre>{ "id": "09ae2d", "vstamp": 6, "config": "sample" }</pre>



7.5.2.3 @SerializableField4XmlList

This annotation provides dedicated support for `<java.util.List>` in XML. It uses existing List related attributes from `@SerializableField` and introduces new annotations. The `@SerializableField4XmlList` annotation can only be used with an XML List.

Table 7-3: @SerializableField4XmlList Attributes

Attribute	Description	Supported Formats	Default
boolean unwrap()	Specifies whether to unwrap the list or array items of this field as the direct members of the serializable type. Migrated from @SerializableField.xmlListUnwrap.	XML	false
boolean asPropBag()	Specifies the list items to be marshalled as a property bag or not.	XML	false
String itemName()	Specifies the list item element name within a {@link java.util.List} data type field. Migrated from @SerializableField.xmlListItemName.	XML	"item"



Note: <java.util.List> in JSON is always marshalled as a JSON array

7.5.2.4 @SerializableField4XmlMap

The `@SerializableField4XmlMap` annotation has been added to provide dedicated support for `<java.util.List>` with XML. This annotation can only be used with an XML map.

Table 7-4: @SerializableField4XmlMap Attributes

Attribute	Description	Supported Formats	Default
boolean asPropBag()	Specifies whether to serialize the key / value pair as a property bag or not.	XML	false
String propBagEleName()	Specifies the XML element name for each entry of {@link java.util.Map} data type field.	XML	"entry"
String propBagKeyName()	Specified the XML attribute name for each key of {@link java.util.Map} data type field.	XML	"key"



Note: When using JSON, <java.util.Map> is always marshalled as a JSON key-value object pair

7.5.2.5 Out-of-box Annotated Models

Foundation REST API provides a set of core models that are annotated with the REST annotations out of the box. You can use them directly as the model of custom resources or extend them with additional fields.

- *AtomFeed*
- *RestError*
- *Repository*
- *PersistentObject*
- Other model types in the Java package `com.emc.documentum.rest.model`

7.5.2.6 Deprecations

The following annotations have been deprecated:

- `@SerializableEntry`

Java Map support has been added since version 7.3 of Foundation REST API. Therefore, `@SerializableEntry` has become redundant and is marked as `@Deprecated`. General key-value support is only available for Map objects.

- `@SerializableField#xmlListUnwrap` and `@SerializableField#xmlListItemName`

The annotation `@SerializableField` had attributes for XML specific support. Moving forward, all the functionality of `@SerializableField` has been incorporated into `@SerializableField4XmlList`.



Caution

Legacy Usage

Extended Foundation REST API that use `@SerializableEntry`, `@SerializableField.xmlListUnwrap` and `@SerializableField.xmlListItemName` can continue to use them moving forward. However, enhancements to these attributes will not be added.

7.5.2.7 Additional Examples

This section lists several examples explaining the usage of REST annotations. Most of the examples use the out-of-box annotated models.

➡ Example 7-16: Core Document Object

The following example illustrates how out-of-box annotated core document objects are serialized:

Java Definition	XML	JSON
<pre>DocumentObject doc = new DocumentObject(); doc.setType("de_document"); doc.setUriDef("http://localhost:8080/types/de_document"); doc.addAttribute(new AttributeString("object_name", "Overview.pdf")); doc.addAttribute(new AttributeListString>("authors", Arrays.asList("Bob", "Alice"))); doc.getLinks().add(new Link("self", "http://localhost:8080/documents/09acd2")); doc.getLinks().add(new Link("edit", "http://localhost:8080/documents/09acd2"));</pre>	<pre><?xml version='1.0' encoding='UTF-8'?> <document xmlns="http://identifiers.emc.com/vocab/documentum" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="de_document" definition="http://localhost:8080/types/de_document"> <properties> <object_name>Overview.pdf</object_name> <authors> <item>Bob</item> <item>Alice</item> </authors> </properties> <links> <link rel="self" href="http://localhost:8080/documents/09acd2"/> <link rel="edit" href="http://localhost:8080/documents/09acd2"/> </links> </document></pre>	<pre>{ "name": "document", "type": "de_document", "definition": "http://localhost:8080/types/de_document", "properties": { "object_name": "Overview.pdf", "authors": ["Bob", "Alice"] }, "links": [{ "rel": "self", "href": "http://localhost:8080/documents/09acd2" }, { "rel": "edit", "href": "http://localhost:8080/documents/09acd2" }] }</pre>



Example 7-17: Core Repository Object

The following example illustrates how out-of-box annotated core repository objects are serialized:

Java Definition	XML	JSON
<pre>Repository repo = new Repository(); repo.setId(15); repo.setName("acme"); repo.setDescription("ACME Company"); Server svr1 = new Server(); svr1.setName("CS71SH"); svr1.setDocbroker("localhost"); svr1.setHost("CS71SH"); svr1.setVersion("7.1.0010.0158 Win64.SQLServer"); repo.get_servers().add(svr1); repo.getLinks().add(new Link("self", "http://localhost:8080/repositories/acme")); repo.getLinks().add(new Link("cabinets", "http://localhost:8080/repositories/acme/cabinets"));</pre>	<pre><?xml version='1.0' encoding='UTF-8'?> <repository xmlns="http://identifiers.emc.com/vocab/documentum"> <id>15</id> <name>acme</name> <description>ACME Company</description> <servers> <server> <name>CS71SH</name> <host>CS71SH</host> <version>7.1.0010.0158 Win64.SQLServer</version> <docbroker>localhost</docbroker> </server> </servers> <links> <link rel="self" href="http://localhost:8080/repositories/acme"/> <link rel="cabinets" href="http://localhost:8080/repositories/acme/cabinets"/> </links> </repository></pre>	<pre>{ "id": 15, "name": "acme", "description": "ACME Company", "servers": [{ "name": "CS71SH", "host": "CS71SH", "version": "7.1.0010.0158 Win64.SQLServer", "docbroker": "localhost" }], "links": [{ "rel": "self", "href": "http://localhost:8080/repositories/acme" }, { "rel": "cabinets", "href": "http://localhost:8080/repositories/acme/cabinets" }] }</pre>



Example 7-18: Core Error Object

The following example illustrates how out-of-box annotated core error objects are serialized:

Java Definition	XML	JSON
<pre>RestError error = new RestError(); error.setCode("E_UNDEFINED_TYPE"); error.setStatus(400); error.setMessage("Undefined type is specified: dm_unknown"); error.addDetail("[E_BAD_TYPE] bad type 'dm_unknown' is specified for IDFSobject"); error.addDetail("[E_DFC_OPERATION_ERROR] DFC operation error");</pre>	<pre><?xml version="1.0" encoding="UTF-8"?> <error xmlns="http://identifiers.emc.com/vocab/documentum"> <status>400</status> <code>E_UNDEFINED_TYPE</code> <message>Undefined type is specified: dm_unknown</message> <details>[E_BAD_TYPE] bad type 'dm_unknown' is specified for IDFSobject;[E_DFC_OPERATION_ERROR] DFC operation error </details> </error></pre>	<pre>{ "status": 400, "code": "E_UNDEFINED_TYPE", "message": "Undefined type is specified: dm_unknown", "details": "[E_BAD_TYPE] bad type 'dm_unknown' is specified for IDFSobject;[E_DFC_OPERATION_ERROR] DFC operation error" }</pre>



Example 7-19: Core Atom Feed With Src Entry

The following example illustrates how out-of-box annotated atom feed objects with `src` entries are serialized:



Example 7-20: Core Atom Feed With Inline Entry

The following example illustrates how out-of-box annotated atom feed objects with embedded entries are serialized:



7.5.3 Java Primitive Types Support

Foundation REST API marshalling framework supports the following Java primitive types together with their wrapper classes:

Table 7-5: Supported Primitive Types and Wrapper Classes

Primitive Type	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	boolean

 **Note:** The primitive type `char` is not supported in the marshalling framework. Instead, the framework supports `String` to cover all the functionalities that `char` provides.

The following diagram illustrates the mapping between Java primitive types and XML or JSON data types:

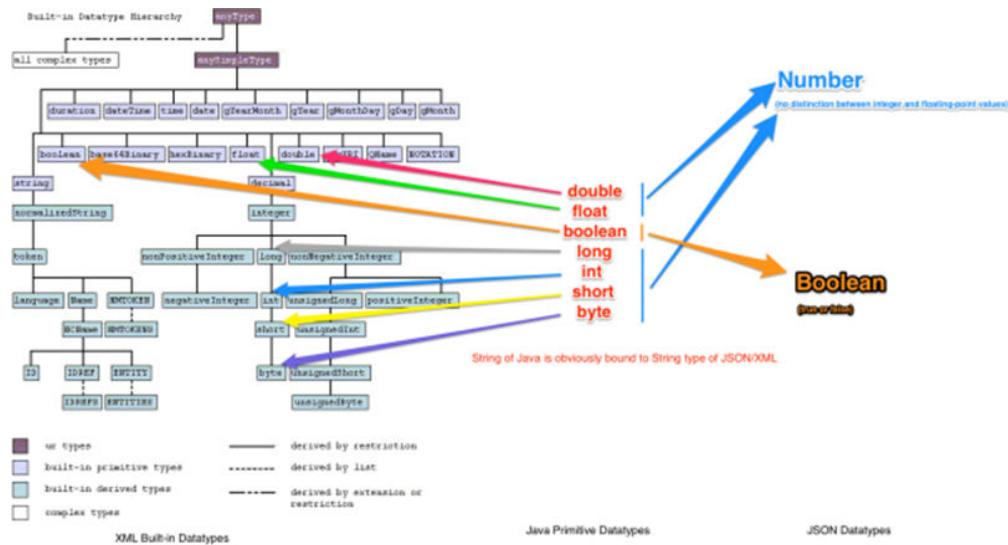


Figure 7-6: Java Primitive Types and XML/JSON Data Types Mapping

In JSON representation, the type `Number` does not distinguish between integer and float-point values, which means:

- During marshalling, following data types (and their wrapper classes) are represented in JSON by the Number data type:
 - byte
 - short
 - int
 - long
 - float
 - double
- During unmarshalling, a JSON Number value is converted based on the type information of the Java class model.

In this process, if a JSON Number value contains a higher precision than what the Java class model defines, the unmarshalling fails.

For example, if JSON representation contains "score": 9.5, and the model has a field int score, the unmarshalling fails. In this case, modify the value of score to an integer, such as 9, or modify the type of score in the model to a type with a higher precision, such as float.

7.5.4 Java Interface, Abstract and Generic Types Support

7.5.4.1 Marshalling

For serialization (marshalling), the marshalling of a Java instance depends on the ability to serialize the data type of the runtime instance. Regardless of interface, abstract, or generic declaration of a class data field, the runtime class instance can be serialized as long as the instance's field types at runtime are serializable. For example, a concrete data type can be annotated by `@SerializableType`.

Here is a code sample that shows you how to do this:

Example 7-21: Annotating a Concrete Data Type

```
// Skipping all methods and constructors

// The root data type with generic field 'organization'
@SerializableType("bog")
public class BusinessObjectGeneric<T> {
    private T organization; // a generic field
}

// One organization implementation
@SerializableType("institution")
public class Institution {
    private String name;
    private Role role; // an abstract field
}

// Role interface
public interface Role { // interface
    String getName();
}
```

```
// One role implementation
@SerializableType("go")
public class GovernmentalOrg implements Role { // implementation of interface
    private String name;
    private int privilege;
}

// The other role implementation
@SerializableType("ngo")
public class NonGovernmentalOrg implements Role { // implementation of interface
    private String name;
    private String description;
}
```



Here's a code sample that shows you how to serialize a complex type:

► Example 7-22: An Instance of BusinessObjectGeneric 1

```
BusinessObjectGeneric<Institution> origin =
    new BusinessObjectGeneric<Institution>(
        new Institution(
            "knowledge",
            new NonGovernmentalOrg(
                "wikimedia",
                "ubiquitous online encyclopaedia"
            )
        )
    );
```



► Example 7-23: XML Representation for Sample 1

```
<bog>
    <organization>
        <name>knowledge</name>
        <role>
            <name>wikimedia</name>
            <description>ubiquitous online encyclopaedia</description>
        </role>
    </organization>
</bog>
```



► Example 7-24: JSON Representation for Sample 1

```
{
    "organization": {
        "name": "knowledge",
        "role": {
            "name": "wikimedia",
            "description": "ubiquitous online encyclopaedia"
        }
    }
}
```



➡ Example 7-25: An Instance of BusinessObjectGeneric 2

```
BusinessObjectGeneric<Institution> origin =
    new BusinessObjectGeneric<Institution>(
        new Institution(
            "funding",
            new GovernmentalOrg(
                "imf",
                8
            )
        );
    )
```



➡ Example 7-26: XML Representation for Sample 2

```
<bog>
  <organization>
    <name>funding</name>
    <role>
      <name>imf</name>
      <priviledges>8</priviledges>
    </role>
  </organization>
</bog>
```



➡ Example 7-27: JSON Representation for Sample 2

```
{
  "organization": {
    "name": "funding",
    "role": {
      "name": "imf",
      "priviledges": 8
    }
  }
}
```



Caution

Unmarshalling the XML and JSON representations will not succeed because:

- In the XML representation, the `<organization>` tag did not interpret which implementation class to use during unmarshalling
- In the JSON representation, the key `<"organization">` does not contain the class implementation information

7.5.4.2 Unmarshalling

You must add annotation metadata to the class definition to deserialize the data class. Adding this data type makes the type writable in XML or JSON.

- *For XML*

For XML, the serializable fields that have been declared as generic, interface, or abstract must set `@SerializableField#xmlWriteRoot = true`. The concrete data type for the field is written into XML as the XML element name. Therefore, it can be resolved during the deserialization process.



Caution

To deserialize the generic, interface or abstract field in XML, the data class cannot have more than one field that contains non-concrete data types.

Here's a code sample that demonstrates how to do this:

```
// xmlWriteRoot=true
@SerializableField(xmlWriteTypeRoot = true)
private T organization;

@SerializableField(xmlWriteTypeRoot = true)
private Role role;
```

- *For JSON*

For JSON, the serializable fields that have been declared as generic, interface or abstract must set `@SerializableField#jsonWriteRoot = true`. The concrete data type for the field is written into XML as the XML element name. Therefore, it can be resolved during the deserialization process.

Here's a code sample that demonstrates how to do this:

```
// jsonWriteRoot=true
@SerializableType(value = "institution", jsonWriteRootAsField = true, jsonRootField =
= "type")
public static class Institution {...}

@SerializableType(value = "go", jsonWriteRootAsField = true, jsonRootField = "type")
public static class GovernmentalOrg implements Role {...}

@SerializableType(value = "ngo", jsonWriteRootAsField = true, jsonRootField =
= "type")
public static class NonGovernmentalOrg implements Role {...}
```

The XML and JSON contain additional type information that allows it be deserialized. Here's the same code with all the changes:

Example 7-28: An Instance of a Writable BusinessObjectGeneric 1

```
BusinessObjectGeneric<Institution> origin =
    new BusinessObjectGeneric<Institution>(
        new Institution(
            "knowledge",
            new NonGovernmentalOrg(
```

```
        "wikimedia",
        "ubiquitous online encyclopaedia"
    )
)
);
```



➤ Example 7-29: XML Representation for a Writable Sample 1

```
<bog>
  <institution>
    <name>knowledge</name>
    <ngo>
      <name>wikimedia</name>
      <description>ubiquitous online encyclopaedia</description>
    </ngo>
  </institution>
</bog>
```



Note: The serializable name `institution` that is used for the actual data type `<Institution>` is written as XML instead of using the generic field name `<organization>`.

The serializable name `ngo` that is used for the actual data type `<NonGovernmentalOrg>` is written as XML instead of using the generic field name `<role>`.

➤ Example 7-30: JSON Representation for a Writable Sample 1

```
{
  "organization": {
    "type": "institution",
    "name": "knowledge",
    "role": {
      "type": "ngo",
      "name": "wikimedia",
      "description": "ubiquitous online encyclopaedia"
    }
  }
}
```



Note: The root information type `:institution` that is used for the actual data type `<Institution>` is written to the JSON object of `<organization>`.

The root information type `: ngo` that is used for the actual data type `<NonGovernmentalOrg>` is written to the JSON object of `<role>`.

➤ Example 7-31: An Instance of a Writable BusinessObjectGeneric 2

```
BusinessObjectGeneric<Institution> origin =
  new BusinessObjectGeneric<Institution>(
    new Institution(
      "funding",
      new GovernmentalOrg(
        "imf",
        8
```

```

        )
    );
}

```



Note: The serializable name `institution` that is used for the actual data type `<Institution>` is written as XML instead of using the generic field name `<organization>`.

The serializable name `go` that is used for the actual data type `<GovernmentalOrg>` is written as XML instead of using the generic field name `<role>`.



Example 7-32: XML Representation for a Writable Sample 2

```

<bog>
  <institution>
    <name>funding</name>
    <go>
      <name>wikimedia</name>
      <priviledges>8</description>
    </go>
  </institution>
</bog>

```



Example 7-33: JSON Representation for a Writable Sample 2

```

{
  "organization": {
    "type": "institution",
    "name": "funding",
    "role": {
      "type": "go",
      "name": "imf",
      "priviledges": 8
    }
  }
}

```



Note: The root information type: `institution` that is used for the actual data type `<Institution>` is written to the JSON object of `<organization>`.

The root information type: `go` that is used for the actual data type `<GovernmentalOrg>` is written to the JSON object of `<role>`.



Example 7-34: Complete Class Definition

```

/*
 * Copyright (c) 2016. OpenText Corporation. All Rights Reserved.
 */
package com.emc.documentum.rest.mock;

import org.apache.commons.lang.builder.EqualsBuilder;
import com.emc.documentum.rest.binding.SerializableField;
import com.emc.documentum.rest.binding.SerializableType;

```

```
@SerializableType("bog")
public class BusinessObjectGeneric <T> {

    @SerializableField(xmlWriteTypeRoot = true)
    private T organization;

    public BusinessObjectGeneric() {}

    public BusinessObjectGeneric(T organization) {
        this.organization = organization;
    }

    public T getOrganization() {
        return this.organization;
    }

    @Override
    public boolean equals(Object other) {
        return EqualsBuilder.reflectionEquals(this, other);
    }

    @SerializableType(value = "institution",
                      jsonWriteRootAsField = true,
                      jsonRootField = "type")
    public static class Institution {
        private String name;
        @SerializableField(xmlWriteTypeRoot = true)
        private Role role;

        public Institution() {}

        public Institution(String name, Role role) {
            this.name = name;
            this.role = role;
        }

        public String getName() {
            return name;
        }

        public Role getRole() {
            return role;
        }

        @Override
        public boolean equals(Object other) {
            return EqualsBuilder.reflectionEquals(this, other);
        }
    }

    @SerializableType(value = "company",
                      jsonWriteRootAsField = true,
                      jsonRootField = "type")
    public static class Company {
        private String name;
        @SerializableField(xmlWriteTypeRoot = true)
        private Industry industry;

        public Company() {}

        public Company(String name, Industry industry) {
            this.name = name;
            this.industry = industry;
        }

        public String getName() {
            return name;
        }
    }
}
```

```

        public Industry getIndustry() {
            return industry;
        }

        @Override
        public boolean equals(Object other) {
            return EqualsBuilder.reflectionEquals(this, other);
        }
    }

    public static interface Role {
        String getName();
    }

    @SerializableType(value = "go",
                      jsonWriteRootAsField = true,
                      jsonRootField = "type")
    public static class GovernmentalOrg implements Role {
        private String name;
        private int privilege;

        public GovernmentalOrg() {}

        public GovernmentalOrg(String name, int privilege) {
            this.name = name;
            this.privilege = privilege;
        }

        @Override public String getName() {
            return name;
        }

        public int getPrivilege() {
            return privilege;
        }

        @Override
        public boolean equals(Object other) {
            return EqualsBuilder.reflectionEquals(this, other);
        }
    }

    @SerializableType(value = "ngo",
                      jsonWriteRootAsField = true,
                      jsonRootField = "type")
    public static class NonGovernmentalOrg implements Role {
        private String name;
        private String description;

        public NonGovernmentalOrg() {}

        public NonGovernmentalOrg(String name, String description) {
            this.name = name;
            this.description = description;
        }

        @Override public String getName() {
            return name;
        }

        public String getDescription() {
            return description;
        }

        @Override
        public boolean equals(Object other) {
            return EqualsBuilder.reflectionEquals(this, other);
        }
    }
}

```

```
@SerializableType
public static abstract class Industry {
    @SerializableField("country")
    public String countryCode;

    public String getCountryCode() {
        return countryCode;
    }

    @Override
    public boolean equals(Object other) {
        return EqualsBuilder.reflectionEquals(this, other);
    }
}

@SerializableType(value = "it",
                  jsonWriteRootAsField = true,
                  jsonRootField = "type")
public static class InformationTechnology extends Industry {
    private String scale;

    public InformationTechnology() {}

    public InformationTechnology(String scale, String countryCode) {
        super.countryCode = countryCode;
        this.scale = scale;
    }

    public String getScale() {
        return scale;
    }

    @Override
    public boolean equals(Object other) {
        return EqualsBuilder.reflectionEquals(this, other);
    }
}

@SerializableType(value = "pharm",
                  jsonWriteRootAsField = true,
                  jsonRootField = "type")
public static class Pharmaceuticals extends Industry {
    private String field;

    public Pharmaceuticals() {}

    public Pharmaceuticals(String field, String countryCode) {
        super.countryCode = countryCode;
        this.field = field;
    }

    public String getField() {
        return field;
    }

    @Override
    public boolean equals(Object other) {
        return EqualsBuilder.reflectionEquals(this, other);
    }
}
```



7.5.5 Java Collection and Array Support

7.5.5.1 Array Serialization Support

For all supported Java primitive types in the REST marshalling framework, the corresponding array form is also supported. For example, the following code snippet serializes an array of *boolean* values.

Java Code

```
@SerializableField
private boolean[] feedCustomizedBooleanArray = {true, false, false};
```

XML Representation

```
<feedCustomizedBooleanArray>
<item>true</item>
<item>false</item>
<item>false</item>
</feedCustomizedBooleanArray>
```

JSON Representation

```
"feedCustomizedBooleanArray": [
  true,
  false,
  false
],
```

Moreover, the framework supports the array form of any custom type instances. For an instance of a custom type, the return of its *toString* method is used as the value when being marshaled to XML or JSON.

Custom type Color

```
public enum Color {
    RED(255,0,0),
    BLUE(0,0,255),
    BLACK(0,0,0),
    YELLOW(255,255,0),
    GREEN(0,255,0);

    private Color(int redValue,int greenValue,int blueValue){
        this.redValue=redValue;
        this.greenValue=greenValue;
        this.blueValue=blueValue;
    }

    public String toString(){
        return super.toString()+"("+redValue+","+greenValue+","+blueValue+)";
    }

    private int redValue;
    private int greenValue;
    private int blueValue;
}
```

Java Code

```
@SerializableField(xmlNS = "http://ns.customization.com/", xmlNSPrefix = "customization")
private Color[] colorArray = {YELLOW, BLUE};
```

XML Representation

```
<customization colorArray
  xmlns:customization="http://ns.customization.com/">
<customization:item> YELLOW(255,255,0) </customization:item>
<customization:item> BLUE(0,0,255)</customization:item>
</customization:colorArray >
```

JSON Representation

```
" colorArray " : [
  " YELLOW(255,255,0) " ,
  " BLUE(0,0,255)" ,
]
```

7.5.5.2 Java Collection Serialization Support

The marshalling framework supports `List` and its subtypes. In JSON representation, a `List` is bound to an Array. In XML representation, each element in a `List` is a sub-element of the element representing the `List`.

For more information, see [Example K on page 151](#) for details.

7.5.5.3 Supported Data Types

The following data types are supported as the parameters of a `List` or `Array`:

- Declared native data types such as `List<String>`, `List<Date>`, `List<Integer>`, `List<Double>`, `List<boolean>`
- Declared enum data types such as `List<EnumUserTypes>`, `List<EnumDocTypes>`
- Declared complex data types with the `<@SerializableType>` annotation such as `List<AtomFeed>`, `List<Link>`, `List<PersistentObject>`
- Interface such as `List<Folder>`, `List<User>`
- Abstract class such as `List<AbstractFolder>`, `List<AbstractDocument>`
- Generic or wild card object data types such as `List<Object>`, `List<?>`, `List<T>`



Caution

Runtime check of parameterized data type

Whether or not instances of *Interface*, *Abstract class*, and *Generic or wild card* can be marshalled is determined at runtime by the data type of the instance.

7.5.5.4 Exclusions

The following parameterized data types are not supported:

- List or Array in a List
For example `List<List<String>>, List<String[]>`
- Map data type in a List `List<Map<String, String>>`
- Any other data type



Caution

List item order

Items in a list are marshalled according to their order within the list

java.util.Collection<> is not supported

The generic collection data type `java.util.Collection<>` is not supported.
The List data type must be an Array. `java.util.List`, or a sub type of `java.util.List`

7.5.5.5 Marshalling with XML

The marshalling of an XML List or Array is more complex than marshalling a JSON List or Array because of the following:

- The XML fields can have separate namespaces.
- The XML fields for a List or Array can be wrapped or unwrapped.
- The XML fields for a List or Array of complex data types can be serialized as property bags or standalone XML objects.

Example 7-35: Native Data Types as XML Elements

Java Code

```
@SerializableType("boa")
public class BusinessObjectArchive {
    private List<String> keywords;

}
BusinessObjectArchive boa = new BusinessObjectArchive();
boa.getKeywords().add("2015");
boa.getKeywords().add("bedrock");
```

XML Code

```
<boa>
    <keywords>
        <item>2015</item>
        <item>bedrock</item>
    </keywords>
</boa>
```

**Example 7-36: Array of Enums as XML Elements***Java Code*

```
@SerializableType("boa")
public class BusinessObjectArchive {
    private LABEL[] labels;

    public static enum LABEL {
        DEV, PRODUCT, SUPPORT
    }
    ...
}
BusinessObjectArchive boa = new BusinessObjectArchive();
boa.setLabels(BusinessObjectArchive.LABEL.DEV,
    BusinessObjectArchive.LABEL.SUPPORT);
```

XML Code

```
<boa>
    <labels>
        <item>DEV</item>
        <item>SUPPORT</item>
    </labels>
</boa>
```

**Example 7-37: List of Complex Data Type as XML Elements***Java Code*

```
@SerializableType("boa")
public class BusinessObjectArchive {
    private List<Role> roles;

    public static interface Role {}

    @SerializableType(value = "admin-role")
    public static class AdminRole implements Role {
        private String name;
        private int privilege;
    }

    @SerializableType(value = "consumer-role")
    public static class ConsumerRole implements Role {
        private String name;
        private String description;
    }
    ...
}
BusinessObjectArchive boa = new BusinessObjectArchive();
boa.getRoles().add(new BusinessObjectArchive.AdminRole("admin", 16));
boa.getRoles().add(new BusinessObjectArchive.ConsumerRole("ios", "iOS mobile device"));
```

XML Code

```
<boa>
    <roles>
        <admin-role>
            <name>admin</name>
            <privilege>16</privilege>
        </admin-role>
    </roles>
</boa>
```

```

        </admin-role>
        <consumer-role>
            <name>ios</name>
            <description>iOS mobile device</description>
        </consumer-role>
    </roles>
</boa>
```



Example 7-38: List of Complex Data Type as XML property-bag Elements

Java Code

```

@SerializableType("boa")
public class BusinessObjectArchive {
    @SerializableField4XmlList(asPropBag = true, itemName="role")
    private List<Role> roles;

    public static interface Role {}

    @SerializableType(value = "admin-role")
    public static class AdminRole implements Role {
        private String name;
        private int privilege;
    }

    @SerializableType(value = "consumer-role")
    public static class ConsumerRole implements Role {
        private String name;
        private String description;
    }
    ...
}
BusinessObjectArchive boa = new BusinessObjectArchive();
boa.getRoles().add(new BusinessObjectArchive.AdminRole("admin", 16));
boa.getRoles().add(new BusinessObjectArchive
    .ConsumerRole("ios", "iOS mobile device"));
```

XML Code

```

<boa>
    <roles>
        <role>
            <name>admin</name>
            <privilege>16</privilege>
        </role>
        <role>
            <name>ios</name>
            <description>iOS mobile device</description>
        </role>
    </roles>
</boa>
```



Example 7-39: List of Native Data Type as XML Unwrapped Elements

Java Code

```

@SerializableType("boa")
public class BusinessObjectArchive {
    @SerializableField4XmlList(unwrap = true, itemName="keyword")
    private List<String> keywords;
}
```

```
BusinessObjectArchive boa = new BusinessObjectArchive();
boa.getKeywords().add("2015");
boa.getKeywords().add("bedrock");
```

XML Code

```
<boa>
    <keyword>2015</keyword>
    <keyword>bedrock</keyword>
</boa>
```



Example 7-40: List of Complex Data Type as XML Elements with namespaces

Java Code

```
@SerializableType("boa")
public class BusinessObjectArchive {
    @SerializableField4Xmllist(asPropBag = false)
    private List<Role> roles;

    public static interface Role {}

    @SerializableType(value = "admin-role", xmlns = "http://acme.org", xmlNSPrefix = "am")
    public static class AdminRole implements Role {
        private String name;
        private int privilege;
    }

    @SerializableType(value = "consumer-role", xmlns = "http://d2.org",
                      xmlNSPrefix = "d2")
    public static class ConsumerRole implements Role {
        private String name;
        private String description;
    }
    ...
}
BusinessObjectArchive boa = new BusinessObjectArchive();
boa.getRoles().add(new BusinessObjectArchive.AdminRole("admin", 16));
boa.getRoles().add(new BusinessObjectArchive.ConsumerRole("ios", "iOS mobile device"));
```

XML Code

```
<boa>
    <roles>
        <am:admin-role xmlns:am="http://acme.org">
            <am:name>admin</am:name>
            <am:privilege>16</am:privilege>
        </am:admin-role>
        <d2:consumer-role xmlns:d2="http://d2.org">
            <d2:name>ios</d2:name>
            <d2:description>iOS mobile device</d2:description>
        </d2:consumer-role>
    </roles>
</boa>
```



► **Example 7-41: List of complex Data Type as XML unwrapped property—
bag Elements**

Java Code

```
@SerializableType("boa")
public class BusinessObjectArchive {
    @SerializableField4XmlList(unwrapped=true, asPropBag = true, itemName="role")
    private List<Role> roles;

    public static interface Role {}

    @SerializableType(value = "admin-role")
    public static class AdminRole implements Role {
        private String name;
        private int privilege;
    }

    @SerializableType(value = "consumer-role")
    public static class ConsumerRole implements Role {
        private String name;
        private String description;
    }
    ...
}
BusinessObjectArchive boa = new BusinessObjectArchive();
boa.getRoles().add(new BusinessObjectArchive.AdminRole("admin", 16));
boa.getRoles().add(new BusinessObjectArchive
    .ConsumerRole("ios", "iOS mobile device"));
```

XML Code

```
<boa>
    <role>
        <name>admin</name>
        <privilege>16</privilege>
    </role>
    <role>
        <name>ios</name>
        <description>iOS mobile device</description>
    </role>
</boa>
```



7.5.5.6 Marshalling with JSON

The data types mentioned in the preceding topic can be serialized (marshalled) into a JSON array. The following code samples show you how to serialize different data types:

► **Example 7-42: Native Data Types as a JSON array**

Java Code

```
@SerializableType("boa")
public class BusinessObjectArchive {
    private List<String> keywords;
    ...
}
BusinessObjectArchive boa = new BusinessObjectArchive();
boa.getKeywords().add("2016");
boa.getKeywords().add("Bedrock");
```

JSON Code

```
{  
    "keywords": [ "2016", "Bedrock" ]  
}
```

**Example 7-43: Enum Array as a JSON array***Java Code*

```
@SerializableType("boa")  
public class BusinessObjectArchive {  
    private LABEL[] labels;  
  
    public static enum LABEL {  
        DEV, PRODUCT, SUPPORT  
    }  
    ...  
}  
BusinessObjectArchive boa = new BusinessObjectArchive();  
boa.setLabels(BusinessObjectArchive.LABEL.DEV,  
    BusinessObjectArchive.LABEL.SUPPORT);
```

JSON Code

```
{  
    "labels": [ "DEV", "SUPPORT" ]  
}
```

**Example 7-44: Interface as a JSON array***Java Code*

```
@SerializableType("boa")  
public class BusinessObjectArchive {  
    private List<Role> roles;  
  
    public static interface Role {}  
  
    @SerializableType(value = "admin-role")  
    public static class AdminRole implements Role {  
        private String name;  
        private int privilege;  
    }  
  
    @SerializableType(value = "consumer-role")  
    public static class ConsumerRole implements Role {  
        private String name;  
        private String description;  
    }  
    ...  
}  
BusinessObjectArchive boa = new BusinessObjectArchive();  
boa.getRoles().add(new BusinessObjectArchive.AdminRole("admin", 16));  
boa.getRoles().add(new BusinessObjectArchive.ConsumerRole("ios", "iOS mobile device"));
```

JSON Code

```
{  
    "roles": [  
    ]
```

```

        {"name":"admin","privilege":16},
        {"name":"ios","description":"iOS mobile device"}
    ]
}

```



➤ Example 7-45: Abstract class as a JSON array

Java Code

```

@SerializableType("boa")
public class BusinessObjectArchive {
    private List<Industry> industries;

    @SerializableType
    public static abstract class Industry {
        protected String industry;
    }

    @SerializableType(value = "healthcare")
    public static class Healthcare extends Industry {
        @SerializableField("cc")
        private int countryCode;
    }

    @SerializableType(value = "energy")
    public static class Energy extends Industry {
        private String type;
    }
    ...
}
BusinessObjectArchive boa = new BusinessObjectArchive();
boa.getIndustries().add(new BusinessObjectArchive.Healthcare("hc", 1));
boa.getIndustries().add(new BusinessObjectArchive.Energy("en", "nuclear"));

```

JSON Code

```
{
    "industries":[
        {"cc":1,"industry":"hc"},
        {"type":"nuclear","industry":"en"}
    ]
}
```



7.5.5.7 Unmarshalling with XML and JSON

A List or Array can be deserialized using JSON or XML messages. However, there are certain constraints with respect to the class definition.

The following field types can be deserialized without constraints:

- *Declared Native data types*

`List<String>, List<Date>, List<Integer>, List<Double>, List<boolean>`

- *Declared Enum data types*

`List<EnumUserTypes>, List<EnumDocTypes>`

- *Declared complex data types using the `@SerializableType` annotation*

```
List<AtomFeed>, List<Link>, List<PersistentObject>
```

The following data types can only be deserialized using annotation attributes for XML or JSON:

- *Interface*

```
List<IFolder>, List<IUser>
```

- *Abstract class*

```
List<AbstractFolder>, List<AbstractDocument>
```

- *Generic or wild card data types*

```
List<Object>, List<?>, List<T>
```

7.5.5.7.1 Unmarshalling with XML

The following data types can be deserialized (unmarshalled) using XML:

- *Native data types*
- *Enum data type*
- *Complex data types*
- *Native data types as XML unwrapped elements*
- *Complex data types as XML elements with namespaces*

Messages of the following data types cannot be deserialized (unmarshalled) using XML because their data type is not determined from their declared data type or XML message:

- *Complex data type as XML property-bag elements*
- *Complex data type as XML elements with namespaces*

To make a List of Interface, Abstract, or Wild card items deserializable, their fields must be marked as *wrapped* and *non-property bag* by setting the `@SerializableField4XmlList.asPropBag` and `@SerializableField4XmlList.unwrap` annotations.

Here's a code sample that shows you how to do that:

```
@SerializableField4XmlList.asPropBag=false  
@SerializableField4XmlList.unwrap=false
```

This code sample shows you how to resolve the preceding limitation so you can deserialize a complex data type as XML elements:

➡ Example 7-46: List of complex Data Type as XML Elements

Java Code

```
@SerializableType("boa")  
public class BusinessObjectArchive {
```

```

private List<Role> roles;

public static interface Role {}

@SerializableType(value = "admin-role")
public static class AdminRole implements Role {
    private String name;
    private int privilege;
}

@SerializableType(value = "consumer-role")
public static class ConsumerRole implements Role {
    private String name;
    private String description;
}

...
}

BusinessObjectArchive boa = new BusinessObjectArchive();
boa.getRoles().add(new BusinessObjectArchive.AdminRole("admin", 16));
boa.getRoles().add(new BusinessObjectArchive
    .ConsumerRole("ios", "iOS mobile device"));

```

XML Code

```

<boa>
    <roles>
        <admin-role>
            <name>admin</name>
            <privilege>16</privilege>
        </admin-role>
        <consumer-role>
            <name>ios</name>
            <description>iOS mobile device</description>
        </consumer-role>
    </roles>
</boa>

```



This code sample shows you how to resolve the preceding limitation so you can deserialize a complex data type as XML elements with namespaces:

Example 7-47: List of complex Data Type as XML Elements with namespaces

Java Code

```

@SerializableType("boa")
public class BusinessObjectArchive {
    @SerializableField4XmlList(asPropBag = false)
    private List<Role> roles;

    public static interface Role {}

    @SerializableType(value = "admin-role", xmlNS = "http://acme.org", xmlNSPrefix =
    "am")
    public static class AdminRole implements Role {
        private String name;
        private int privilege;
    }

    @SerializableType(value = "consumer-role", xmlNS = "http://d2.org",
                      xmlNSPrefix = "d2")
    public static class ConsumerRole implements Role {
        private String name;
        private String description;
    }
}

```

```

        }
    ...
}
BusinessObjectArchive boa = new BusinessObjectArchive();
boa.getRoles().add(new BusinessObjectArchive.AdminRole("admin", 16));
boa.getRoles().add(new BusinessObjectArchive
    .ConsumerRole("ios", "iOS mobile device"));

```

XML Code

```

<boa>
    <roles>
        <am:admin-role xmlns:am="http://acme.org">
            <am:name>admin</am:name>
            <am:privilege>16</am:privilege>
        </am:admin-role>
        <d2:consumer-role xmlns:d2="http://d2.org">
            <d2:name>ios</d2:name>
            <d2:description>iOS mobile device</d2:description>
        </d2:consumer-role>
    </roles>
</boa>

```

**7.5.5.7.2 Unmarshalling with JSON**

The following data types can be deserialized (unmarshalled) using JSON:

- *Native data types*
- *Enum data type*

Messages of the following data types cannot be serialized (unmarshalled) using JSON because their data type is not determined from their declared data type or the JSON message:

- *Interfaces*
- *Abstract classes*

To make a List of Interface, Abstract, or Wild card items serializable, the item object definition must enable JSON write root. To enable JSON write root, set the following:

```
@SerializableType.jsonWriteRootAsField=true
```

Here's a code sample that shows you how to do that in more detail:

JSON code

```
{
    "roles": [
        {"type": "admin-role", "name": "admin", "privilege": 16},
        {"type": "consumer-role", "name": "ios", "description": "iOS mobile device"}
    ]
}
```

Java code

```

@SerializableType("boa")
public class BusinessObjectArchive {
    private List<Role> roles;

    public static interface Role {}

    @SerializableType(value = "admin-role",
        jsonWriteRootAsField = true, jsonRootField = "type")
    public static class AdminRole implements Role {
        private String name;
        private int privilege;
    }

    @SerializableType(value = "consumer-role",
        jsonWriteRootAsField = true, jsonRootField = "type")
    public static class ConsumerRole implements Role {
        private String name;
        private String description;
    }
}
BusinessObjectArchive boa = new BusinessObjectArchive();
boa.getRoles().add(new BusinessObjectArchive.AdminRole("admin", 16));
boa.getRoles().add(new BusinessObjectArchive
    .ConsumerRole("ios", "iOS mobile device"));

```



Note: The annotation `@SerializableType` for field data types have been set as `<jsonWriteRootAsField = true>, <jsonRootField = "type">`

7.5.5.8 Summary of Support for Java List Data Type

List Item Data Type	Marshalling	Unmarshalling
Simple data types: <ul style="list-style-type: none"> • <code><String></code> • <code><Date></code> • <code><Integer></code> • <code><Double></code> • <code><boolean></code> • <code><Enum></code> 	Yes	Yes
Complex type annotated by <code>@SerializableType</code>	Yes	Yes
<ul style="list-style-type: none"> • Interface • Abstract Class • Generic type • Wild card type 	Yes but only when the data type in the runtime instance is serializable	Yes but only when the data type in the runtime instance is serializable Specifically for JSON when: <ul style="list-style-type: none"> • The JSON root is written Specifically for XML when: <ul style="list-style-type: none"> • Wrapped • Written as a non property bag
None of the above	No	No

7.5.5.9 Limitation in Unmarshalling Number Lists

The current framework has a limitation in unmarshalling XML representations that contain number lists. This is because the framework cannot tell the level of precision of a number from an `item` element. For example, `<item>10</item>` can be a byte, but it can also be an `int`. Similarly, `<item>9.5</item>` can be a `float`, but it can also be a `double`. In this case, the framework adopts the lowest precision type that is compatible with the value. Therefore, `<item>10</item>` is unmarshalled to a `byte` and `<item>9.5</item>` is unmarshalled to a `float`.

7.5.6 Java Map Support

7.5.6.1 Java Map Data Type Support

Foundation REST API version 7.3 and later have support for `<java.util.Map>`.

```
@SerializableField
private Map<String, String> countryCodes;
```

JSON objects can be marshalled as JSON key value pairs. For example:

```
"countryCodes": {
    "usa" : "001",
    "china" : "086"}
```

XML objects can also be marshalled using property bags. For example:

```
<countryCodes>
    <countryCode>
        <country name="usa">001</country>
        <country name="china">086</country>
    </countryCode>
</countryCodes>
```

7.5.6.1.1 Supported Data Type



Caution

To be consistent for XML and JSON marshalling, the map key must be a `String` data type

7.5.6.1.2 Inclusions

The first argument (key) data type in a Map must have a `String` data type. The second argument (value) data type in a Map can be any one of the following data types:

- *Declared native data type*

`Map<String, String>, Map<String, Date>, Map<String, Integer>, Map<String, Double>, Map<String, boolean>`

- *Declared enum data type*

`Map<String, EnumUserTypes>, Map<String, EnumDocTypes>`

- Declared List data type
Map<String, List<String>>, Map<String, List<?>>
- Declared complex data type with annotation @SerializableType
Map<String, AtomFeed>, Map<String, Link>, Map<String, PersistentObject>
- Interface
Map<String, IFolder>, Map<String, IUser>
- Abstract class
Map<String, AbstractFolder>, Map<String, AbstractDocument>
- Generic or wild object data type
Map<String, Object>, Map<String, ?>, Map<String, T>



Caution

Runtime check of parameterized data type

Whether or not instances of *List*, *Interface*, *Abstract class*, and *Generic or wild card* objects can be marshalled is determined at runtime by the data type of the instance.

7.5.6.1.3 Marshalling

7.5.6.1.3.1 Marshalling with XML

Marshalling a Map with XML is more complex than JSON due to the following:

- XML fields can have separate namespaces
- XML fields for a Map of complex types can be serialized as property bags or standalone XML objects

Here are some code samples that show you how to serialize (marshall) with XML:

➤ Example 7-48: Map of Native Data Type as XML Elements

Java Code

```
@SerializableType("bom")
public class BusinessObjectMap {
    Map<String, String> codes;
    ...
}
BusinessObjectMap bom = new BusinessObjectMap();
bom.getCodes().put("usa", "001");
bom.getCodes().put("china", "086");
```

XML Code

```
<bom>
    <codes>
        <usa>001</usa>
        <china>086</china>
```

```
</codes>
</bom>
```



Example 7-49: Map of Complex Data Type as XML Elements

Java Code

```
@SerializableType("bom")
public class BusinessObjectMap {
    @SerializableField(defaultImpl = TreeMap.class)
    Map<String, Industry> industries;

    @SerializableType
    public static abstract class Industry {
        protected String industry;
    }

    @SerializableType(value = "healthcare")
    public static class Healthcare extends Industry {
        @SerializableField("cc")
        private int countryCode;
    }

    @SerializableType(value = "energy")
    public static class Energy extends Industry {
        private String field;
    }
    ...
}
BusinessObjectMap bom = new BusinessObjectMap();
bom.getIndustries().put("s1", new BusinessObjectMap.Healthcare("hc", 1));
bom.getIndustries().put("s2", new BusinessObjectMap.Energy("en", "nuclear"));
```

XML Code

```
<bom>
    <industries>
        <s1>
            <healthcare>
                <cc>1</cc>
                <industry>hc</industry>
            </healthcare>
        </s1>
        <s2>
            <energy>
                <field>nuclear</field>
                <industry>en</industry>
            </energy>
        </s2>
    </industries>
</bom>
```



Example 7-50: Map of Array of Native Data Type as XML Elements

Java Code

```
@SerializableType("bom")
public class BusinessObjectMap {
    Map<String, String[]> codes;
    ...
}
BusinessObjectMap bom = new BusinessObjectMap();
```

```
bom.getCodes().put("asia", new String[]{"086", "091"});
bom.getCodes().put("west", new String[]{"001", "044"});
```

XML Code

```
<bom>
  <codes>
    <asia>
      <item>086</item>
      <item>091</item>
    </asia>
    <west>
      <item>001</item>
      <item>044</item>
    </west>
  </codes>
</bom>
```

**Example 7-51: Map of List of Complex Data Type as XML Elements***Java Code*

```
@SerializableType("bom")
public class BusinessObjectMap {
    Map<String, List<Role>> roles;

    public static interface Role {}

    @SerializableType(value = "admin-role")
    public static class AdminRole implements Role {
        private String name;
        private int privilege;
    }

    @SerializableType(value = "consumer-role")
    public static class ConsumerRole implements Role {
        private String name;
        private String description;
    }
    ...
}
BusinessObjectMap bom = new BusinessObjectMap();
bom.getRoles().put("r1", Arrays.asList(
    new BusinessObjectMap.AdminRole("admin", 16),
    new BusinessObjectMap.ConsumerRole("ios", "iOS mobile device")
));
bom.getRoles().put("r2", Arrays.asList(
    new BusinessObjectMap.AdminRole("dbowner", 8),
    new BusinessObjectMap.ConsumerRole("android", "Android mobile device")
));
```

XML Code

```
<bom>
  <roles>
    <r1>
      <admin-role>
        <name>admin</name>
        <privilege>16</privilege>
      </admin-role>
      <consumer-role>
        <name>ios</name>
        <description>iOS mobile device</description>
      </consumer-role>
    </r1>
  </roles>
</bom>
```

```

<r2>
<admin-role>
    <name>dbowner</name>
    <privilege>8</privilege>
</admin-role>
<consumer-role>
    <name>android</name>
    <description>Android mobile device</description>
</consumer-role>
</r2>
</roles>
</bom>

```



Example 7-52: Map of Native Data Type as XML prop-bag Elements

Java Code

```

SerializableType("bom")
public class BusinessObjectMap {
    @SerializableField4XmlMap(asPropBag = true)
    Map<String, String> codes;
    ...
}
BusinessObjectMap bom = new BusinessObjectMap();
bom.getCodes().put("usa", "001");
bom.getCodes().put("china", "086");

```

XML Code

```

<bom>
    <codes>
        <entry key="usa">001</entry>
        <entry key="china">086</entry>
    </codes>
</bom>

```



Example 7-53: Map of Complex Data Type as XML prop-bag Elements

Java Code

```

@SerializableType("bom")
public class BusinessObjectMap {
    @SerializableField(defaultImpl = TreeMap.class)
    @SerializableField4XmlMap(asPropBag = true,
        propBagEleName = "industry",
        propBagKeyName = "type")
    Map<String, Industry> industries;

    @SerializableType
    public static abstract class Industry {
        protected String industry;
    }

    @SerializableType(value = "healthcare")
    public static class Healthcare extends Industry {
        @SerializableField("cc")
        private int countryCode;
    }

    @SerializableType(value = "energy")
    public static class Energy extends Industry {

```

```

        private String field;
    }
...
}
BusinessObjectMap bom = new BusinessObjectMap();
bom.getIndustries().put("s1", new BusinessObjectMap.Healthcare("hc", 1));
bom.getIndustries().put("s2", new BusinessObjectMap.Energy("en", "nuclear"));

```

XML Code



7.5.6.1.3.2 Marshalling with JSON

When marshaling a Map with JSON, all map entries are serialized as key-value pairs of a JSON object.

► Example 7-54: Map of Native Data Types as JSON Fields

Java Code

```

@SerializableType("bom")
public class BusinessObjectMap {
    Map<String, String> codes;

}
BusinessObjectMap bom = new BusinessObjectMap();
bom.getCodes().put("usa", "001");
bom.getCodes().put("china", "086");

```

JSON Code

```
{
    "codes":{
        "usa":"001",
        "china":"086"
    }
}
```



► Example 7-55: Complex Data Type as JSON Fields

```

@SerializableType("bom")
public class BusinessObjectMap {
    Map<String, Industry> industries;

    @SerializableType
    public static abstract class Industry {
        protected String industry;
    }

    @SerializableType(value = "healthcare")
    public static class Healthcare extends Industry {
        @SerializableField("cc")
        private int countryCode;
    }

    @SerializableType(value = "energy")
    public static class Energy extends Industry {
        private String type;
    }
...

```

```

}
BusinessObjectMap bom = new BusinessObjectMap();
bom.getIndustries().put("s1", new BusinessObjectMap.Healthcare("hc", 1));
bom.getIndustries().put("s2", new BusinessObjectMap.Energy("en", "nuclear"));

```

JSON Code

```
{
    "industries": {
        "s2": {
            "type": "nuclear",
            "industry": "en"
        },
        "s1": {
            "cc": 1,
            "industry": "hc"
        }
    }
}
```

**Example 7-56: List of Complex Data Type as JSON Fields***JAVA Code*

```

@SerializableType("bom")
public class BusinessObjectMap {
    private Map<String, Role> roles;

    public static interface Role {}

    @SerializableType(value = "admin-role")
    public static class AdminRole implements Role {
        private String name;
        private int privilege;
    }

    @SerializableType(value = "consumer-role")
    public static class ConsumerRole implements Role {
        private String name;
        private String description;
    }

    ...
}

BusinessObjectMap bom = new BusinessObjectMap();
bom.getRoles().put("r1", Arrays.asList(
    new BusinessObjectMap.AdminRole("admin", 16),
    new BusinessObjectMap.ConsumerRole("ios", "iOS mobile device")
));
bom.getRoles().put("r2", Arrays.asList(
    new BusinessObjectMap.AdminRole("dbowner", 8),
    new BusinessObjectMap.ConsumerRole("android", "Android mobile device")
));
{

```

JSON Code

```

"roles": {
    "r1": [
        {
            "name": "admin",
            "privilege": 16
        },

```

```
{
    "name": "ios",
    "description": "iOS mobile device"
}
],
"r2": [
{
    "name": "dbowner",
    "privilege": 8
},
{
    "name": "android",
    "description": "Android mobile device"
}
]
}
```



7.5.6.1.4 Unmarshalling

A Map can be deserialized using JSON or XML messages but it has certain constraints with respect to the class definition.

The following field types can always be serialized:

- Declared native data type as the Map value

```
Map<String, String>, Map<String, Date>, Map<String, Integer>, Map<String, Double>, Map<String, boolean>
```

- Declared Enum data type

```
Map<String, Enum User Types>, Map<String, Enum DocTypes>
```

- Declared List data type with concrete List item type

```
Map<String, List<String>>
```

- Declared complex data type with annotation `@SerializableType`

```
Map<String, AtomFeed>, Map<String, Link>, Map<String, PersistentObject>
```

The following field data types can be serialized using additional annotation attributes for XML and JSON, respectively:

- *Interface*

```
Map<String, IFolder>, Map<String, IUser>
```

- *Abstract class*

```
Map<String, AbstractFolder>, Map<String, AbstractDocument>
```

- *Generic or wild card data type*

```
Map<String, Object>, Map<String, ?>, Map<String, T>
```

- *Declared List data type with unresolved list item type*

```
Map<String, List<? extends IUser>>
```

7.5.6.1.4.1 Unmarshalling with XML

For XML, the Map entry value is always deserializable and there are no constraints. Messages for the following data types can be serialized to a Map field:

- Native data types as XML elements
- Complex data types as XML elements
- Array of native data types as XML elements
- List of complex data types as XML elements
- Simple data types as XML prop-bag elements
- Complex data types as XML prop-bag elements

7.5.6.1.4.2 Unmarshalling with JSON

The following data types can be serialized:

- Native data type as JSON fields
- Complex data type as JSON fields

A List of complex data type cannot be serialized into JSON fields because the type information is not determined from either the declared data type such as `Map<String, Role>`, or the JSON message such as `{name:xx, ...}`.

To make a Map of an Interface, Abstract class, and Wild card items serializable, the object definition for the item must enable the JSON write root attribute as shown here:

```
SerializableType.jsonWriteRootAsField=true
```

Here's a code sample that shows you how to do that:

➡ Example 7-57: Read a Map of Interfaces from JSON

Java

```
@SerializableType("bom")
public class BusinessObjectMap {
    private Map<String, Role> roles;

    public static interface Role {}

    @SerializableType(value = "admin-role",
                      jsonWriteRootAsField = true,
                      jsonRootField = "type")
    public static class AdminRole implements Role {
        private String name;
        private int privilege;
    }

    @SerializableType(value = "consumer-role",
                      jsonWriteRootAsField = true,
                      jsonRootField = "type")
    public static class ConsumerRole implements Role {
        private String name;
        private String description;
    }
}
```

```

        }
    ...
}
BusinessObjectMap bom = new BusinessObjectMap();
bom.getRoles().put("r1", Arrays.asList(
    new BusinessObjectMap.AdminRole("admin", 16),
    new BusinessObjectMap.ConsumerRole("ios", "iOS mobile device")
));
bom.getRoles().put("r2", Arrays.asList(
    new BusinessObjectMap.AdminRole("dbowner", 8),
    new BusinessObjectMap.ConsumerRole("android", "Android mobile device")
));

```

The annotation `@SerializableType` for the field data types has been set to `jsonWriteRootAsField=true, jsonRootField="type"`.

JSON

```
{
    "roles": {
        "r1": [
            {
                "type": "admin-role",
                "name": "admin",
                "privilege": 16
            },
            {
                "type": "consumer-role",
                "name": "ios",
                "description": "iOS mobile device"
            }
        ],
        "r2": [
            {
                "type": "admin-role",
                "name": "dbowner",
                "privilege": 8
            },
            {
                "type": "consumer-role",
                "name": "android",
                "description": "Android mobile device"
            }
        ]
    }
}
```

The `type` attribute is written into the JSON object.



7.5.6.1.4.3 Summary of Support for the Map Data Type

The following table shows Map data type support:

Map Entry Value Data Type	Marshalling	Unmarshalling
Simple data types: String, Date, Integer, Double, boolean, Enum	Yes	Yes
Complex type annotated by <code>@SerializableType</code>	Yes	Yes
Interface, Abstract Class Generic type or Wild card type	Yes but only when the data type in the runtime instance is serializable	Yes but only when the data type in the runtime instance is serializable Specifically for JSON when: <ul style="list-style-type: none">• json root is written
List data type with item type as preceding data types	Yes	Accordingly
None of the above	No	No

7.5.7 Circular References Not Supported

The Foundation REST API marshalling framework does not support circular references. If a data model has a field referencing itself and the field is annotated, then marshalling framework returns a stack overflow error.

7.5.8 Custom Serializers and Deserializers

The unified data binding framework allows you to customize marshalling and unmarshalling data model in many aspects.

There are requirements at the field level that cannot be fulfilled by the out-of-box capability of the framework.

Here are several typical cases:

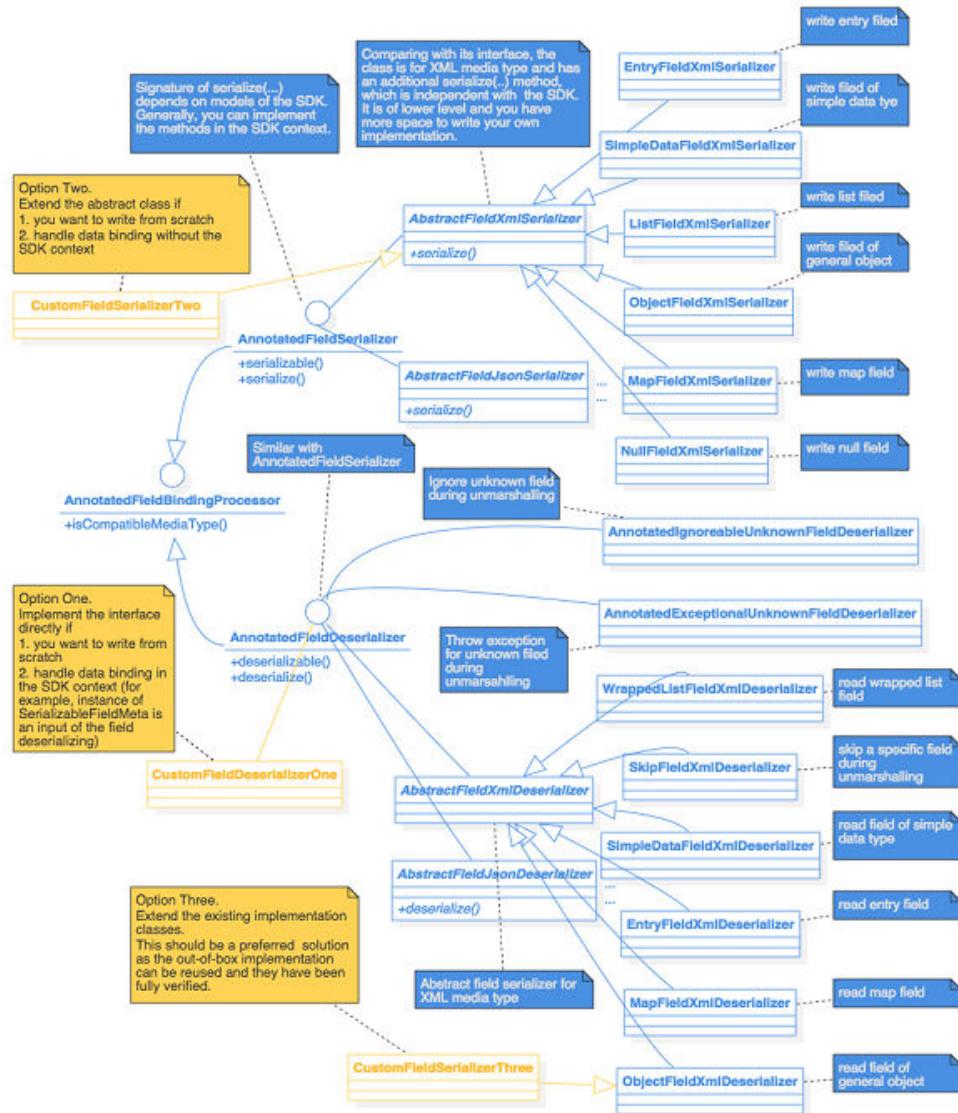
1. Representation of different media types is different, like the `<AtomEntry>` representation of XML and JSON media types.
2. Data binding depends on runtime data. For example, do not marshall a wrapped list when it is empty.
3. A field of the class must be written out during marshalling, but should not be read in during unmarshalling.

7.5.8.1 Programming Interface

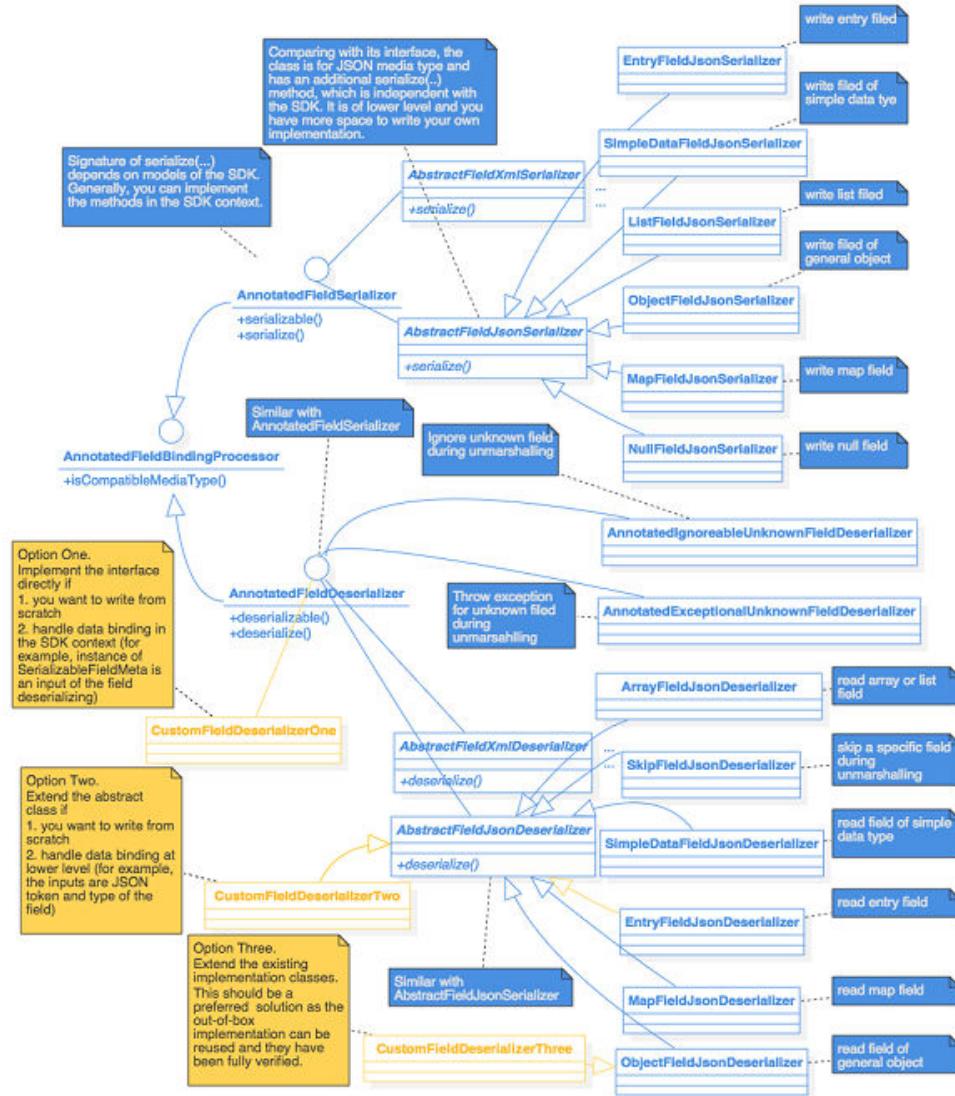
As you can see in the following image, the Foundation REST API data binding framework is composed of many serializers and deserializers for different kinds of fields.

This architecture can be extended. Along with the `SerializableField#serializers` and `SerializableField#deserializers` annotations, you can also link new ones with specified fields.

The following is a class diagram for XML data binding classes:



Here's the class diagram for JSON



Generally, you can implement a new custom serializer or deserializer in two ways:

1. By extending abstract serializers or deserializers and implementing the abstract methods
2. By extending Foundation REST API out-of-box serializers or deserializers

Building new serializers or deserializers from scratch is not easy work. We recommend that you extend existing serializers or deserializers of the Foundation REST API.

Samples are available in `<dctm-rest-sdk-root>/samples/custom-field-binding-samples`

7.5.9 Annotation Scanner

An annotation scanner is available in the tools directory of the SDK to validate custom resources focusing on the annotation aspect. This annotation scanner runs when you build up the WAR package and generates a report listing all invalidated annotation occurrences in your code, such as `Field not serializable` and `Duplicated annotation values`.

The annotation scanner can be used as a runnable JAR or a Maven plug-in.

Runnable JAR

To use the annotation scanner as a runnable JAR, navigate to tools directory of the SDK and then run the following command:

```
java -jar dctm-rest-ext-validator-runnable [TARGET] [option]
```

[TARGET] represents the source files to scanner, which can be:

- A single JAR, WAR, or EAR file, such as `./tmp/dctm-rest.jar`
- Multiple JAR or WAR files separated by comma, such as `./tmp/databinding.jar, ./tmp/test-annotation.jar`
- A directory containing the binary, JAR, or EAR files, such as `./tmp`

[option] represents the options of this command:

- `-h` Display help messages
- `-o` Specify the output directory, such as `-o ./output`
- `-x` Display debug messages

Maven plug-in

To use the annotation scanner as a Maven plug-in:

1. Navigate to tools directory of the SDK and then run the following command to install the plug-in to your local repository:

```
mvn install:install-file -Dfile=documentum-rest-annotation-plugin-<version-number>.jar -DpomFile=pom
```

2. Open the `pom.xml` file of your project and then append the following plug-in to the `plugins` block .

```
<plugin>
    <groupId>com.emc.documentum.rest</groupId>
    <artifactId>documentum-rest-extension-validating</artifactId>
    <version>7.3</version>
    <inherited>true</inherited>
    <executions>
        <execution>
            <id>validate-annotation</id>
            <phase>verify</phase>
            <goals>
                <goal>check-annotation</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

```
        </goals>
        <configuration>
            <input>${project.basedir}/target/${scan.artifactId}-${version}.war</
input>
            <outputDir>${project.basedir}/target</outputDir>
            <debug>false</debug>
            <failBuild>true</failBuild>
        </configuration>
        </execution>
    </executions>
</plugin>
```

In the plug-in configuration:

- The `input` element specifies the source files to scan
- The `outputDir` element specifies directory to hold the report
- The `debug` element determines whether the debug mode is turned on when the scanner runs
- The `failBuild` element determines whether to terminate and fail the building process when the scanner detects errors

When the scan completes, an HTML file named Scan Report of Documentum REST Services Extensibility Annotation is generated in the output directory. This file lists all annotation violations and fix recommendations. All issues should be resolved before you build the custom REST WAR file.

If necessary (though not recommended), you can disable this scanner by removing the plug-in configuration in the `pom.xml` file.

7.6 Developing custom resources

This section discusses how to develop simple custom resources with the Documentum REST Services Extensibility.

Typically, custom resource development can be divided into the following phases:

- *Designing Custom Resources*
- *Setting up a Custom Resource Project*
- *Programming for Custom Resources*
- *Linking Custom and Core Resources*
- *Packaging and Deploying*

Each one of the preceding phases is discussed here.

7.6.1 Designing Custom Resources

This phase focuses on the following tasks:

- Designing the XML and/or JSON representation of your custom resources
- Determining how client applications discover your custom resources
- Determining the operations to support

Typically, a custom resource can represent a persistent object, a collection of persistent objects, or a certain state of the persistent objects. A key property that distinguishes the REST architecture from other software architectures is that REST is resource-oriented. So the first step for custom resource development is to model the persistent data into resources. If the persistent data is too large, has deep hierarchy, or there are a lot of operations on it, you may consider designing more than one resources for the large persistent data.

7.6.1.1 URI

The URI design for a resource is implementation-specific. Theoretically, you can design any URI pattern for a custom resource. Practically, however, the custom resource URI pattern should be similar to that of the Core resources. One benefit is to apply the same authentication schemes as Core resources, because Core REST security component checks the resource URLs to determine whether authentication is required on the custom resources. For instance, all repository level resources are required for authentication by default, so the URI of a custom resource under a particular repository should follow this pattern: `/repositories/<repositoryName>/<customSegments>`.



Note: When the designed resource URI template contains path variables where their values come from the object properties, there could be URI encoding/decoding issues if the property values contain URI preserved characters. Foundation REST API SDK provides a utility `com.emc.documentum.rest.utils.NameAsPathCoder` helping you encode and decode the path variable values to escape special characters.

7.6.1.2 HTTP Methods

HTTP /1.1 specification provides several standard HTTP methods for clients to perform on a resource. Although you are allowed to create custom HTTP methods on a custom resource, the challenges it brings is usually far more than the given benefits. So it is recommended to stick to standard HTTP methods for resource operations. In case there are a lot of logic operations applied to the same persist data that the resource represents for, you may consider dividing the operations into multiple resources.

Core resources use standard HTTP methods GET, POST, PUT and DELETE in all places.

7.6.1.3 Representations

Core resources support both JSON and XML representations. For a custom resource, whether to support one or both representations depends on your business requirements. Foundation REST API provides an annotation-based marshalling framework so that an annotated Java model can support both JSON and XML representations out of the box. You do not need to handle the message marshalling or unmarshalling except for the Java model design.

If you want to limit the custom resource to support JSON or XML representation only, specific media type constraints can be applied to the resource controller to precisely define the supported representation with the Spring annotation `@RequestMapping`.

7.6.1.4 Content Negotiation

Custom resources follow the same content negotiation mechanism as Core resources to determine a specific representation format for a client request.

In both Core and custom resources, all operations support the following media types:

- `application/atom+xml`
- `application/vnd.emc.documentum+xml`
- `application/vnd.emc.documentum+json`

The GET operation also supports the following two generic media types:

- `application/xml`
- `application/json`

The current Documentum REST Services Extensibility feature does not support custom media types.

7.6.1.5 Link Relations

In most cases, a custom resource design has one or more outer links pointing to Core resources or other custom resources. This will be done by designing new link relations on the custom resource representation. It is recommended to look up well-known link relations first from IANA website and use them as much as possible before considering inventing new link relation names for the custom resources. Besides, a good practice for inventing new link relation names is that nouns are preferred other than verbs to name a link relation.

It is possible to add new links to Core resources. For more information, see [Adding Links to Core Resources](#).

7.6.2 Setting up a Custom Resource Project

In this phase, you will set up a project with the appropriate structure to hold various packages, source files, and configuration files. We recommend that you leverage the Maven or Ant toolkit provided in the SDK for project setup. Foundation REST API SDK provides you with a convenient way to set up the first custom resource project with a Maven archetype. This section introduces the general project structure for custom resource development.

Foundation REST API resource development mainly leverages Spring Web MVC to construct the resources and uses Maven to build projects. Therefore, each resource implementation must have well-formed code structure as follows.

Typical project structure

```
rest-api-foo
|---rest-api-foo-model
|---rest-api-foo-persistence
|---rest-api-foo-resource
```

This code structure is a typical Foundation REST API resource project structure, which should be aligned with in your custom resource structure so that the implementation is well layered. This code structure separates the custom resource implementation into three modules:

- model
Designs annotated Java model classes for custom resources
- persistence
Creates persistence API beans by referencing Foundation Java API or other local persistence APIs
- resource
Creates resource controllers for custom resources

With this code structure, the custom model, persistence, and controller implementations are built as separate JAR files.

For a custom resource that does not require additional models or persistence APIs, the corresponding module and/or persistence can be omitted.

For each sub module, the code structure is organized as a typical Maven module.

Typical module structure

```
rest-api-foo
|---rest-api-foo-xxx
|   |---src
|   |   |---main
|   |   |   |---java
|   |   |   |---resources
|   |---test
|       |---java
|       |-resources
```

```
|---pom  
|---pom
```

The quick start is to use the Maven archetype project in Foundation REST API SDK to create such a project structure.

7.6.3 Programming for Custom Resources

In this phase, you will take advantage of the marshalling framework and core REST Java library to develop your custom resources. This section introduces you to the important features of custom resource programming. For a comprehensive review of typical custom resource development, see [Tutorial: Foundation REST API Extensibility Development](#).

7.6.3.1 Model: Programming

The model package defines the data structure of the resource representation for both input and output. Being decorated with Foundation REST API marshalling framework, an annotated model class has direct data binding to its resource representation. You only need to focus on the decision of what domain information to be exposed in the model, and the marshalling framework takes full responsibility for marshalling and unmarshalling the data model into/from XML and JSON messages. Here is an example of a new resource model.

```
/*  
 * Define a server model which has three properties.  
 */  
@SerializableType(value = "server",  
    fieldVisibility = SerializableType.FieldVisibility.ALL,  
    fieldOrder = {"name", "host", "version"},  
    xmlNS = "http://identifiers.emc.com/vocab/documentum",  
    xmlNSPrefix = "dm")  
public class Server {  
    private String name;  
    private String host;  
    private String version;  
  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public String getHost() {  
        return host;  
    }  
    public void setHost(String host) {  
        this.host = host;  
    }  
    public String getVersion() {  
        return version;  
    }  
    public void setVersion(String version) {  
        this.version = version;  
    }  
}
```



Note: An annotated model class must have one parameterless constructor. Otherwise, an exception is thrown during unmarshalling. In the sample, we do not define any constructor for the `Server` class so that the default (no-

argument) one provided by the compiler is applied. If you define a constructor with arguments for the model class, you must explicitly define one parameterless constructor at the same time.

7.6.3.2 Model: Extending Core

You can also extend Core REST resource model classes for holding additional information on the custom resource representation. The Core REST library provides a persistent object model class `com.emc.documentum.rest.model.PersistentObject` that provides the fundamental properties and links collection for a persistent OpenText Documentum CM object. Here is an example of an extended persistent object model.

```
/*
 * Define a business object model that has one additional property called 'uuid'.
 */
@SerializableType(value = "biz-object",
    jsonWriteRootAsField = true,
    fieldVisibility = SerializableType.FieldVisibility.NONE,
    ignoreNullFields = true,
    jsonRootField = "name",
    fieldOrder = {"type", "definition", "uuid", "properties"},
    xmlNS = "http://identifiers.emc.com/vocab/documentum",
    xmlNSPrefix = "dm")
public class BusinessObject extends com.emc.documentum.rest.model.PersistentObject {
    @SerializableField (xmlAsAttribute = true)
    private String uuid;

    public String getUuid() {
        return uuid;
    }

    public void setUuid(String uuid) {
        this.uuid = uuid;
    }
}
```



Note: There are more model samples in the SDK: `documentum-rest-<version>.zip/samples/documentum-rest-model-samples/`

7.6.3.3 Model: Validation

It is always good to validate the designed models with the [Foundation REST API Annotation Scanner](#). The tool can be run during the process of Maven project creation or run separately in a command line. The scanning report tells the detail of the model analyze result, as well as fix instructions. All ERROR level rule violations must be fixed. Here is a sample of the scan report.

Scan Report of Documentum Rest Extensibility Annotation

Summary

Validator Version: 1.0

OS Info: Mac OS X, 10.9.5, x86_64

JRE Info: 1.7.0_60

Start Time: 2015/01/05 15:05:11

Time Cost: 2.841000 seconds

Items Scanned:

///Users/Developer/Workspace/acme-rest/acme-rest-web/target/acme-rest-web-0.0.1-SNAPSHOT.war

Figure 7-7: Scan Report

7.6.3.4 Persistence: Programming

The persistence API wraps the Foundation Java API operations to provide persistent data operations for the resources. The best practice for creating a new persistence API is to separate the API into one interface and one implementation class, and load the implementation class by using a Java bean.

1. Create a Java interface.

```
public interface UserManager {  
    com.emc.documentum.rest.model.UserObject createUser(UserObject newUser);  
}
```

2. Create a Java class to implement this interface.

```
public class UserManagerImpl  
    extends com.emc.documentum.rest.dfc.SessionAwareAbstractManager  
    implements UserManager {  
    public com.emc.documentum.rest.model.UserObject createUser(UserObject newUser) {  
        ...  
    }  
}
```

3. Implement it as a Java bean.

There are two ways to create a Java bean:

- Spring Java code configuration
- Using an XML namespace

➡ Example 7-58: Create a Java Bean Using Spring Code Configuration

```
@Configuration  
@ComponentScan(  
    basePackages = { "com.acme" },  
    excludeFilters = {  
        @ComponentScan.Filter(
```

```

        type = FilterType.CUSTOM,
        classes = {
            com.emc.documentum.rest.context.ComponentScanExcludeFilter.class
        }
    }
}

public class CustomContextConfig {
    @Bean(name="customUserManager")
    public UserManager customUserManager(){
        return new UserManagerImpl();
    }
}

```

You must ensure that the package where you created your custom configuration class is specified in the `rest.context.config.location` property within the `rest-api-runtime.properties` file.

The `com.emc.documentum.rest.context.ComponentScanExcludeFilter` exclude filter is mandatory when you use the `@ComponentScan` Spring annotation in your custom defined configuration class. This exclude filter ensures that the Spring framework loads all of the resources that you have defined.



➡ Example 7-59: Create a Java Bean Using Spring XML Namespace

Define the bean in class path file: `/META-INF/spring/custom.xml`

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans ...>
    <bean id="customUserManager" class="com.acme.UserManagerImpl"/>
</beans/>

```



The Java beans can be referenced by resource controllers by using the `@Autowired` Spring annotation. Here is a code sample of the persistence API reference in a resource controller:

7.6.3.5 Persistence: Referencing Core

Foundation REST API SDK library provides lots of persistence APIs for developers to reuse to operate the persistent OpenText Documentum CM objects. For example, `com.emc.documentum.rest.dfc.ObjectManager` provides basic CRUD operations for any persistent objects; `com.emc.documentum.rest.dfc.SysObjectManager` provides additional methods for sysobject related operations, such as copy and checkout. Custom resource controllers can reference the instance of these Core persistence APIs as same as to custom persistence APIs.



Note: The complete persistence API docs can be found in the SDK:
`documentum-rest-<version>.zip/apidocs/`

7.6.3.6 Persistence: Getting Login User Context

The persistence API `<com.emc.documentum.rest.config.RepositoryContextHolder>` provides static methods to retrieve the login user's context information, such as the *login name, username, user privileges*, and so on.

Some persistence API methods require additional Foundation Java API calls, and should be used as little as possible. Refer to the JavaDoc API for more information.

7.6.3.7 Persistence: Session Management

Foundation REST API SDK library provides the interfaces `com.emc.documentum.rest.dfc.RepositorySessionManager` and `com.emc.documentum.rest.dfc.RepositorySession` for the session management in the persistence APIs.

The default implementations for these two interfaces have been integrated with the security module of the Foundation REST API. Therefore, after a user logs in, the persistence API retrieves the right user session without the need to explicitly instantiate the Foundation Java API session manager. The `SessionAwareAbstractManager` persistence API can be used to extend the `com.emc.documentum.rest.dfc` interface to get the user session. Here is a code sample that shows you how to do this:

 **Example 7-60: Getting User Session by Extending the `SessionAwareAbstractManager` API**

```
/**  
 * A user manager implementation extends SessionAwareAbstractManager to reuse the  
 * session beans  
 */  
public class CustomUserManager extends SessionAwareAbstractManager  
    implements UserManager {  
  
    @Override  
    public UserObject get(String userName,  
                          AttributeView attributeView) throws DfException {  
        IDfSession session = null;  
  
        try {  
            // Use the method from super class to get a session for the login user  
            session = getSessionRepository().getSession(false);  
            IDfUser dfUser = session.getUser(userName);  
  
            if(dfUser == null) {  
                return null;  
            }  
            else {  
                return convert(dfUser, attributeView);  
            }  
        }  
        finally {  
            // Do not forget to release the session  
            release(session);  
        }  
    }  
}
```



When you are not extending the `SessionAwareAbstractManager` persistence API, it can use the auto wired `RepositorySession` instance to retrieve a session by using the `@Autowired` Spring annotation. Here is a code sample that shows you how to do this:

 **Example 7-61: Getting User Session without Extending the `SessionAwareAbstractManager` API**

```
public class CustomUserManager implements UserManager {
    // Reference the repository session bean directly
    @Autowired
    private RepositorySession sessionRepository;

    @Override
    public UserObject get(String userName, AttributeView attributeView)
        throws DfException {
        IDfSession session = null;

        try {
            session = sessionRepository getSession(false);
            IDfUser dfUser = session.getUser(userName);

            if(dfUser == null) {
                return null;
            }
            else {
                return convert(dfUser, attributeView);
            }
        }
        finally {
            release(session);
        }
    }
}
```



If you are using a version of the JDK that is Java 7 or later, you can use Core REST `IDfCloseableSession` API to manage the session. This API automatically handles the session release. Here is a code sample that shows you how to use this API:

 **Example 7-62: Session management usage in Java and later**

```
public class CustomUserManager extends SessionAwareAbstractManager implements
UserManager{
    @Override
    public UserObject get(String userName, AttributeView attributeView) throws
DfException{
        IDfSession session = null;

        try (IDfCloseableSession session = dfcSessionRepository.getCloseableSession()) {
            // Use a method from super class to get the session for the login user
            IDfUser dfUser = session.getSession(false).getUser(userName);

            if(dfUser == null) {
                return null;
            }
            else {
                return convert(dfUser, attributeView);
            }
        }
    }
}
```



When the Foundation REST API security module is not used while performing unit testing of the persistence API, you can set the login username and password in your test code, which makes the session available to the test code. The code mocks a user login from a servlet request. Here is a code sample that shows you how to do this:

➤ **Example 7-63: Setting the Username and Password for Unit Testing**

```
com.emc.documentum.rest.config.RepositoryContextHolder.setRepositoryName(...)  
com.emc.documentum.rest.config.RepositoryContextHolder.setLoginName(...)  
com.emc.documentum.rest.config.RepositoryContextHolder.setPassword(...)
```

When a transaction is required on the resource controller, the `com.emc.documentum.rest.dfc.ContextSessionManager` class can be used to commit the transaction with the persistence API as shown in the following code sample:

➤ **Example 7-64: Commit a Transaction with the Persistence API**

```
public class CustomSysObjectManager implements SysObjectManager {  
    @Autowired  
    ContextSessionManager contextSessionManager;  
  
    @Override  
    public <T extends SysObject> T copy(final String objectId, final String folderId)  
        throws DfException {  
        // Put a regular operation into the transaction manager  
        return contextSessionManager.executeWithinTheContextTran  
            (new SessionCallable<T>() {  
                public T call(IDfSession session) throws Exception {  
                    return doRegularCopy(objectId, folderId);  
                }  
            });  
    }  
}
```



Note: Additional session management samples are available in the `documentum-rest-<version>.zip/samples/documentum-rest-java_api-samples/` SDK.

7.6.3.8 Persistence: Creating Feed Pages from DQL Result

For an implementation of a feed resource, the persistence API may execute a DQL query to get the object collection. Foundation REST API SDK library provides the `com.emc.documentum.rest.paging.Page<T>` class to retrieve the object collection as a feed page.

The `com.emc.documentum.rest.dfc.query.PagedQueryTemplate<T>` class is the basic DQL template that is used to produce a DQL query expression. A custom feed resource can extend the `PagedQueryTemplate` class to create a custom DQL query expression and call the `com.emc.documentum.rest.paging.PagedDataRetriever<T>` method to get a `Page<T>`. Here are some code samples that demonstrate how to do some typical tasks:

► **Example 7-65: Use the PagedPersistentDataRetriever to get a page for a custom PagedQueryTemplate**

The Foundation REST API SDK library provides a default implementation of `PagedDataRetriever com.emc.documentum.rest.paging`.

`PagedPersistentDataRetriever` to retrieve a page of persistent objects. The return type is a `com.emc.documentum.rest.model.PersistentObject` type or one of its sub types. Here is a code sample that shows you how to retrieve a page of document objects:

```
// Define a custom DQL for the document collection
PagedQueryTemplate pagedQueryTemplate = new CustomPagedQueryTemplate(getUsername());

// Retrieve a page of document objects with the default object collection
// implementation and paging parameters
PagedDataRetriever<DocumentObject>
pagedDataRetriever = new PagedPersistentDataRetriever<DocumentObject>(
    pagedQueryTemplate, 1, 5, true, AttributeView.DEFAULT, DocumentObject.class);
Page<DocumentObject> page = pagedDataRetriever.get();
```



► **Example 7-66: Use the PagedDataRetriever to get a page for a custom PagedQueryTemplate with custom models**

You can create a custom object collection manager if the data model in the page does not extend `com.emc.documentum.rest.model.PersistentObject`. Here is a code sample that shows you how to do this:

```
PagedQueryTemplate pagedQueryTemplate = new CustomPagedQueryTemplate(getUsername());

PagedDataRetriever<MyDocument>
pagedDataRetriever = new PagedDataRetriever<MyDocument>(
    new MyDocumentCollectionManager(),
    pagedQueryTemplate, 1, 5, true, AttributeView.DEFAULT);
Page<MyDocument> page = pagedDataRetriever.get();
```



► **Example 7-67: Use the Paginator to produce the page from any generic collection**

For a feed resource implementation where its persistent data does not come from a simple DQL, you can create your own implementation of the persistence layer and call the `com.emc.documentum.rest.paging.Paginator` method to generate a page from the object collection. Here is a code sample that shows you how to do this:

```
String dql = "select * from dm_document";
session = dfcSessionRepository.getSession();
List<IDfTypedObject> results = QueryExecutor.run(session, dql);
List<DocumentObject> docs = convert(results);
List<IDfTypedObject> counts = QueryExecutor
    .run(session,
        "select count(r_object_id) as total from dm_document");

int total = counts.get(0).getInt("total");
Paginator<DocumentObject>
paginator = new Paginator<DocumentObject>(docs, total);
int pageNumber = 3;
```

```

int itemsPerPage = 5;
Page<DocumentObject>
currentPage = paginator.paginate(pageNumber, itemsPerPage);

```



Note: There are detailed code samples in the `documentum-rest-<version>.zip/samples/documentum-rest-java_api-samples/` SDK.

7.6.3.9 Resource: Programming the Controller

The resource implementation uses the Spring controller to define the REST mapping of the resource. The controller class has Spring REST annotations to define its request mapping and Response mapping. It loads the auto wired persistence API to manipulate the model data. The return type of the controller method is the model class. Here is a code sample that shows you how to use the resource controller:

Example 7-68: Using the Spring Resource Controller

```

@Controller("format")
@RequestMapping("/repositories/{repositoryName}/formats-x/{formatName}")
@ResourceViewBinding(value = FormatView.class)
public class MyFormatController extends AbstractController {
    @Autowired
    private FormatManager dfcFormatManager;
    @RequestMapping( method = RequestMethod.GET, produces = {
        SupportedMediaTypes.APPLICATION_VND_DCTM_JSON_STRING,
        SupportedMediaTypes.APPLICATION_VND_DCTM_XML_STRING,
        MediaType.APPLICATION_JSON_VALUE,
        MediaType.APPLICATION_XML_VALUE
    })
    @ResponseBody
    @ResponseStatus(HttpStatus.OK)
    public Format get(
        @PathVariable("repositoryName") String repositoryName,
        @PathVariable("formatName") String formatName,
        @TypedParam final SingleParam param,
        @RequestUri final UriInfo uriInfo) throws DfException {
        Format format =
            dfcFormatManager.getFormat(NameAsPathCoder.decode(formatName),
                param.getAttributeView());
        if (format == null) {
            throw new DfNoMatchException(NameAsPathCoder.decode(formatName));
        }
        return getRenderedObject(repositoryName, format, param.isLinks(), uriInfo, null);
    }
}

```



The controller class declares the `@Controller` Spring annotation, which identifies it as a resource controller. We recommend that you set a unique name for the `@Controller`. The name for the controller is used as the code name of this resource, which is used by configurations and log entries that reference this resource.

The controller class defines its request mapping with the `@RequestMapping` Spring annotation. There are some additional mapping rules, however they do not apply to the URI on this annotation. For more information, refer to the Spring framework documentation.

In the controller definition, the `@ResourceViewBinding` Core REST annotation defines the default view for this resource.

The controller class extends from `com.emc.documentum.rest.controller.AbstractController`, which provides some useful methods that can be reused by the extended controller.

In the controller method definition, the `@RequestMapping` annotation can be declared and used to map the HTTP method and media types for a resource operation.

The controller method defines the `@ResponseBody` Spring annotation , which indicates that the returning object format can be directly sent to the marshalling framework.

The controller method defines the `@ResponseStatus` Spring controller annotation, which specifies the HTTP status of this resource operation. In the event that there is an exception thrown by this method, error mapping takes place and the status of the method is reset to the corresponding error status.

For more information, see [Error Handling and Representations](#).

The resource method can call its controller super class method `getRenderedObject` or `getRenderedPage` to invoke dynamic view definitions that are used to render the model instance and produce the links or the other representation customizations.



Note: There are more resource controller samples in the `documentum-rest-<version>.zip/samples/documentum-rest-resource-samples/` SDK.

7.6.3.10 Resource: Programming the View

Views are implementations that customize the resource model instances for further information in the representation, especially for creating links for the resources.

There are three abstract view classes for view implementations: `LinkableView`, `EntryableView`, and `FeedableView`

- `LinkableView`

A non-collection resource (also called single data object resource) must have a `LinkableView` implementation. The extended class implementation must implement the abstract methods. Optionally, the protected methods can be overridden

- `EntryableView`

If the non-collection resource (also called single data object resource) can further be presented into the inline feed of a collection resource. Its view implementation instead should extend `EntryableView`. The extended class implementation must implement the abstract methods. Optionally, the protected methods can be overridden

- `FeedableView`

A collection resource must have a FeedableView implementation. The extended class implementation must implement the abstract methods. Optionally, the protected methods can be overridden

Foundation REST API library provides two view implementations to implement part of the methods. Custom views can extend these classes and reuse their methods.

- com.emc.documentum.rest.view.impl.PersistentLinkableView
- com.emc.documentum.rest.view.impl.DefaultFeedView

There are two view annotations used for the view implementations. A view implementation class must declare one of the following annotations for the corresponding usages.

A @ DataViewBinding annotation is applied to a com.emc.documentum.rest.view.LinkableView or com.emc.documentum.rest.view.EntryableView implementation. The @ DataViewBinding resolves which data model the view definition is bound to. Here is an example.

```
@DataViewBinding(modelType = RelationObject.class)
public class RelationView extends EntryableView<RelationObject> { .. }
```

A @ FeedViewBinding annotation is applied to a com.emc.documentum.rest.view.FeedableView implementation. This annotation resolves inline EntryableView for a feed view. The entry views can be more than one since a feed may contain different types of models. Each entry view should correspond to a unique data model class defined by @ DataViewBinding. Here is an example.

```
@FeedViewBinding(RelationView.class)
public class RelationsFeedView extends FeedableView<RelationObject> { .. }
```



Note: There are more samples for the resource views in the documentum-rest-<version>.zip/samples/documentum-rest-resource-samples/ SDK.

7.6.3.11 Resource: Binding the View and the Controller

As previously mentioned, a resource can be bound to a view implementation by declaring the @ResourceViewBinding Core REST annotation in the resource controller. A @ResourceViewBinding can be applied to a controller at the class level or at the method level.

The annotation has an attribute that you can use to specify which view definitions the resource controller (or method) uses to render the resource model in the controller method. You can bind more than one view definition to a controller or a method (repeating value). However, each view definition must correspond to a unique resource model type, such as feed, document, folder, and so on.

Here are some things to consider when deciding whether to apply the annotation at the class level or at the method level.

- When the annotation is applied at the class level, all methods in the controller by default uses the view definitions in the class level annotation

- When the annotation is applied at a class method level, the specific method in the controller by default uses the view definitions in the method level annotation, and it overrides the class level annotation.

Here is a code sample that shows you how to use the `@ResourceViewBinding` annotation at both class and method levels. The class level supports the feed representation by default, while the create method returns a single relation resource as the Response.

Example 7-69: Using the `@ResourceViewBinding` Annotation

```
@Controller("acme#relations")
@RequestMapping("/repositories/{repositoryName}/relations-x")
@ResourceViewBinding({ RelationsFeedView.class })

public class RelationsController extends AbstractController{
    @RequestMapping(method = RequestMethod.GET)
    @ResponseBody
    @ResponseStatus(HttpStatus.OK)
    public AtomFeed get(
        @RequestParam(value = "name", required = false) final String relationName,
        @RequestUri final UriInfo uriInfo)
        throws MissingServletRequestParameterException, DfException {
        // ...
    }
    @RequestMapping(method = RequestMethod.POST)
    @ResponseBody
    @ResourceViewBinding(RelationView.class)
    public ResponseEntity<RelationObject> createRelation(
        @PathVariable("repositoryName") final String repositoryName,
        @RequestBody final RelationObject relationObject,
        @RequestUri final UriInfo uriInfo)
        throws DfException, MissingServletRequestParameterException {
        // ...
    }
}
```



Notes

- A `@ResourceViewBinding` annotation can bind to multiple `<View>` definitions. When multiple `<View>` definitions are registered for a `@ResourceViewBinding`, each `<View>` definition must correspond to a unique data model. There can be at most one `FeedableView` definition in the `@ResourceViewBinding` definition because `FeedableView` corresponds to the `AtomFeed` data model.
- When `@ResourceViewBinding` annotation is used at the method level, the `@RequestUri final UriInfo uriInfo` declaration is required in the method definition.

Resource controllers, models, and views can be bound with one another by using the Java annotations shown in the following diagram:

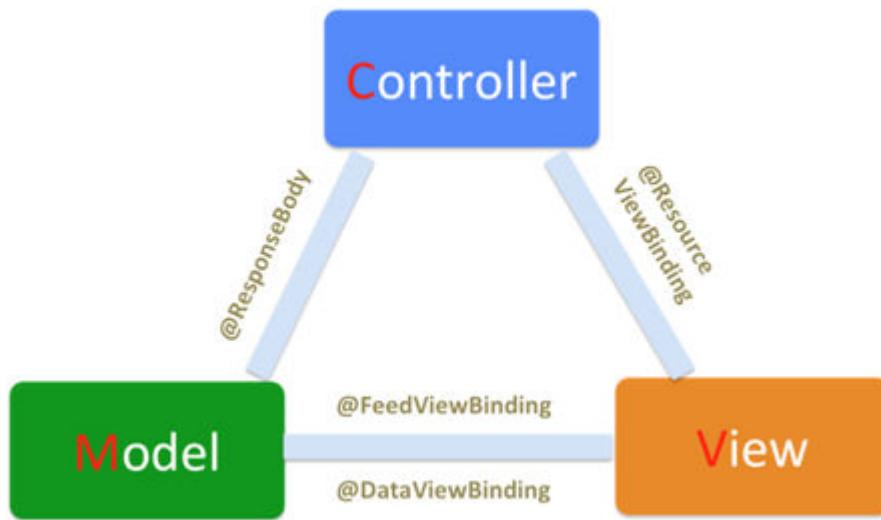


Figure 7-8: Java Annotations and MVC

7.6.3.12 Resource: Customizing Resource Views

Customization involves the following tasks:

1. Write custom view class.
2. Register the class to a resource.

7.6.3.12.1 Customize Resource View

REST extensibility allows you to write your own view implementation classes for a specific data model.

7.6.3.12.1.1 Hide Link Relations

In a custom view, you can call the `<removeLink(..)>` method to hide link relations. You can manually extend from the Core `<View>` implementation when the modification that you want to make is small.

The only method that must be overridden is the `<customize(>` method.

Here's a code sample showing a `<View>` implementation that extends a Core repository `<View>`. In this `<View>` class, the link relation `relation-types` is removed from the repository resource:

Example 7-70: A View that Extends a Core Repository

```

/**
 * Customize repository with extension to remove relation-types link relation
 */
  
```

```

public class RepositoryViewExtend extends RepositoryView {
    public RepositoryViewExtend(Repository repository,
                               UriInfo uriInfo,
                               String repositoryName,
                               boolean returnLinks,
                               Map<String, Object> others) {
        super( repository, uriInfo, repositoryName, returnLinks, others);
    }

    @Override
    public void customize() {
        // Remove relation-types link relation from the repository
        removeLink(CoreLinkRelation.RELATION_TYPES.rel(), null);
    }
}

```



7.6.3.12.1.2 Edit Link Relations

In the custom view, you can call the `<clearLinks()>`, `<makeLink(..)>`, or `<makeLinkIf(..)>` methods to build arbitrary link relations to a resource model. When a significant change is required, you can write a completely new view implementation class for a resource model.

Here are some important points:

- A new view implementation class must extend from `<EntryableView>`, or `<LinkableView>` where all abstract methods must be implemented.
- A new view implementation class must have the `@DataViewBinding` annotation added to the class definition.

In the following code sample of the `<View>` implementation for the *Repository* resource, the links on the *Repository* resource are rebuilt in the View class. Take notice of the following items:

- The link relation *current-user* and *cabinets* are static. This means they are available in any repository resource.
- The link relation *users* is dynamic. This means that only the `<admin>` or `<super user>` can see it.

Here's a code sample that illustrates the preceding points:

► Example 7-71: Rebuilding Links in the View Class

```

/**
 * Customize the repository with totally new view definitions that specify the
 * exact link relations
 */
@DataViewBinding(modelType = Repository.class)
public class RepositoryViewNew extends EntryableView<
Repository>
{
    public RepositoryViewNew(Repository repository, UriInfo uriInfo, String repositoryName,
                           boolean returnLinks, Map<String, Object> others) {
        super(repository, uriInfo, repositoryName, returnLinks, others);
    }
}

```

```
@Override
public String entryTitle() {
    return getDataInternal().getName();
}

@Override
public String entrySummary() {
    return getDataInternal().getDescription();
}

@Override
public Date entryUpdated() {
    return new Date();
}

@Override
public Date entryPublished() {
    return new Date();
}

@Override
public List<
AtomAuthor>
entryAuthors() {
    return Collections.emptyList();
}

@Override
public List<
Link>
entryLinks() {
    return getDefaultEntryLinks();
}

@Override
public void customize() {
    customizeLinks();
}

@Override
public String canonicalResourceUri(boolean validate) {
    return getUriFactory(validate).repositoryUri(getDataInternal().getName(), null);
}

@Override
protected Map<
String, Object>
resolveUriTemplateVariables(Map<String,
                             String> valueMapping) {
    return Collections.emptyMap();
}

private void customizeLinks() {
    // Clear all links
    clearLinks();
    // Build users link relation on condition when the login user is admin or
    // super user
    makeLinkIf(RepositoryContextHolder.getUserPrivileges() > 8,
               CoreLinkRelation.USERS.rel(), getUriFactory().usersUri(null));
    // Build current-user link relation without condition
    makeLink(CoreLinkRelation.CURRENT_USER.rel(),
            getUriFactory().currentUsersUri(null));
    // Build cabinets link relation without condition
    makeLink(CoreLinkRelation.CABINETS.rel(), getUriFactory().cabinetsUri(null));
}
}
```



7.6.3.12.1.3 Hide Properties

When using a custom view, you can also customize what object properties to return the client.

Here's a code sample that shows a view implementation on the cabinet resource model where all internal properties are hidden.

```
/**
 * Customize cabinet resource to remove internal attributes.
 */
public class CabinetViewExtend extends CabinetView {
    public CabinetViewExtend(CabinetObject cabinet, UriInfo uriInfo,
        String repositoryName, boolean returnLinks,
        Map<String, Object> others) {
        super(cabinet, uriInfo, repositoryName, returnLinks, others);
    }

    @Override
    public void customize() {
        // Remove all internal attributes with prefix "i_"
        for (Attribute attr : serializableData.getAttributes()) {
            if (attr.getName().startsWith("i_")) {
                serializableData.removeAttributeIfExisted(attr.getName());
            }
        }
    }
}
```

7.6.3.12.1.4 Customize ATOM Attributes

A custom view also allows you to customize the atom feed and entry attributes on a collection or on a single data object resource.

In the following example of the View implementation on the *Cabinets* feed resource, link relations and feed title are modified.

Example 7-72: Customize the Cabinet Resource to Add New Link Relations

```
/**
 * Customize cabinet resource to add new links and change feed title.
 */
public class CabinetsFeedViewExtend extends CabinetsFeedView {
    public CabinetsFeedViewExtend(Page<CabinetObject> cabinets, UriInfo uriInfo,
        String repositoryName, boolean returnLinks,
        Map<String, Object> others) {
        super(cabinets, uriInfo, repositoryName, returnLinks, others);
    }

    @Override
    public String feedTitle() {
        // Customize feed title
        return "Cabinets in organization: " + getRepositoryName();
    }

    @Override
    public List<Link> feedLinks() {
        List<Link> links = super.feedLinks();
        // Add about link relation to feed links
        links.add(new Link("about", getUriFactory().productInfoUri()));
        return links;
    }
}
```

```
    }  
}
```



The view implementation classes must be built and packaged in a jar file where the final REST war file can access it and load the classes into the class path.

7.6.3.12.2 Register Custom Resource View

The next and last step is to register the custom views to resources. The new view implementations overrides the default views on resources. The configuration is in the same YAML file.

```
---  
# RESOURCE VIEW REGISTRY #  
#####  
resource-view-registry:  
# - resource: repository  
#   view: [org.acme.view.impl.custom.RepositoryViewExtend]
```

Here's a code sample that shows you how to register a previous view for resources.

```
---  
# RESOURCE VIEW REGISTRY #  
#####  
resource-view-registry:  
- resource: repository  
  view: [org.acme.view.impl.custom.RepositoryViewExtend]  
- resource: cabinet  
  view: [org.acme.view.impl.custom.CabinetViewExtend]  
- resource: cabinets  
  view: [org.acme.view.impl.custom.CabinetsFeedViewExtend]
```



Using YAML Config

When multiple views are registered for a resource, each view definition should correspond to a different data model. There could at most be one custom *<FeedableView>* definition for a resource.

Here are some possible error codes or messages for invalid view registry values:

```
E_UNKNOWN_RESOURCE_NAME=Unknown resource name is found: {0}.  
E_RESOURCE_VIEW_DISALLOWED=Preserved core resources {0} are not allowed for view  
  customization. Please review your input resources: {1}  
E_RESOURCE_VIEW_CLASS_NOT_FOUND=The view definition class is not found in class-path: {0}
```

7.6.3.12.2.1 View Loading Precedence

The precedence of the view definition used by a controller method is: YAML defined view > method level @ResourceViewBinding annotation > class level @ResourceViewBinding annotation.

View Precedence

- When a custom `<FeedableView>` is defined on the feed resource, the feed resource uses the custom `<FeedableView>` to render the atom feed. Otherwise, the feed resource uses the default `<FeedableView>` definition on the annotation `@ResourceViewBinding` of the feed resource
- When the feed is inline, the feed resource looks up the `<EntryableView>` definition on the annotation `@FeedViewBinding` of the `<FeedableView>` used by this feed resource

7.6.3.12.2.2 Troubleshooting

After enabling DEBUG level log4j logging for the `emc.emc.documentum.rest` package, the following messages, which have been formatted to fit this document, are printed in the log file:

```
+++++ RESOURCE CUSTOM VIEW PRINT START ++++++
+ -- NAME --          -- VIEW -- +
+ -- cabinet           -- org.acme.view.impl.custom.CabinetViewExtend|
+ -- cabinets          -- org.acme.view.impl.custom.CabinetsFeedViewExtend|org
                           .acme.view.impl.custom.CabinetViewExtend|
+ -- repository        -- org.acme.view.impl.custom.RepositoryViewNew|
+ -- NAME --          -- VIEW -- +
+++++ RESOURCE CUSTOM VIEW PRINT END  ++++++
+++++ RESOURCE DEFAULT VIEW PRINT START ++++++
+ -- NAME --          -- LEVEL --          -- VIEW -- +
+ -- acme#alias-set   -- CLASS            -- org.acme.view.impl
                           .AliasSetView|
+ -- acme#alias-sets  -- CLASS            -- org.acme.view.impl
                           .AliasSetsFeedView|
+ -- acme#alias-sets  -- METHOD:getAliasSets -- org.acme.view.impl
                           .AliasSetsFeedView|
+ -- acme#alias-sets  -- METHOD:createAliasSet -- org.acme.view.impl
                           .AliasSetView2|
+ -- acme#user         -- CLASS            -- com.emc.documentum.rest
                           .view.impl.UserView|
+ -- acme#users        -- CLASS            -- com.emc.documentum.rest
                           .view.impl
                           .UsersFeedView|
+ -- batch-capabilities -- CLASS           -- com.emc.documentum.rest
                           .view.impl
                           .BatchCapabilitiesView|
+ -- cabinet           -- CLASS            -- com.emc.documentum.rest
                           .view.impl.CabinetView|
+ -- cabinets          -- CLASS            -- com.emc.documentum.rest
                           .view.impl
                           .CabinetsFeedView|com
                           .emc.documentum.rest
                           .view.impl.CabinetView|
+ -- cabinets          -- METHOD:createCabinet -- com.emc.documentum.rest
                           .view.impl.CabinetView|
+ -- checked-out-objects -- CLASS           -- com.emc.documentum.rest
                           .view.impl
                           .CheckedOutObjectsFeedV
                           iew|
+ -- child-folder-link -- CLASS           -- com.emc.documentum.rest
```

```

+ -- child-folder-links      -- CLASS           .view.impl
                                         .FolderLinkView|  

                                         -- com.emc.documentum.rest  

                                         .view.impl
                                         .FolderLinksFeedView|  

com                                         .emc.documentum.rest  

                                         .view.impl
                                         .FolderLinkView|  

+ -- child-folder-links      -- METHOD:link    -- com.emc.documentum.rest  

                                         .view.impl
                                         .FolderLinkView|  

+ -- content                  -- CLASS          -- com.emc.documentum.rest  

                                         .view.impl.ContentView|  

+ -- contents                 -- CLASS          -- com.emc.documentum.rest  

                                         .view.impl
                                         .ContentsFeedView|com  

                                         .emc.documentum.rest  

                                         .view.impl.ContentView|  

+ -- contents                 -- METHOD:createContent -- com.emc.documentum.rest  

                                         .view.impl.ContentView|  

+ -- current-user              -- CLASS          -- com.emc.documentum.rest  

                                         .view.impl
                                         .UserView|  

+ -- current-user2             -- CLASS          -- com.emc.documentum.rest  

                                         .view.impl.UserView|  

+ -- current-version           -- CLASS          -- com.emc.documentum.rest  

                                         .view.impl.SysObjectVie  

w|  

+ -- default-folder            -- CLASS          -- com.emc.documentum.rest  

                                         .view.impl.FolderView|  

+ -- document                  -- CLASS          -- com.emc.documentum.rest  

                                         .view.impl.DocumentView  

|  

+ -- dq1-query                 -- CLASS          -- com.emc.documentum.rest  

                                         .view.impl
                                         .QueryResultFeedView|  

com                                         .emc.documentum.rest  

                                         .view.impl
                                         .QueryResultItemView|  

+ -- folder                    -- CLASS          -- com.emc.documentum.rest  

                                         .view.impl.FolderView|  

+ -- folder-child-documents   -- CLASS          -- com.emc.documentum.rest  

                                         .view.impl
                                         .DocumentsFeedView|com  

                                         .emc.documentum.rest  

                                         .view.impl.DocumentView  

|  

+ -- folder-child-documents   -- METHOD:createChildDocument -- com.emc.documentum.rest  

                                         .view.impl.DocumentView  

|  

+ -- folder-child-folders     -- CLASS          -- com.emc.documentum.rest  

                                         .view.impl
                                         .FoldersFeedView|com  

                                         .emc.documentum.rest  

                                         .view.impl.FolderView|  

+ -- folder-child-folders     -- METHOD:createChildFolder  -- com.emc.documentum.rest  

                                         .view.impl.FolderView|  

+ -- folder-child-objects      -- CLASS          -- com.emc.documentum.rest  

                                         .view.impl
                                         .SysObjectsFeedView|com  

                                         .emc.documentum.rest  

                                         .view.impl
                                         .SysObjectView|com.emc  

                                         .documentum.rest.view  

                                         .impl
                                         .ContentfulObjectView|  

+ -- folder-child-objects      -- METHOD:postSysObject   -- com.emc.documentum.rest  

                                         .view.impl
                                         .SysObjectView|com.emc  

                                         .documentum.rest.view

```

```

+-- group                                -- CLASS
+-- group-member-groups                  -- CLASS
+-- group-member-users                   -- CLASS
+-- groups                               -- CLASS
+-- lock                                 -- CLASS
+-- network-location                     -- CLASS
+-- network-locations                   -- CLASS
ew|
+-- object                               -- CLASS
+-- parent-folder-link                  -- CLASS
+-- parent-folder-links                 -- CLASS
com
+-- parent-folder-links                -- METHOD:link
+-- product-information                -- CLASS
+-- relation                            -- CLASS
|
+-- relation-type                      -- CLASS
+-- relation-types                     -- CLASS
+-- relations                           -- CLASS
|
+-- relations                          -- METHOD:createRelation
|
+-- repositories                       -- CLASS
+-- repository                         -- CLASS
+-- search                             -- CLASS
                                         .impl
                                         .ContentfulObjectView|
                                         -- com.emc.documentum.rest
                                         .view.impl.GroupView|
                                         -- com.emc.documentum.rest
                                         .view.impl
                                         .GroupsFeedView|
                                         -- com.emc.documentum.rest
                                         .view.impl
                                         .UsersFeedView|
                                         -- com.emc.documentum.rest
                                         .view.impl
                                         .GroupsFeedView|
                                         -- com.emc.documentum.rest
                                         .view.impl
                                         .SysObjectView|
                                         -- com.emc.documentum.rest
                                         .view.impl
                                         .NetworkLocationView|
                                         -- com.emc.documentum.rest
                                         .view.impl
                                         .NetworkLocationsFeedVi
                                         -- com.emc.documentum.rest
                                         .view.impl
                                         .SysObjectView|com.emc
                                         .documentum.rest.view
                                         .impl
                                         .ContentfulObjectView|
                                         -- com.emc.documentum.rest
                                         .view.impl
                                         .FolderLinkView|
                                         -- com.emc.documentum.rest
                                         .view.impl
                                         .FolderLinksFeedView|
                                         .emc.documentum.rest
                                         .view.impl
                                         .FolderLinkView|
                                         -- com.emc.documentum.rest
                                         .view.impl
                                         .FolderLinkView|
                                         -- com.emc.documentum.rest
                                         .view.impl
                                         .ProductInfoView|
                                         -- com.emc.documentum.rest
                                         .view.impl.RelationView
                                         -- com.emc.documentum.rest
                                         .view.impl
                                         .RelationTypeView|
                                         -- com.emc.documentum.rest
                                         .view.impl
                                         .RelationTypesFeedView|
                                         -- com.emc.documentum.rest
                                         .view.impl
                                         .RelationsFeedView|com
                                         .emc.documentum.rest
                                         .view.impl.RelationView
                                         -- com.emc.documentum.rest
                                         .view.impl.RelationView
                                         -- com.emc.documentum.rest
                                         .view.impl
                                         .RepositoriesFeedView|
                                         -- com.emc.documentum.rest
                                         .view.impl
                                         .RepositoryView|
                                         -- com.emc.documentum.rest
                                         .search.representation

```

```

+ -- type          -- CLASS
+ -- types         -- CLASS
+ -- user          -- CLASS
+ -- users         -- CLASS
+ -- versions      -- CLASS

+ -- versions      -- METHOD:checkIn
+ -- versions      -- METHOD:checkInContent
+ -- versions      -- METHOD:checkInMetadata

+ -- NAME --       -- LEVEL --       -- VIEW -- +
++++++ RESOURCE DEFAULT VIEW PRINT END     +++++++

++++++ QUERYABLE VIEW PRINT START ++++++
+ -- TYPE --       -- VIEW -- +
+ -- dm_alias_set   -- com.emc.documentum.rest.view.impl.UserView|
+ -- dm_cabinet     -- com.emc.documentum.rest.view.impl.CabinetView|
+ -- dm_document    -- com.emc.documentum.rest.view.impl.DocumentView|
+ -- dm_folder      -- com.emc.documentum.rest.view.impl.FolderView|
+ -- dm_group       -- com.emc.documentum.rest.view.impl.GroupView|
+ -- dm_network_location_map
+ -- dm_relation    -- com.emc.documentum.rest.view.impl.RelationView|
+ -- dm_relation_type
+ -- dm_sysobject   -- com.emc.documentum.rest.view.impl.SysObjectView|
+ -- dm_type         -- com.emc.documentum.rest.view.impl.TypeView|
+ -- dm_user         -- com.emc.documentum.rest.view.impl.UserView|
+ -- dmi_dd_type_information
+ -- dmr_content    -- com.emc.documentum.rest.view.impl.TypeView|
+ -- TYPE --        -- VIEW -- +
++++++ QUERYABLE VIEW PRINT END     ++++++

```

7.6.3.13 Resource: Building Resource links

The abstract view classes `EntryableView`, `LinkableView`, and `FeedableView` provide fundamental links for single object resources and feed resources. A custom resource view class extending one of these abstract classes can override or extend the links provided by the abstract class.

LinkableView

By default, the class `LinkableView` returns the `self` link for the resource model in the protected method `xselfLinks()`. The class provides several abstract or protected methods for overriding.

- To customize `self` link generation, you can override the method `selfLinks()`.
- To add additional links, you can implement the method `customize()`, and call `makeLink()`, `makeLinkIf()`, `makeLinkTemplate()`, or `makeLinkTemplateIf()` in the `customize()` method.
- To remove a specific link, you can call `removeLink()` in the `customize()` method.
- To clear all default links, you can call `clearLinks()` in the `customize()` method.

The method `makeLinkIf()` adds a link relation to the resource only when the condition is met at runtime. This method is useful for adding link relations upon certain conditions.

In the following sample implementation of the `customize()` method, we add an additional link relation 'author' to the alias set resource upon the existence of the `owner_name` attribute.

```
@Override
public void customize() {
    // Add the author link relation
    makeLinkIf(
        serializableData.getAttributeByName("owner_name") != null,
        LinkRelation.AUTHOR.rel(),
        ResourceUriBuilder
            .onResource("user")
            .pathVariables((String)serializableData.getAttributeByName("owner_name")))
}
```

EntryableView

The `EntryableView` class extends `LinkableView` with additional atom entry customizations. Besides the link methods that the `LinkableView` class provides, you can use the `entryLinks()` method to define your own entry links in the atom entry representation. When your custom resource view class extends the `PersistentLinkableView` class, a default link relation called `edit` is added to the entry link collection.

FeedableView

By default, the `FeedableView` class returns the `self` link and pagination links (`first`, `next`, `previous`, or `last` depending on the current page position) for the atom feed

model. You can override the `feedLinks()` method to modify these link relations or add other link relations.

In the following code sample, we override the `feedLinks()` method to add an additional link relation called `about` to the alias set feed resource:

```
@Override  
public List<Link> feedLinks() {  
    List<Link> links = super.feedLinks();  
    // Add the 'about' link relation to feed links  
    links.add(new Link("about", ResourceUriBuilder.onResource("product-info")));  
    return links;  
}
```

7.6.3.14 Resource: Building Resource URIs with ResourceUriBuilder

In release 7.2, users had to call `<com.emc.documentum.rest.http.UriFactory>` to build links for core resources or custom resources. For links of custom resources, you must call `<com.emc.documentum.rest.http.UriFactory#buildUriByTemplateName()>` to build links from YAML URI template information.

In 7.3 release, a new API `<com.emc.documentum.rest.context.ResourceUriBuilder>` was introduced to build resource links in a consistent way. The detailed API description can be found in the SDK JavaDoc.

Here's a code sample of the resource URI builder:

```
// Build user resource URI: <base uri context>/repositories/  
// <repoName>/users/dmadmin  
String href = ResourceUriBuilder  
    .onResource("user")  
    .pathVariable(true, "dmadmin")  
    .build();  
  
// Build query resource URI as hreftemplate: <base uri context>/  
// repositories/<repoName>/{?q}  
String href = ResourceUriBuilder  
    .onResource("query")  
    .asTemplate("q")  
    .build();  
  
// Build cabinets resource URI with query: <base uri context>/  
// repositories/<repoName>/cabinets?inline=true&view=:all  
String href = ResourceUriBuilder  
    .onResource("cabinets")  
    .QueryParam("view", ":all")  
    .QueryParam("inline", "true")  
    .build();  
  
// Build custom alias-set resource URI by resource name: <base uri context>/  
// repositories/<repoName>/alias-set/1234?view=:all  
String href = ResourceUriBuilder  
    .onResource("acme#alias-set")  
    .pathVariables("1234")  
    .QueryParam("view", ":all")  
    .build();  
  
// Build custom alias-set resource URI by resource controller: <base uri context>/  
// repositories/<repoName>/alias-set/1234?view=:all  
String href = ResourceUriBuilder  
    .onResource(org.acme.rest.AliasSetController.class)  
    .pathVariables("1234")  
    .QueryParam("view", ":all")  
    .build();
```

```
// Build custom alias-set resource URI by custom URI template: <base uri context>/  
// repositories/<repoName>/alias-set/1234?view=:all  
String href = ResourceUriBuilder  
    .onTemplate("X_ALIAS_SET_URI_TEMPLATE")  
    .pathVariables("1234")  
    .queryParam("view", ":all")  
    .build();
```

At runtime, the <base uri context> and the format extension are resolved automatically. In unit testing, you can specify the base URI context and format extension to verify the full path. Here's a code sample that shows you how to do that:

```
UriInfo uriInfo = new UriInfo();  
uriInfo.setBaseUri("http://localhost:8080/dctm-rest");  
uriInfo.setFormatExtension("");  
String href = ResourceUriBuilder.onResource("user")  
    .repository("ACME")  
    .uriInfo(uriInfo)  
    .pathVariable(true, "dmadmin")  
    .build();  
assertEquals("http://localhost:8080/dctm-rest/repositories/ACME/users/  
dmadmin", href);
```

When implementing a resource view, you can use the newly added methods on the abstract views to get the URI resource builder. Here's a listing of those methods:

- com.emc.documentum.rest.view.LinkableView#uriBuilder(string)
- com.emc.documentum.rest.view.LinkableView#idUriBuilder(string, string)
- com.emc.documentum.rest.view.LinkableView#nameUriBuilder(string, string)
- com.emc.documentum.rest.view.FeedableView#uriBuilder(string)

7.6.3.15 Resource: Making It Queryable

The DQL query resource returns a collection of query result items as atom entries. The entry can contain a link pointing to the canonical resource of the typed object if there is a resource implementation for the specified object type.

To make the custom resource linkable from the DQL query resource, add query types on the controller annotation `@ResourceViewBinding` for your custom resource controller, as shown in the following sample:

```
@Controller("acme#alias-set")  
@RequestMapping("/repositories/{repositoryName}/alias-sets/{aliasSetId}")  
// make query results of type "dm_alias_set" linkable to the acme#alias-set resource  
@ResourceViewBinding(value = AliasSetView.class, queryTypes = "dm_alias_set")  
public class AliasSetController extends AbstractController { ... }
```



Note: The *OpenText Documentum Content Management - Foundation REST API Reference Guide (EDCPKRST250400-PRE)* introduces the detail rules for all of the DQL expressions that can produce resource links.

7.6.3.16 Resource: Deciding Whether to Be Batchable

By default, custom resources can be executed in the batch. You will find all custom resources in the batchable-resource list when you send a GET request to the Batch Capabilities resource. However, in some scenarios, you may not want to make a custom resource batchable. To help you manage the batch property of a custom resource or even a specific method of the custom resource, Foundation REST API provides the following two annotations:

- com.emc.documentum.rest.model.batch.annotation.BatchProhibition
 - com.emc.documentum.rest.model.batch.annotation.TransactionProhibition
- @BatchProhibition*

This annotation prevents client applications from embedding a custom resource in batch requests. To use the *@BatchProhibition* annotation, add it to the controller class of this resource as shown in the following code snippet:

```
@Controller  
@BatchProhibition  
public class MyResourceController {...}
```

When the controller class of a custom resource is annotated with *@BatchProhibition*, the resource is moved to the non-batchable-resources list of the Batch Capabilities resource.

Besides of adding *@BatchProhibition* to the class level, you can add this annotation to one or more methods in the controller class to prevent certain operations of the resource from being embedded to batch requests.

```
@Controller  
public class MyResourceController {  
    //...  
  
    @RequestMapping (  
        value = {" /users/{userName}/MyResources/{MyResourceId}"},  
        method = RequestMethod.GET, produces = {  
            SupportedMediaTypes.APPLICATION_VND_DCTM_JSON_STRING,  
            SupportedMediaTypes.APPLICATION_VND_DCTM_XML_STRING,  
            MediaType.APPLICATION_JSON_VALUE,  
            MediaType.APPLICATION_XML_VALUE})  
  
    @ResponseBody  
    @ResponseStatus(HttpStatus.OK)  
    @BatchProhibition  
    public MyResource getMyResource (...){...}  
  
    @RequestMapping(  
        value = {" /users/{userName}/MyResources/{MyResourceId}"},  
        method = RequestMethod.DELETE)  
  
    @ResponseBody  
    @ResponseStatus(HttpStatus.NO_CONTENT)  
    public void deleteMyResource(...){...}  
  
    //...  
}
```

This code snippet prevents the GET operation on *MyResource* from being embedded in batch requests. However, you can still perform a bulk delete on *MyResource* in a

batch request because neither the `MyResourceController` class nor the `deleteMyResource` method is annotated with `@BatchProhibition`.

`@TransactionProhibition`

The `@TransactionProhibition` annotation enables you to remove the transaction support from a custom resource. Typically, you add the `@TransactionProhibition` annotation to the controller class of this resource as shown in the following code snippet:

```
@Controller
@TransactionalProhibition
public class MyResourceController {...}
```

This setting prevents the custom resource from being embedded in transactional batch requests.

Similar to `@BatchProhibition`, the `@TransactionProhibition` annotation can also be added to one or more methods in the controller class to prevent certain operations of the resource from being embedded to transactional batch requests.

Note that the `@TransactionProhibition` annotation only affects custom resources' applicability in transactional batch requests. To prevent a custom resource from being embedded in all batch requests, use `@BatchProhibition`.

For more information about batch requests, see the Batch and Batch Capabilities sections in the *OpenText Documentum Content Management - Foundation REST API Reference Guide (EDCPKRST250400-PRE)*.

7.6.3.17 Resource: Extending Atom Feed and Entry

The custom resource that returns the object collection as a feed may need to add some additional attributes to the atom feed or the atom entry. This can be done by extending the model classes `com.emc.documentum.rest.model.AtomFeed` and `com.emc.documentum.rest.model.AtomEntry`, and then creating custom views for them.

The extended feed model should extend `<com.emc.documentum.rest.model.AtomFeed>` and specify the `@SerializableType` attribute `inheritValue` as true. Here is an example.

```
@SerializableType(inheritValue = true,
    xmlNS = "http://www.w3.org/2005/Atom", xmlNSPrefix = "atom")
public class AliasSetFeed extends AtomFeed {
    @SerializableField(xmlNS = "http://ns.acme.com/", xmlNSPrefix = "acme")
    private int score;

    public int getScore() {
        return score;
    }
}
```

This sample model contains a custom feed attribute 'score' but will still be marshaled as the same feed root `<feed ...>` for XML.

If the custom attributes are added to the atom entry, the view must extend `com.emc.documentum.rest.model.AtomEntry`, too. Here is the sample for the extended atom entry.

```
@SerializableType(inheritValue = true, xmlNS = "", xmlNSPrefix = "atom")
public class AliasSetEntry extends AtomEntry {
    @SerializableField("own-by-login-user")
    private boolean ownByLoginUser;

    public boolean isOwnByLoginUser() {
        return ownByLoginUser;
    }
    public void setOwnByLoginUser(boolean ownByLoginUser) {
        this.ownByLoginUser = ownByLoginUser;
    }
}
```

After the model definition, you need to create custom views for the extended feed and entry to populate the custom attribute data. For a feed view definition, the class must explicitly declare its binding feed type as the extended feed type. Here is the feed view example.

```
@FeedViewBinding(value = AliasSetViewX.class, feedType = AliasSetFeed.class)
public class AliasSetsFeedViewX extends FeedableView <AliasSet> {
    public AliasSetsFeedViewX(Page<AliasSet> page, UriInfo uriInfo,
        String repositoryName, boolean returnLinks,
        Map<String, Object> others) {
        super(page, uriInfo, repositoryName, returnLinks, others);
    }
    @Override
    public String feedTitle() {
        return "Alias Sets";
    }
    @Override
    public Date feedUpdated() {
        return new Date();
    }
    @Override
    protected void customizeFeed(AtomFeed feed) {
        AliasSetFeed aliasSetFeed = (AliasSetFeed) feed;
        aliasSetFeed.setScore(new SecureRandom().hashCode());
    }
}
```

For an entry view definition, it must explicitly declare the binding entry type is the extended atom entry type. Here is the entry view example.

```
@DataViewBinding(queryTypes = "dm_alias_set",
modelType = AliasSet.class, entryType = AliasSetEntry.class)
public class AliasSetViewX extends PersistentLinkableView <AliasSet> {
    public AliasSetViewX(AliasSet aliasSet, UriInfo uriInfo,
        String repositoryName, boolean returnLinks,
        Map<String, Object> others) {
        super(aliasSet, uriInfo, repositoryName, returnLinks, others);
    }
    ...
    @Override
    protected void customizeEntry(AtomEntry atomEntry) {
        AliasSetEntry aliasSetEntry = (AliasSetEntry) atomEntry;
        aliasSetEntry.setOwnByLoginUser(
            RepositoryContextHolder.getUserName().equals
                (getDataInternal().getAttributeByName("owner_name")));
    }
}
```

Last, in the resource controller, the implementation is same to non-extended atom feed model. The actual returning instance extends atom feed. Here is the controller example.

```
@Controller("acme#alias-sets")
@RequestMapping("/repositories/{repositoryName}/alias-sets")
public class AliasSetCollectionController extends AbstractController {
    @RequestMapping(
        method = RequestMethod.GET,
        produces = {
            SupportedMediaTypes.APPLICATION_VND_DCTM_JSON_STRING,
            MediaType.APPLICATION_ATOM_XML_VALUE,
            MediaType.APPLICATION_JSON_VALUE,
            MediaType.APPLICATION_XML_VALUE
        })
    @ResponseBody
    @ResponseStatus(HttpStatus.OK)
    @ResourceViewBinding(AliasSetsFeedViewX.class)
    public AliasSetFeed getAliasSets(
        @PathVariable("repositoryName") final String repositoryName,
        @TypedParam final CollectionParam param,
        @RequestUri final UriInfo uriInfo)
        throws Exception {
        PagedQueryTemplate template = new AliasSetCollectionQueryTemplate()
            .filter(param.getFilterQualification())
            .order(param.getSortSpec().get())
            .page(param.getPagingParam().getPage(),
                param.getPagingParam().getItemsPerPage());
        PagedDataRetriever<AliasSet> dataRetriever = new
        PagedPersistentDataRetriever<AliasSet> (
            template,
            param.getPagingParam().getPage(),
            param.getPagingParam().getItemsPerPage(),
            param.getPagingParam().isIncludeTotal(),
            param.getAttributeView(),
            AliasSet.class);
        return (AliasSetFeed) getRenderedPage (
            repositoryName,
            dataRetriever.get(),
            param.isLinks(),
            param.isInline(),
            uriInfo,
            null);
    }
}
```

Using a Generic<AtomEntry>: The <AtomFeed> can have a generic <AtomEntry> entry type defined. Here's a code sample that illustrates how to do this:

```
@SerializableType(value = "feed", ...)
public class AtomFeed<T extends AtomEntry> {
    private String id;
    private String title;
    ...
    private List<T> entries;

    // To make this compatible with previous versions,
    // the getEntries() and setEntries() methods still use an
    // explicit List<AtomEntry>
    public List<AtomEntry> getEntries() {}
    public void setEntries(List<AtomEntry> entries) {}

    // For better leveraging of the generic T type,
    // two other similar functions are added, which use a T parameter
    public List<T> getGenericEntries() {}
    public void setGenericEntries(List<T> entries) {}
}
```

When you extend from `<AtomFeed>`, the new entry type must be provided, in addition to adding new fields.

```
@SerializableType(value = "feed", ...)  
public class CustomizedAtomFeed extends AtomeFeed<CustomizedAtomEntry> {  
    private String customFeedField;  
}
```

Now, `<CustomizedAtomFeed>` and `<CustomizedAtomEntry>` can be used to serialize and deserialize at both the server and client side.

A small compatibility issue exists in the 7.2 code. Code that is currently working with the 7.2 version of `<AtomFeed>`, `<AtomEntry>`, or both, may code as shown here:

```
AtomFeed feed = ...  
// Iterate the entries  
for(AtomEntry entry : feed.getEntries()) { //compile error  
    ...  
}  
  
// Or get the entry by index directly  
feed.getEntries().get(0).getContent(); //compile error
```

Due to a limitation of Java generics, the new version of `<AtomFeed>` in version 7.3 code has compilation issues. Since `<AtomFeed>` is now defined as a Java generic class, using it as a normal class causes its generic information to disappear. The return type of `<feed.getEntries(>)` becomes `List<Object>` instead of `List<AtomEntry>` as expected.

To remedy this issue, you can use generic information with `<AtomFeed>` or you can explicitly cast the return type of `<feed.getEntries(>)` to `List<AtomEntry>`. Here's a code sample that illustrates this:

```
AtomFeed<AtomEntry> feed = ...  
  
// Iterate the entries  
for(AtomEntry entry : (List<AtomEntry>)feed.getEntries()) {  
    ...  
}  
  
// Or get the entry directly by index  
((List<AtomEntry>)feed.getEntries()).get(0).getContent
```

7.6.3.18 Resource: Error Handling and Representation

Foundation REST API leverages the Spring framework to resolve the error mapping from Java exceptions to error messages. The Foundation REST API SDK library provides the default implementation `com.emc.documentum.rest.error.http.GeneralExceptionMapping` that defines a lot of error mappings for various Java exception classes. If you want to define your own error mappings in custom resources, you must extend `GeneralExceptionMapping` and override the following bean with the custom class, which is in the `BaseMarshallingConfig` class of the `rest-api-mvc-resource` jar file.

```
@Bean(name = "generalExceptionMapping")  
public GeneralExceptionMapping generalExceptionMapping() {  
    return new GeneralExceptionMapping();  
}
```

The extended exception mapping class adds more methods for the exception mapping with Spring annotation `@Exceptionhandler`. The output of the method must be an instance of `com.emc.documentum.rest.model.RestError`. Here is an example of the default error mapping for `ConversionFailedException`.

```
@ResponseBody
@ResponseStatus(HttpStatus.BAD_REQUEST)
@ExceptionHandler(ConversionFailedException.class)
public RestError onConversionFailedException(ConversionFailedException e) {
    return buildRestError(new GenericExceptionConverter(e,
        HttpStatus.BAD_REQUEST, "E_INPUT_ILLEGAL_ARGUMENTS"));
}
```

7.6.3.19 Resource: Consuming Multipart Contents in Custom Resource Controller

You can define an `Iterator<Part>` controller parameter for the Request body of custom resources and annotate your definition with the `@RequestBody` annotation. Doing this gives you access to all parts of the Request. Here is a code sample that shows you how to read the multipart input, and write back to the Response.

Example 7-73: Read Multipart Input and Write to Response

```
@RequestMapping(method = RequestMethod.POST,
    consumes = { "multipart/form-data", "multipart/mixed" })
@ResponseStatus(HttpStatus.OK)
public void processMultipart(@RequestBody final Iterator<Part> parts)
    throws DfException, MissingServletRequestParameterException, IOException {
    if(parts != null) {
        Response.setContentType(MediaType.TEXT_PLAIN.toString());
        OutputStream out = Response.getOutputStream();
        int p = 1;
        while(parts.hasNext()) {
            Part part = parts.next();
            for(String header : part.getHeaderNames()) {
                //write the part header to the Response directly
                out.write((header + "=" + part.getHeader(header)) + "\r\n").getBytes();
            }
            //write the part content to the Response
            IOUtils.copy(part.getInputStream(), out);
            out.write("\r\n-----end-----\r\n".getBytes());
            out.flush();
        }
    }
}
```

Each part of the multipart contents contains Headers and the content stream. These parts can be read one by one using standard Iterator methods such as `hasNext` and `next`. All parts of the multipart contents must be read sequentially because they are all in one multipart stream, and the Foundation REST API server does not cache any part of the stream.

7.6.4 Linking Custom and Core Resources

In this phase, you will make the newly-created resources discoverable from existing core REST resources or the Home Document via HATEOAS links.

To make the custom REST services hypermedia driven, there must establish link relations between Core resources and custom resources. Foundation REST API SDK provides an approach to add link relations to Core resources for the custom links. Refer to the section [Adding Links to Core Resources](#) for the details.

7.6.5 Packaging and Deployment

In this phase, you will package all your source files to build up a WAR file and then deploy the WAR file to your application server.

For Maven users, the custom resource project can leverage the Maven war overlay plugin to repackage the REST war file by bundling both Core resources and custom resources into the single WAR file. Foundation REST API SDK provides the sample Maven pom file in its Maven archetype project to illustrate how to build the custom resource WAR. The war overlay plugin configuration looks similar to the following.

```
<build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-war-plugin</artifactId>
<version>2.4</version>
<configuration>
<overlays>
<overlay>
<groupId>com.emc.documentum.rest</groupId>
<artifactId>documentum-rest-web</artifactId>
<excludes />
</overlay>
</overlays>
</configuration>
</plugin>
</plugins>
</build>
```

With the new custom resource WAR file, both Core resources and custom resources are running in the same server box and can work together smoothly. In your development environment, you can enable DEBUG level in log4j for the package `emc.emc.documentum.rest`. All resource controller and view information is printed out in the log file or on the server console.

7.7 Working with YAML configuration

The YAML file `rest-api-custom-resource-registry.yaml`, which is located in `dctm-rest.war/WEB-INF/classes/com/emc/documentum/rest/script`, provides configurations for users to customize Core REST resources. You can also customize the YAML location within `dctm-rest.war/WEB-INF/classes/rest-api-runtime.properties`, as follows:

 **Example 7-74: Setting the Package for a Custom YAML File**

```
# Sets the package for custom yaml file.
rest.ext.yaml.package=
```



You are free to rename the YAML file, but the file extension must remain `.yaml`, for example `my-custom-rest.yaml`. The `rest-api-runtime.properties.template` file, which is located in the same location, has more details on the configuration.

7.8 Working with HAL

7.8.1 Overview

The Hypertext Application Language (HAL) is a JSON media type that can be used to develop Web APIs that are consumed using a series of links. Clients that consume these APIs can select hyperlinks, and by using their link relation type, can easily move through the application as needed.

There are two HAL properties that are reserved, they are the `_links` and the `_embedded` properties. These two reserved properties are the key difference between the JSON (`application/vnd.emc.documentum+json`) media type and the HAL (`application/hal+json`) media type.

7.8.1.1 The `_links` reserved property

In contrast to the properties of the `application/vnd.emc.documentum+json` media type, the HAL `_links` property uses a link relation as its key.

Therefore when there is:

- One link specified for the link relation, then the value of that link's key is the JSON object.
- More than one link specified for the link relation, then the values of that link are stored in a JSON array. Each link is one element of the JSON array object and each link's title property is used to distinguish different links.

The `_links` reserved property uses link relations to expose links to other resources. The link relation may link to a link object or to an array of link objects. The link object has the following properties that are used by Foundation REST API:

- href
- templated
- title
- type

The definitions for each of the object's link relation properties is found in the HAL specification at IETF Tools website.

 **Example 7-75: The _links property**

HAL

```
{  
  "...",  
  "_links": {  
    "self": {  
      "href": "xxx"  
    },  
    "the_link_rel_with_title" : [  
      {  
        "title": "title_1",  
        "href": "xxx"  
      },  
      {  
        "title": "title_2",  
        "href": "xxx"  
      }  
    ]  
  }  
}
```

JSON

```
{  
  "...",  
  "links": [  
    {  
      "href": "xxx",  
      "rel": "self"  
    },  
    {  
      "href": "xxx",  
      "title": "title_1",  
      "rel": "the_link_rel_with_title"  
    },  
    {  
      "href": "xxx",  
      "title": "title_2",  
      "rel": "the_link_rel_with_title",  
    }  
  ]  
}
```



7.8.1.2 The `_embedded` reserved property

The `_embedded` property contains embedded resources, which are used with Foundation REST API collection type resources. The `collection` keyword is used to define the collection that is used in all of the Foundation REST API collection responses.

 **Example 7-76: The `_embedded` property**

HAL

```
{
  "_embedded": {
    "collection": [
      ...
    ]
  }
}
```

JSON

```
{
  "entries": [
    ...
  ]
}
```



7.8.2 Using HAL with a single resource

The HAL media type can be used with single or collection type resources. We will discuss each of these uses in the following sections.

The HAL representation for a single resource is similar to the original `application/vnd.emc.documentum+json` media type. However, while the resource's properties are not changed, the links for that resource are put within the `_links` element as required by the HAL specification.

 **Example 7-77: Using HAL with a single resource**

HAL

```
{
  "other_properties": "...",
  "_links": [
    {
      "self": {
        "href": ...
      }
    }
  ]
}
```

JSON

```
{
  "other_properties": "xxx",
  "links": [
    ...
  ]
}
```

```
        {
          "rel": "self",
          "href": "xxx"
        }
    ]  
}
```



A request with Content-Type=application/hal+json has the same format as a response from a GET operation.

7.8.3 Using HAL with a collection resource

The format of collection type resources is different depending on the *inline* URL parameter being *true* or *false*. When the *inline* URL parameter is *false*, only basic information, such as the <*id*>, <*title*>, and <*summary*> are contained in the format of the resource. However, when the *inline* URL parameter is *true*, the details of each single item are contained in the `content` element.

When using the POST method to create instances for a resource, a request with Content-Type=application/hal+json has the same format as a request used for its single resource.

Example 7-78: Non-inline content for the application/hal+json media type

In the HAL media type shown in the following code sample, the `src` attribute is removed and its value is merged into the `_links` element. The following code sample shows you the Response from a collection resource without inline items.

HAL

```
{
  "id": "xxx",
  "title": "xxx",
  "_embedded": {
    "collection": [
      {
        "id": "xxx",
        "title": "xxx",
        "_links": {
          "self": {
            "href": "xxx"
          }
        }
      }
    ],
    "_links": {
      "self": {
        "href": "xxx"
      }
    }
  }
}
```

JSON

```
{
  "id": "xxx",
  "title": "xxx",
  "author": [
    {
      "name": "Open Text Documentum"
    }
  ],
  "updated": "2018-05-02T07:26:34.311+00:00",
  "page": 1,
  "items-per-page": 10,
  "total": 100,
  "entries": [
    {
      "id": "xxx",
      "title": "xxx",
      "summary": "xxx",
      "published": "xxx",
      "updated": "xxx",
      "author": [
        {
          "name": "Open Text Documentum"
        }
      ],
      "links": [
        {
          "rel": "edit",
          "href": "xxx"
        }
      ],
      "content": {
        "type": "application/vnd.emc.documentum+json",
        "src": "xxx"
      }
    }
  ],
  "links": [
    {
      "rel": "self",
      "href": "xxx"
    }
  ]
}
```

Notice how there is no content element.



➡ Example 7-79: A collection resource with Inline contents

For inline representation of collection resources, the content is put into the content keyword element of each object in the collection resource.

The following code sample shows you the format of a response from a collection resource where its contents is inline (its *inline URL* parameter is *true*). The content keyword element with its inline property items has been emphasized in bold.

HAL

```
{
  "id": "xxx",
  "title": "xxx",
```

```
"author": [
  {
    "name": "Open Text Documentum"
  }
],
"updated": "2018-05-02T07:26:34.311+00:00",
"page": 1,
"items-per-page": 10,
"total": 100,
"_embedded": {
  "collection": [
    {
      "id": "xxx",
      "title": "xxx",
      "updated": "xxx",
      "published": "xxx",
      "summary": "xxx",
      "content": {"xxx": "xxx"},
      "_links": {
        "self": {"href": "xxx"}
      }
    }
  ],
  "_links": {
    "self": {"href": "xxx"}
  }
}
```

JSON

```
{
  "id": "xxx",
  "title": "xxx",
  "author": [
    {
      "name": "Open Text Documentum"
    }
  ],
  "updated": "2018-05-02T07:26:34.311+00:00",
  "page": 1,
  "items-per-page": 10,
  "total": 100,
  "entries": [
    {
      "id": "xxx",
      "title": "xxx",
      "summary": "xxx",
      "published": "xxx",
      "updated": "xxx",
      "author": [
        {
          "name": "Open Text Documentum"
        }
      ],
      "links": [
        {
          "rel": "edit",
          "href": "xxx"
        }
      ],
      "content": {
        "xxx": "xxx",
        "links": [
          {
            "rel": "edit",
            "href": "xxx"
          }
        ]
      }
    }
  ]
}
```

```

        }
    ],
    "_links": [
        {
            "rel": "self",
            "href": "xxx"
        }
    ]
}

```

As you can see in the preceding example, the inline item is embedded into the `content` element. However, there is no `_links` in the `content`. Instead, all links are put into the entry level `_links` element.



7.8.4 Updated Foundation REST API resources

A number of Foundation REST API resources have been updated to support the use of the HAL media type. The following are the resources that have been updated.

7.8.4.1 The Services resource

In addition to the `application/home+json` and `application/home+xml` media types, the Services resource supports the `application/hal+json` media type. The following code sample shows you a response from the Services resource.

Example 7-80: The Services resource

```
{
    "resources": {
        "http://identifiers.emc.com/linkrel/repositories": {
            "href": "http://localhost:8080/dctm-rest/repositories",
            "hints": {
                "allow": ["GET"],
                "representations": [
                    "application/xml",
                    "application/json",
                    "application/vnd.emc.documentum+json",
                    "application/atom+xml",
                    "application/hal+json"
                ]
            }
        },
        "about": {
            "href": "http://localhost:8080/dctm-rest/product-info",
            "hints": {
                "allow": ["GET"],
                "representations": [
                    "application/xml",
                    "application/json",
                    "application/vnd.emc.documentum+json",
                    "application/hal+json",
                    "application/vnd.emc.documentum+xml"
                ]
            }
        }
    }
}
```



7.8.4.2 Folder Child Objects, Folder Child Documents, and All Versions resources

When the Request is a multipart request, depending on the media type of the first metadata part, the content type can be any one of the following:

- application/vnd.emc.documentum+xml
- application/vnd.emc.documentum+json
- application/hal+json

The following code sample shows you how to make a multipart request using the application/hal+json Content-Type.

```
POST http://localhost/dctm-rest/repositories/myrepo/objects/0c00000c80000105/objects
Content-Type: multipart/form-data; boundary=314159265358979

--314159265358979
Content-Disposition: form-data; name=metadata
Content-Type: application/hal+json

{"properties": {"object_name": "rest-api-test"}}
--314159265358979
Content-Disposition: form-data; name=binary1
Content-Type: text/plain

This is primary content
--314159265358979--
```

7.8.4.3 The Batches resource

Batch without attachments

When you use the Batch resource without an attachment and the batch operation has the Content-Type header defined, its value element can be application/hal+json. The following code sample shows you how to use the Batch resource to make a request without attachments:

```
{
  "operations": [
    {
      "id": "id-001",
      "request": {
        "method": "POST",
        "uri": "/<base URI>/objects",
        "headers": [
          {
            "name": "Content-Type",
            "value": "application/hal+json"
          }
        ],
        "entity": "{\"properties\":{\"object_name\":\"my test object\"}}"
      }
    }
  ]
}
```

Batch with Attachments

When you make a request using `application/hal+json` as the metadata, then the `start-info` of the request `content-type` should also be `application/hal+json` and the content type of the metadata part is `application/jop+json; type="application/hal+json"`.

The following code sample shows you how to use the Batch resource to make a multipart request that uses the `application/hal+json` media type and has attachments:

```
POST http://localhost:8080/dctm-rest/repositories/acme/batches
Content-Type: Multipart/Related;boundary=frontier;type="application/jop+json";
               start="batch";start-info="application/hal+json"
Accept: application/hal+json

--frontier
Content-Type: application/jop+json; type="application/hal+json"
Content-ID: batch
Content-disposition: form-data; name=batch
{
  "operations" :
  [
    {
      "id" : "id-100",
      "request" :
      {
        "method" : "POST",
        "uri" : "/repositories/REPO/folders/0c00208080000107/objects",
        "headers" :
        [
          {
            "name" : "Content-Type",
            "value" : "application/hal+json"
          },
          {
            "name" : "Accept",
            "value" : "application/hal+json"
          }
        ],
        "entity" : "{\\"properties\\":{\\\"object_name\\\":\\\"my test object\\\"}}",
        "attachments" : [{ "Include" : { "href" : "cid:id-100-content" } }]
      }
    },
    {
      "id" : "id-101",
      "request" :
      {
        "method" : "POST",
        "uri" : "/repositories/REPO/folders/0c00208080000107/objects",
        "headers" :
        [
          {
            "name" : "Content-Type",
            "value" : "application/hal+json"
          },
          {
            "name" : "Accept",
            "value" : "application/hal+json"
          }
        ],
        "entity" : "{\\"properties\\":{\\\"object_name\\\":\\\"my test object\\\"}}",
        "attachments" : []
      }
    }
  ]
}
```

```
        "attachments" : [ {
            "Include" : {
                "href" : "cid:id-101-content"
            }
        } ]
    }

--frontier
Content-Type: text/plain
Content-ID: id-100-content
Content-disposition: form-data; name=id-100-content
i'm the content of id-100

--frontier
Content-Type: text/plain
Content-ID: id-101-content
Content-disposition: form-data; name=id-101-content
i'm the content of id-101

--frontier--
```

7.8.5 URI extension support

Using hal as a URL extension is supported.

```
GET http://localhost/dctm-rest/repositories/myrepo/objects/0c00000c80000105/
objects.hal
```

7.8.6 HAL in custom resources

By default, most custom resources automatically support the HAL media type for requests and responses.

7.8.6.1 A resource with the consumes media type explicitly defined

When a controller has the `RequestMapping` annotation and the `consumes` element is explicitly defined for specified media types, to support the use of the `application/hal+json` media type, you must manually add it to the `consumes` element.

```
@RequestMapping{
    consumes = {
        "application/vnd.emc.documentum+json",
        "application/vnd.emc.documentum+xml",
        "application/hal+json"
    }
    ...
}
```

7.8.6.2 Customized feed

When a resource returns a standard AtomFeed, then the application/hal+json media type is automatically supported. However, when the feed is customized, the feed model should handle the customized part. This ensures that the framework will marshal and unmarshal it correctly. Only the standard AtomFeed can be automatically converted into HAL.

The following code sample shows you how to extend the standard AtomFeed to make a customized feed for HAL. The following SearchAtomFeed class has the facets property as one more property than the standard AtomFeed class.

Here is a code sample that shows you the SearchAtomFeed class with the declaration of the facets property:

► Example 7-81: Customized SearchAtomFeed class

```
public class SearchAtomFeed extends AtomFeed<AtomEntry>
    implements HalConvertible {
    @SerializableField(value = "facets")
    @SerializableField4XmlList(unwrap = true)
    List<FacetEntry> facetResults;
    ...
}
```



► Example 7-82: Existing application/vnd.emc.documentum+json response

The format of the existing application/vnd.emc.documentum+json media type response should look like the following:

```
{
    "id": "http://localhost:8080/dctm-rest/repositories/REPO/search",
    "links": [...],
    "entries": [
        {
            "id": "09000001800020ec",
            "links": [...],
            "content": {
                "type": "application/vnd.emc.documentum+json",
                "src": "http://localhost:8080/dctm-rest/repositories/REPO/objects/09000001800020ec"
            },
            "score": "1.0",
            "terms": [
                "test"
            ]
        },
        ...
    ],
    "facets": [
        {
            "facet-id": "facet_r_modify_date",
            "facet-label": "Modified",
            "facet-value": [...]
        }
    ]
}
```



 **Example 7-83: Customized application/hal+json response**

The expected format of the customized application/hal+json media type response should look like the following:

```
{  
    "id": "http://localhost:8080/dctm-rest/repositories/REPO/search",  
    "_embedded": {  
        "collection": [{  
            "id": "09000001800020ec",  
            "score": "1.0",  
            "terms": [  
                "test"  
            ],  
            "_links": {  
                "self": {  
                    "href": "http://localhost:8080/dctm-rest/repositories/REPO/  
                            objects/09000001800020ec"  
                }  
            }  
        }  
    },  
    "facets": [{  
        "facet-id": "facet_r_modify_date",  
        "facet-label": "Modified",  
        "facet-value": [...]  
    }]  
},  
    "_links": {}  
}
```



In addition to the standard AtomFeed information, the facets element must be added and returned. To do this, SearchAtomFeed must implement the HalConvertible interface.

Here's a code sample that shows you the HalConvertible interface:

 **Example 7-84: The HalConvertible interface**

```
public interface HalConvertible {  
    public void convert(HalObject object);  
}
```



This interface has only one method (the convert method), which is used to convert customized information into a HalObject object. The HalObject already has pre-converted information, such as entries and feed information.



Tip: You can also convert your customized information into a totally customized HAL format.

 **Example 7-85: The HalObject class**

The HalObject has two fields:

```
public class HalObject extends HalLinkable {
    @SerializableField
    private Object object;
    @SerializableField("_embedded")
    private HalCollection embedded;
}
```

The Object member of the HalObject class is used for a single resource. However, it may also be used for feed level information of a collection resource.

The HalCollection member of the HalObject class is only used for the collection resource and it refers to the `collection` element of the `_embedded` element.



▶ Example 7-86: The HalSearchCollection class

Here the `facets` parameter should be represented at the same level as the `collection`, so a customized HalCollection model, in which the `facets` object is defined, is required.

```
@SerializableType(ignoreNullFields = true, fieldOrder = {"collection", "facets"})
public static class HalSearchCollection extends HalCollection {
    @SerializableField4XmlList(unwrap=true, asPropBag=true)
    private List<FacetEntry> facets;
    ...
}
```



Lastly, you must implement the `convert(HalObject)` method of the `HalConvertible` interface within the `SearchAtomFeed` class.

Here is a code sample that shows you the `SearchAtomFeed` class definition:

```
public class SearchAtomFeed extends AtomFeed<AtomEntry>
    implements HalConvertible {
    ...
    @Override
    public void convert(HalObject object) {
        // Create a new HalSearchCollection, and set the original embedded
        // collection of the HalObject to the new HalSearchCollection.
        // Then set the facets within the HalSearchCollection object and
        // don't forget to set the original variable facetResults to null.
        // Otherwise, it will be in the Response representation.
        if(facetResults != null && !facetResults.isEmpty()) {
            object.setEmbedded(new HalSearchCollection(object.getEmbedded()
                .setFacets(facetResults)));
            facetResults = null;
        }
    }
}
```

7.8.7 Configure HAL as a standalone media type

The media type registry can be configured to use HAL as a stand alone media type. This means that you can turn off the use of both XML and JSON.

For more information on turning off media types, see “[Turning off XML, JSON, or HAL media types](#)” on page 265.

Here is a code snippet that shows you the HAL only registry definition:

```
media-type-registry:  
  - media-type: hal
```

With this configuration, the application/vnd.emc.documentum+json and the application/vnd.emc.documentum+xml are not supported. Only the application/hal+json media type is supported.

7.9 Registering URI templates

A RESTful style to make your custom resources discoverable is to add HATEOAS links (link relations) into the existing Core resources, including the Home Document of custom resources. Foundation REST API allows you to register custom resource URI templates and then add them as link relations into resources. This section introduces the registration of URI templates.

In the REST WAR file, the rest-api-custom-resource-registry.yaml YAML file provides a configuration entry called *uri-template-registry* that can be used to register new URI templates.

Here's a sample showing the syntax for a URI template definition:

Example 7-87: A URI template definition

```
uri-template-registry:  
  - name: <template name>  
    [href|hreftemplate]: <URI template pattern for 'href' or 'hreftemplate'>  
    encoding: <encoding method>  
    external: <true or false>
```

Example:

```
uri-template-registry:  
  - name: X_ALIAS_SET_RESOURCE_TEMPLATE  
    href: '{repositoryUri}/alias-sets/{aliasSetId}{ext}?owner={userName}'  
    encoding: dual-url-encoding  
  - name: X_SEARCH_RESOURCE_TEMPLATE  
    hreftemplate: '{baseUri}/x-search{?q,facet}'  
  - name: X_MODULE_RESOURCE_TEMPLATE  
    href: '{baseUri}/global-modules/{moduleId}{ext}'
```



Note: YAML is very strict with indentation and spaces. Incorrect indentation and redundant spaces may cause errors. Tab characters (\t) are never allowed for indentation in YAML.

Alternatively, when you want to modify `rest-api-custom-resource-registry.yaml` before generating the WAR file, the file can be put into the web module path: `<custom-resource-web>/src/main/resources/com/emc/documentum/rest/script` and use the Maven overlay plugin to overwrite the default YAML file.

This configuration enables you to register new URI templates with the shared URI factory. The system uses these URI templates to resolve new links that are added to existing resources. A URI template consists of the following elements as key-value pairs:

Table 7-6: URI Template Elements

Key	Value
name *	Name of a custom URI template. The name must start with the prefix X_. We recommend that you use uppercase words delimited by underscore (_) to name a URI template. Example: X_ALIAS_SET_RESOURCE_TEMPLATE

Key	Value
[href hreftemplate] *	<p>URI template pattern for a link. The key is either <code><href></code> or <code><hreftemplate></code>. Only one <code><href></code> or <code><hreftemplate></code> exists in a URI template.</p> <p>The URI template pattern can contain variables or query parameters enclosed in curly brackets <code>{}</code>. For example:</p> <pre>{repositoryUri}/<item1>/<item2>/<...>?<param1>&<param2>&<...>{ext}</pre> <p>For an href link, all variables in the URI pattern must be resolved on the server side, meaning that you must modify value-mapping correspondingly in the ADD LINKS ON EXISTING RESOURCES section for non-predefined variables.</p> <p>By contrast, an <code><hreftemplate></code> link must contain variables that are treated as placeholders, such as <code>{?foo, bar}</code>, which does not need resolving on the server side.</p> <p><i>Examples of hreftemplates</i></p> <ul style="list-style-type: none"> • <code><hreftemplate></code> with no fixed query parameters: <code>/repositories/{repositoryName}/search?{?q,locations}</code> • <code><hreftemplate></code> with both fixed query parameters and placeholders: <code>/repositories/{repositoryName}/search?page=10&items-per-page={numberOfItems}&{q,locations}</code> <p>Predefined variables are resolved without additional settings in value-mapping. They are:</p> <ul style="list-style-type: none"> • <code>{baseUri}</code>: URI root of the current deployment. Example: <code>{baseUri}</code> may refer to <code>https://current-deploy-host:8443/dctm-rest</code> • <code>{repositoryUri}</code>: URI root of the current repository. Example: <code>{repositoryUri}</code> may refer to <code>https://current-deploy-host:8443/dctm-rest/repositories/acme</code> • <code>{ext}</code>: URI extension for the representation format, such as <code>.XML</code> or <code>.JSON</code>
encoding	Encoding method for path variables. Valid values are: <ul style="list-style-type: none"> • default • safe-text-encoding, which must be used when the property value to resolve a path variable contains special characters that may impact a URI, such as the slash character (/)

Key	Value
external	<p>Specifies whether this URI template is external or internal. Valid values are:</p> <ul style="list-style-type: none"> • true: Indicates this is an external URI template. External URI templates are not mapped to any custom resources. The external URI templates MUST NOT use predefined variables {<repositoryUri>} or {<ext>}. The accessibility of an external URI template will not be validated by the Foundation REST API server. • false: Indicates this is an internal URI template. The internal URI template is mapped to a REST resource. The internal URI template MUST start with predefined variables {<baseUri>}, {<repositoryUri>} <p>The default value is false.</p>
* Required	

The URI template name must be unique so that other resources used to build link relations can reference the template. In the code, the URI template can also be used to create an actual URL using the `buildUriByTemplateName` method of `com.emc.documentum.rest.http.UriFactory`. For example:

```
String moduleUri = uriFactory.buildUriByTemplateName(
    "X_MODULE_RESOURCE_TEMPLATE",
    Collections.singletonMap("moduleId", getDataInternal().getId()));
```

7.9.1 Normalization of URI Templates

Starting from Foundation REST API 7.3, the system checks the format of an internal URI template and automatically appends the predefined variables `{baseUri}`, `{repositoryUri}`, and `{ext}` when necessary.

With this feature, the following URI templates are the same:

- /help vs. ./help{ext} vs. {baseUri}/help vs. {baseUri}/help{ext}
- /repositories/{repositoryName}/users?name={userId} vs. {repositoryUri}/users{ext}?name={userId} vs. {baseUri}/repositories/{repositoryName}/users?name={userId}

7.9.2 Normalization of Custom URI Templates

While it is not necessary that you reference resources in YAML by defining custom URI templates, the custom URI template configuration in YAML is still useful when you want to:

- Build external link relations
- Build internal link relations with customizations. For example, when adding fixed query parameters, template parameters, and so on

From Foundation REST API 7.3 and later, a complete definition of which customizations can be applied to a custom URI template have been added. Here's a code sample that shows you the URI template registry

```

uri-template-registry:
  - name: X_HELP_TEMPLATE
    href: '{baseUri}/help{ext}'
  - name: X_POLICIES_FEED_TEMPLATE
    href: '{repositoryUri}/objects/{objectId}/policies{ext}'
  - name: X_POLICY_TEMPLATE
    href: '{repositoryUri}/objects/{objectId}/policies/{policyId}{ext}'
  - name: X_AUTHOR1_TEMPLATE
    href: '{repositoryUri}/users/{userId}{ext}'
    encoding: safe-text-encoding
  - name: X_AUTHOR2_TEMPLATE
    href: '{repositoryUri}/users{ext}?name={userId}'
  - name: X_CUSTOM_SEARCH_TEMPLATE
    hreftemplate: '{baseUri}/x-search{?,locations,page,items-per-page}'
  - name: X_CUSTOM_LOCATION_SEARCH_TEMPLATE
    hreftemplate: '{repositoryUri}/x-search?locations={path}{&q,page,items-per-page}'
  - name: X_EXTERNAL_SEARCH_TEMPLATE
    href: 'http://www.google.com?q={keyword}'
    external: true
  
```

7.9.3 Normalization of Internal URI Templates

Starting from Foundation REST API 7.2 two predefined variables `{baseUri}`, `{repositoryUri}`, and the suffix variable `{ext}` are applicable to internal URIs. When a URI template is internal (default to internal), the runtime automatically checks the URI format and appends the prefix or suffix when necessary.

With this feature, the following URI templates are the same:

- `{baseUri}/help{ext}` vs. `/help` vs. `{baseUri}/help` vs. `/help{ext}`
- `{repositoryUri}/users{ext}?name={userId}` vs. `/repositories/{repositoryName}/users?name={userId}` vs. `{baseUri}/repositories/{repositoryName}/users?name={userId}`

7.9.4 Formats for the href Template

When a URI template is defined as an href template, there are href template parameters appended to the link. Foundation REST API supports a sub set of RFC6570.

- Appending template parameters `{a,b,c,...}` onto a URI where there are no fixed query parameters
For example: `/repositories/{repositoryName}/search?{?q, locations, ...}`
- Appending template parameters `{&a, b, c, ...}` onto a URI where there are fixed query parameters
For example: `/repositories/{repositoryName}/search?page=10&items-per-page={numberOfItems}{&q, locations, ...}`

7.9.5 Appending User Defined Variables onto the URI Template

User defined variables can be applied to internal and external URI templates, both href or href template URI templates. The variables must be resolved in custom Java code or by using variable value mapping in YAML.

Here's a code sample that shows you how to do that:

Example 7-88: Custom URI Templates With User Defined Variables

```

uri-template-registry:
  - name: X_AUTHOR_TEMPLATE
    href: '{repositoryUri}/users/{userId}{ext}'
    encoding: safe-text-encoding
  - name: X_MARKETING_GROUPS_TEMPLATE
    href: '{repositoryUri}/groups{ext}?owner={userId}&subject={subject}&inline=true&items-per-page=50'
  - name: X_CUSTOM_LOCATION_SEARCH_TEMPLATE
    hreftemplate: '{repositoryUri}/x-search?locations={location}&q,page,items-per-page'
  - name: X_EXTERNAL_SEARCH_TEMPLATE
    href: 'http://www.google.com?q={keyword}'
    external: true

---
resource-link-registry:
  - resource: document
    link-relation: 'http://www.custom.com/sample/linkrel/author'
    uri-template: 'X_AUTHOR_TEMPLATE'
    value-mapping: [userId:owner_name]
  - resource: document
    link-relation: 'http://www.custom.com/sample/linkrel/marketing-groups'
    uri-template: 'X_AUTHOR_TEMPLATE'
    value-mapping: [userId:owner_name,subject:subject]
  - resource: document
    link-relation: 'http://www.custom.com/sample/linkrel/folder-search'
    uri-template: 'X_CUSTOM_LOCATION_SEARCH_TEMPLATE'
    value-mapping: [location:r_folder_path]
  - resource: document
    link-relation: 'http://www.custom.com/sample/linkrel/gsearch'
    uri-template: 'X_EXTERNAL_SEARCH_TEMPLATE'
    value-mapping: [keyword:keywords]

```



7.10 Normalization of custom URI templates

Though it is no longer necessary to reference resources in YAML by defining custom URI templates, the custom URI template configuration in YAML is still useful when you want to:

- Build external links relations
- Build internal link relations with customizations. For example adding fixed query parameters, template parameters, and more

Here are some samples of the URI template registry

```

uri-template-registry:
  - name: X_HELP_TEMPLATE

```

```
    href:      '{baseUri}/help{ext}'
- name:     X_POLICIES_FEED_TEMPLATE
  href:      '{repositoryUri}/objects/{objectId}/policies{ext}'
- name:     X_POLICY_TEMPLATE
  href:      '{repositoryUri}/objects/{objectId}/policies/{policyId}{ext}'
- name:     X_AUTHOR1_TEMPLATE
  href:      '{repositoryUri}/users/{userId}{ext}'
  encoding: safe-text-encoding
- name:     X_AUTHOR2_TEMPLATE
  href:      '{repositoryUri}/users{ext}?name={userId}'
- name:     X_CUSTOM_SEARCH_TEMPLATE
  hreftemplate: '{baseUri}/x-search{?q,locations,page,items-per-page}'
- name:     X_CUSTOM_LOCATION_SEARCH_TEMPLATE
  hreftemplate: '{repositoryUri}/x-search?locations={path}{&q,page,items-per-page}'
- name:     X_EXTERNAL_SEARCH_TEMPLATE
  href:      'http://www.google.com?q={keyword}'
  external: true
```

7.10.1 Normalization of Internal URI Templates

There are two URI prefix variables; `{baseUri}` and `{repositoryUri}` the other suffix variable `{ext}` that are applicable to internal resource URIs. When a URI template is internal (default to internal), the runtime automatically checks the URI format and appends the prefix or suffix where necessary.

For example, the following internal URI template definitions are equivalent:

- `{baseUri}/help{ext}` vs. `/help` vs. `{baseUri}/help` vs. `/help{ext}`
- `{repositoryUri}/users{ext}?name={userId}` vs. `/repositories/{repositoryName}/users?name={userId}` vs. `{baseUri}/repositories/{repositoryName}/users?name={userId}`

7.10.2 Formats for href Template

When a URI template is defined as `<hreftemplate>`, there are href template parameters appended to the link. Foundation REST API supports a sub set of the RFC6570 standard.

- Appending template parameters `{?a,b,c,...}` onto a URI where there are no fixed query parameters.

For example: `/repositories/{repositoryName}/search{?q,locations,...}`

- Appending template parameters `{&a,b,c,...}` onto a URI where there are fixed query parameters

For example: `/repositories/{repositoryName}/search?page=10&items-per-page={numberOfItems}{&q,locations,...}`

7.10.3 User Defined Variables of the URI Template

User defined variables can interfere with internal and external URI templates, including `href` or `hreftemplate` URI templates. The variables must be resolved either in custom Java code, or by using the variable value mapping in YAML.

Here's a code sample that shows you how to work with URI templates:

Example 7-89: Custom URI Templates with User Defined Variables

```

uri-template-registry:
  - name: X_AUTHOR_TEMPLATE
    href: '{repositoryUri}/users/{userId}{ext}'
    encoding: safe-text-encoding
  - name: X_MARKETING_GROUPS_TEMPLATE
    href: '{repositoryUri}/groups{ext}?owner={userId}&subject={subject}&
          inline=true&items-per-page=50'
  - name: X_CUSTOM_LOCATION_SEARCH_TEMPLATE
    hreftemplate: '{repositoryUri}/x-search?locations={location}&q,page,items-per-page'
  - name: X_EXTERNAL_SEARCH_TEMPLATE
    href: 'http://www.google.com?q={keyword}'
    external: true

---
resource-link-registry:
  - resource: document
    link-relation: 'http://www.custom.com/sample/linkrel/author'
    uri-template: 'X_AUTHOR_TEMPLATE'
    value-mapping: [userId:owner_name]
  - resource: document
    link-relation: 'http://www.custom.com/sample/linkrel/marketing-groups'
    uri-template: 'X_AUTHOR_TEMPLATE'
    value-mapping: [userId:owner_name,subject:subject]
  - resource: document
    link-relation: 'http://www.custom.com/sample/linkrel/folder-search'
    uri-template: 'X_CUSTOM_LOCATION_SEARCH_TEMPLATE'
    value-mapping: [location:r_folder_path]
  - resource: document
    link-relation: 'http://www.custom.com/sample/linkrel/gsearch'
    uri-template: 'X_EXTERNAL_SEARCH_TEMPLATE'
    value-mapping: [keyword:keywords]

```



7.11 Registering root resources

A top-level custom resource may not have a relevant resource providing a link relation to discover it. For such resources, you must add it into the Home document with a designed link relation. That way, custom REST services comply with the hypermedia-driven design instead of providing hard-coded URIs to Foundation REST API clients. The only resource URI the Foundation REST API clients need to bookmark is the Home document resource whose URI path is `/services`. All root resources should be registered to the Home document with link relations.

In the REST WAR file, the YAML file `rest-api-custom-resource-registry.yaml` provides a configuration entry `root-service-registry` to register top-level resources to the Home document. You can modify this YAML file to add top-level resources.

The syntax for a root service registry is shown as follows:

```
root-service-registry:  
- name: <descriptive information of the resource>  
target-resource:<resource name register for this service>  
link-relation:<the link relation name of the resource>  
uri-template: <the URI template name registered for this service>  
allowed-methods:<allowed HTTP methods on this resource>  
media-types:<supported media types for this resource>
```

Template Example using uri-template:

```
root-service-registry:  
- name: User defined global search resource  
link-relation:'x-search'  
uri-template:X_SEARCH_RESOURCE_TEMPLATE  
allowed-methods:[POST]  
media-types:[application/vnd.emc.documentum+json]  
- name: User defined custom alias set feed resource  
link-relation:'x-alias'  
uri-template: X_ALIAS_SETS_RESOURCE_TEMPLATE  
allowed-methods:[GET]  
media-types:[application/vnd.emc.documentum+json, application/vnd.emc.documentum+xml]
```

Template Example using target-resource:

```
root-service-registry:  
- name: User defined global search resource  
link-relation: 'http://identifiers.emc.com/linkrel/x-search'  
target-resource: 'acme#global-search'  
allowed-methods: [POST]  
media-types: [application/vnd.emc.documentum+json, application/vnd.emc.documentum+xml]
```

This configuration enables you to register new resources to the Home Document. You can specify the following elements (key-value pairs):

Table 7-7: Root Service Registry

Key	Value
name	Descriptive information of a resource registered to the Home Document.

Key	Value
uri-template	<p>URI template registered for this resource. The URI template must exist in the REGISTER NEW URI TEMPLATES on page 245 section.</p> <p>For resources registered to the Home Document, only the following variables are allowed in a specified URI template:</p> <ul style="list-style-type: none"> • predefined variables {baseUri} and {ext} • variables that are treated as placeholders, such as query strings (<i>hreftemplate</i> only) <p>The predefined variable {repositoryUri} and any variables resolved via value-mapping settings are not allowed in the URI template.</p> <p> Note: One of the two attributes, <i>uri-template</i> or <i>target-resource</i> is required but not both. When both <i>uri-template</i> and <i>target-resource</i> are set, <i>target-resource</i> takes priority. Both of the <i>uri-template</i> and <i>target-resource</i> attributes cannot be left empty, at least one of them must contain a value.</p>
target-resource	<p>Allows users to reference an existing resource by name.</p> <p>For more information, see the preceding Note.</p>
link-relation *	<p>Link relation name of a registered resource, which enables you to locate the resource in the Home Document.</p> <p>Only one link relation can be used to refer to a registered resource.</p>
allowed-methods *	<p>A comma-separated list of the HTTP methods that can be performed on this resource.</p> <p>Example: [POST, GET]</p>
media-types *	<p>A comma-separated list of the supported media types for this resource.</p> <p>Example: [application/vnd.emc.documentum+json, application/vnd.emc.documentum+xml]</p>
* Required	

Once a custom top-level resource is registered, the Core Home document resource will contain the additional entry for the top-level resource. Here is an example.

1. In the `uri-template-registry` section, add the URI template of the link relation by specifying these elements: `name`, `href/hreftemplate`, `encoding` (optional) and `external` (optional).

Example:

```
uri-template-registry:
- name:X_SEARCH_RESOURCE_TEMPLATE
  hreftemplate:'{baseUri}/x-search{?q,facet}'
  encoding:dual-url-encoding
  external:false
```

For link relations added to the Home Document, only predefined variables and variables used as placeholders can appear in the URI templates. For example, the following URI template is not eligible for link relations added to the Home Document:

Example:

```
uri-template-registry:  
- name: X_ALIAS_SET_RESOURCE_TEMPLATE  
  hreftemplate: '{repositoryUri}/alias-sets/{aliasSetId}{ext}?owner={userName}'  
  encoding:dual-url-encoding  
  external:false
```

2. In the root-service-registry section, specify the following elements for the link relation: name, link-relation, uri-template, allowed-method, and media-types.

Example:

```
root-service-registry:  
- name: User defined global search resource  
  link-relation: 'x-search'  
  uri-template: X_SEARCH_RESOURCE_TEMPLATE  
  allowed-methods:[POST]  
  media-types: [application/vnd.emc.documentum+json, application/vnd.emc.documentum  
+xml]
```

3. Deploy the custom resources WAR file after the YAML is updated. The following sample of Home Document has the custom link relation x-search added.

```
<?xml version="1.0" encoding="UTF-8"?>  
<resources xmlns="http://identifiers.emc.com/vocab/documentum"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  
  <!-- core top resource -->  
  <resource rel="http://identifiers.emc.com/linkrel/repositories">  
    <link href="http://localhost:8080/acme-rest/repositories"/>  
    <hints>  
      <allow><i>GET</i></allow>  
      <representations>  
        <i>application/xml</i>  
        <i>application/json</i>  
        <i>application/atom+xml</i>  
        <i>application/vnd.emc.documentum+json</i>  
      </representations>  
    </hints>  
  </resource>  
  <!-- core top resource -->  
  <resource rel="about">  
    <link href="http://localhost:8080/acme-rest/product-information"/>  
    <hints>  
      <allow><i>GET</i></allow>  
      <representations><i>application/xml</i>  
        <i>application/json</i>  
        <i>application/vnd.emc.documentum+xml</i>  
        <i>application/vnd.emc.documentum+json</i>  
      </representations>  
    </hints>  
  </resource>  
  <!-- custom top resource -->  
  <resource rel="x-search">  
    <link hreftemplate="http://localhost:8080/acme-rest/x-search{?q,facet}" />  
    <hints>  
      <allow><i>POST</i></allow>  
      <representations>  
        <i>application/vnd.emc.documentum+json</i>  
        <i>application/vnd.emc.documentum+json</i>  
      </representations>
```

```
</representations>
</hints>
</resource>
</resources>
```

7.12 Adding links to core resources

To make a custom resource discoverable via Core resources, you need to add new link relations pointing to the custom resources to one or more Core resources.

In the REST WAR file, the YAML file `rest-api-custom-resource-registry.yaml` provides a configuration entry `resource-link-registry` to add link relations to Core resources. You can modify this YAML file to add root resources.

The syntax for a link relation registry is shown as follows:

```
resource-link-registry:
- resource: <the resource code name>
  link-relation:<the new link relation name>
  target-resource:<the resource code name to link to>
  uri-template: <the name of the URI template defined in uri-template-registry>
  value-mapping:<the property names from the resource to resolve the variable values in the URI template>
```

Link Example Using uri-template:

```
resource-link-registry:
- resource: object
  link-relation: 'http://identifiers.emc.com/linkrel/acl'
  uri-template: 'X_ACL_RESOURCE_TEMPLATE'
  value-mapping:[objectId:r_object_id]
- resource: folder
  link-relation: 'x-folder-attachment'
  uri-template: 'X_ATTACHMENT_RESOURCE_TEMPLATE'
  value-mapping:[]
```

Link Example Using target-resource:

```
# build ACL resource link relation on core document resource using 'target-resource'
resource-link-registry:
- resource: document
  link-relation: 'http://www.custom.com/sample/linkrel/acl'
  target-resource: 'acme#acl'
  value-mapping: [aclName:acl_name]
```

This configuration allows you to register new links to Core resources. You can specify the following elements (key-value pairs):

Table 7-8: Resource Link Registry

Key	Value
resource *	The code name of the resource to which the links are added. Each Core resource has a unique code name. In the link registry, following resources are supported:[format, user, group, content, relation, type, object, document, folder, cabinet, network-location, relation-type, repository]. See Appendix C for all code names of Core resources.

Key	Value
link-relation *	<p>Link relation name of the registered resource, which enables you to locate the resource in the existing resource.</p> <p>One and only one link relation can be used to refer to a registered resource.</p>
uri-template *	The name of the URI template registered in <code>uri-template-registry</code> . The URI template can contain variables which need to be resolved during the generation of the link relation <code>href</code> .
value-mapping	<p>Value mappings are used to resolve values of path variables or query parameters on URI templates defined in <code>uri-template</code>.</p> <p>A value mapping follows the pattern:</p> <p><code>[<variableName1>:<propertyName1>[,<variableName2>:<propertyName2>]*</code></p> <p><code><variableName></code> represents a variable specified in a URI template. There cannot be any duplicated variables in a URI template.</p> <p><code><propertyName></code> represents a property in the resource. Note that properties of the Repository resource (such as <code>Id</code>, <code>name</code>, and <code>description</code>) cannot be part of a path variable.</p> <p>When the URI is generated at runtime, the variables will be replaced by the property values; the property can be repeating. When any of the properties in the variables is repeating, multiple links will be generated, each with a different value in the variable segment. Besides, a 'title' attribute will be set on the link representation to differentiate <code>href</code> for different repeating values. There could be at most one repeating property in the variables. If the property values for the variables are not returned in the resource object (e.g. by custom view), the links will neither be generated nor be presented on the resource.</p>
target-resource	Allows users to reference an existing resource by name.
* Required	

Once the custom root resource is registered, the Core home document resource will contain the additional entry for the root resource. Here is an example.

- In the `uri-template-registry` section, add the URI template of the link relation by specifying these elements: `name`, `href`/`hrefTemplate`, `encoding` (optional) and `external` (optional).

Example:

```
uri-template-registry:
- name: X_ACL_RESOURCE_TEMPLATE
  href: '{repositoryUri}/objects/{objectId}/acl{ext}'
```

- In the `resource-link-registry` section, specify the following elements for the link relation: `resource`, `link-relation`, `uri-template`, and `value-mapping`.

Example:

```
resource-link-registry:
- source: object
link-relation:'http://identifiers.emc.com/linkrel/acl'
uri-template: X_ACL_RESOURCE_TEMPLATE
value-mapping:[objectId:r_object_id]
```

3. Deploy the custom resources WAR file after the YAML is updated. The following sample of Home Document has the custom link relation `http://identifiers.emc.com/linkrel/acl` is added.

Example:

```
<object xmlns="http://identifiers.emc.com/vocab/documentum"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="dm_cabinet"
definition="http://localhost:8080/acme-rest/repositories/
REPO/types/dm_cabinet">
<propertiesxsi:type="dm_cabinet-properties">
<object_name>a;x.f</object_name>
...
<r_object_id>0c0020808000dfb6</r_object_id>
</properties>
<links>
<!-- core link relations-->
<link rel="self"
href="http://localhost:8080/acme-rest/repositories/REPO/objects/
0c0020808000dfb6" />
..
<link rel="http://identifiers.emc.com/linkrel/relations"
href="http://localhost:8080/acme-rest/repositories/REPO/relations?
related-object-id=0c0020808000dfb6&related-object-role=any" />
<!--customer defined link relations-->
<link rel="http://identifiers.emc.com/linkrel/acl"
href="http://localhost:8080/acme-rest/repositories/REPO/objects/
0c0020808000dfb6/acl" />
</links>
</object>
```

7.13 Disabling specific resources

Disabling specific Core resources may become necessary for REST extensibility development work. For example, you may want to avoid exposing specific resources to end users, or you may want to develop a custom resource to override an existing core resource. In such scenarios, you can disable a specific core resource and package your own custom resource into the WAR.

7.13.1 Disable a Specific Resource with YAML

You can use the same YAML configuration file under `WEB-INF\classes` to define which resources will be disabled. Use the `<disabling-resource-registry>` attribute to define which core resources will be disabled.

There are some elements that you may want to reference when developing custom resources. For more information, see [Appendix B, Resource coding index on page 395](#).

7.13.2 Impact of Disabling Core Resources

When you disable some particular core resources, it may cause link discovery of resources to be broken. In this case, a hypermedia driven client, which follows link relations to discover resources, cannot behave as expected.

It is imperative that you understand the impact of disabling resources before proceeding to disable any core resources. There are several different cases that impact representations of other resources when certain other resources are disabled.

For example:

- *Case 1:* Inactive link relations are removed on existing resources because certain other resources are disabled.

The *<user>* resource and the *<user default folder>* resource. When the *<user default folder>* resource is disabled, the link relation `http://identifiers.emc.com/linkrel/default-folder` on *<user>* resource is no longer presented. Here's an image that illustrates this case.



Example 7-90: Default User Resource XML Representation

The following code sample shows the default XML for the user resource, before the user default folder resource is disabled. Notice the link relation that is shown in bold.

```

<user xsi:type="dm_user"
      definition="http://localhost/dctm-rest/repositories/REPO/types/dm_user"
      xmlns="http://identifiers.emc.com/vocab/documentum"
      xmlns:dm="http://identifiers.emc.com/vocab/documentum"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <properties>
    <user_name>dmadmin</user_name>
    <r_object_id>1100208080000501</r_object_id>
    ...
  </properties>
  <links>
    <link rel="self"
          href="http://localhost/repositories/REPO/users/dmadmin" />
    <link rel="parent"
          href="http://localhost/repositories/REPO/groups?username=dmadmin" />
    <link rel="http://identifiers.emc.com/linkrel/default-folder"
          href="localhost/repositories/REPO/users/dmadmin/home" />
  </links>
</user>

```

▶ **Example 7-91: User Resource XML Representation (user default resource disabled)**

After the default folder resource is disabled, the link relation shown in bold in the preceding sample is no longer present in the XML representation.

```
<user xsi:type="dm_user"
      definition="http://localhost/dctm-rest/repositories/REPO/types/dm_user"
      xmlns="http://identifiers.emc.com/vocab/documentum"
      xmlns:dm="http://identifiers.emc.com/vocab/documentum"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <properties>
    <user_name>dmadmin</user_name>
    <r_object_id>1100208080000501</r_object_id>
    ...
  </properties>
  <links>
    <link rel="self"
          href="http://localhost/repositories/REPO/users/dmadmin"/>
    <link rel="parent"
          href="http://localhost/repositories/REPO/groups?username=dmadmin"/>
  </links>
</user>
```



- Case 2: The atom entry edit link is not present when the single object resource is disabled.

For example, the `<users>` resource and the `<user>` resource. When the `<user>` resource is disabled, the link relation `edit` on the `<users>` resource's entries is no longer present.



▶ **Example 7-92: Default Users Resource XML Representation**

The following code sample shows the default XML for the `<users>` resource, before the `<user>` resource is disabled. Notice the link relation that is shown in bold:

```
<feed xmlns="http://www.w3.org/2005/Atom"
      xmlns:dm="http://identifiers.emc.com/vocab/documentum"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <id>http://localhost/repositories/REPO/users</id>
  <title>Users</title>
  ...
  <link rel="self" href="http://localhost/repositories/REPO/users"/>
  <entry>
    <id>http://localhost/repositories/REPO/users/Administrator</id>
    <title>dmadmin</title>
    ...
    <content type="application/xml"
             src="http://localhost/repositories/REPO/users/dmadmin"/>
    <link rel="edit" href="http://localhost/repositories/REPO/users/dmadmin"/>
  </entry>
</feed>
```



Example 7-93: Users Resource XML Representation (user resource disabled)

After the `<user>` resource is disabled, the `edit` link relation shown in bold, is no longer present in the XML representation.

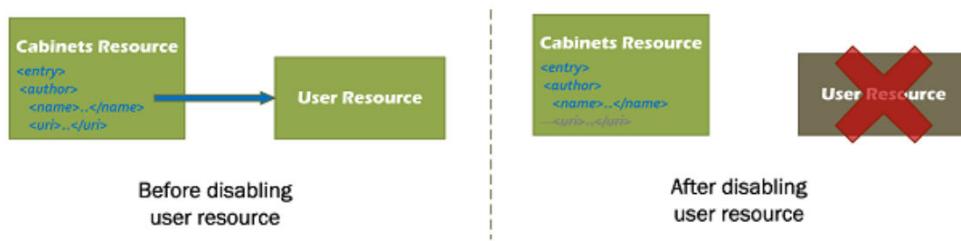
```
<feed xmlns="http://www.w3.org/2005/Atom"
      xmlns:dm="http://identifiers.emc.com/vocab/documentum"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <id>http://localhost/repositories/REPO/users</id>
  <title>Users</title>
  ...
  <link rel="self" href="http://localhost/repositories/REPO/users"/>
  <entry>
    <id>http://localhost/repositories/REPO/users/Administrator</id>
    <title>dmadmin</title>
    ...
  </entry>
```



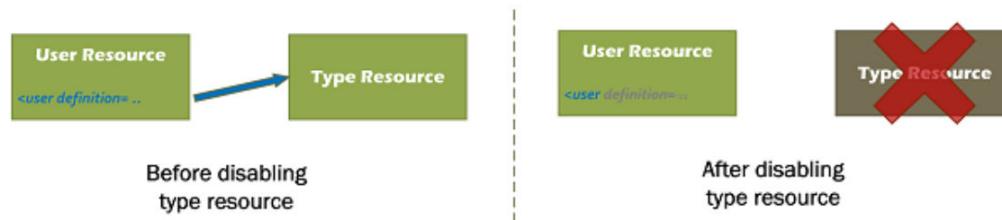
Note: Disabling resources has an impact on DQL query resources. When certain resources are disabled, the DQL query result does not provide the `<edit>` link for the query result items in their entries as expected.

The following diagrams illustrate other cases where disabling a resource has an impact on another resource.

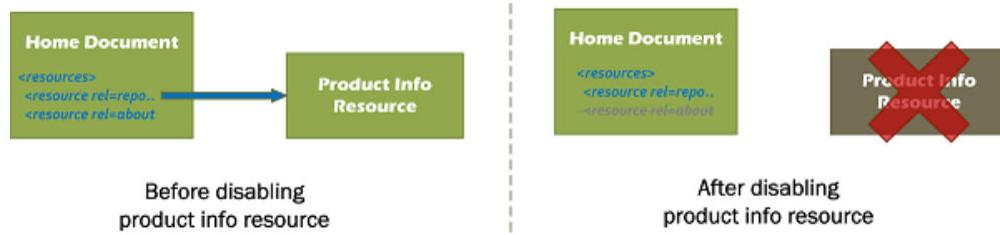
When the `<user>` resource is disabled, the atom user URL in the atom feed is removed. One example is the `<cabinets>` resource.



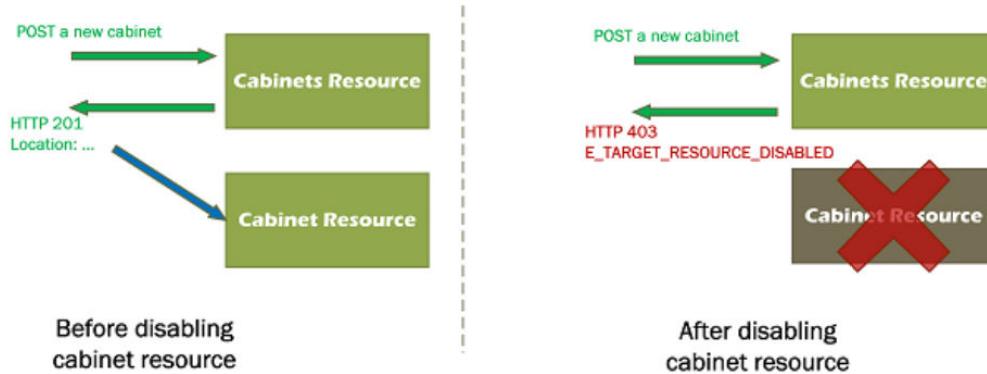
When the `<type>` resource is disabled, the type definition URL in the persistent object resource is removed. One example is the `<cabinet>` resource.



When the `<root>` is disabled, the link relation in the home document is removed. One example is the `<product information>` resource.



Some collection based resources support the HTTP POST method to create a new resource in the collections. When the single resource is disabled, the POST method on the collection resource fails. One example is the <cabinets> resource and <cabinet> resource.



7.13.3 Samples

Example 7-94: Disable a Resource

Configuration

The <formats> resource and the <format> resource are disabled.

```
disabling-resource-registry:
  - resources: [formats,format]
```

Representation

At runtime, both the <formats> and the <format> resources are not accessible.

```
GET /acme-rest/repositories/REPO/formats HTTP/1.1
GET /acme-rest/repositories/REPO/format/crtext HTTP/1.1
```

```
Http 404 Not Found
```

In the <repository> resource, the link relation to <formats> is not available.

```
<repository xmlns="http://identifiers.emc.com/vocab/documentum"
  xmlns:dm="http://identifiers.emc.com/vocab/documentum"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

```

<id>8320</id>
<name>REPO</name>
<links>
    <link rel="self" href="/acme-rest/repositories/REPO"/>
    <link rel="http://identifiers.emc.com/linkrel/alias"
        href="/acme-rest/repositories/REPO/alias-sets"/>
    <link rel="http://identifiers.emc.com/linkrel/users"
        href="/acme-rest/repositories/REPO/users"/>
    <link rel="http://identifiers.emc.com/linkrel/current-user"
        href="/acme-rest/repositories/REPO/currentuser"/>
    <link rel="http://identifiers.emc.com/linkrel/groups"
        href="/acme-rest/repositories/REPO/groups"/>
    <link rel="http://identifiers.emc.com/linkrel/cabinets"
        href="/acme-rest/repositories/REPO/cabinets"/>
    <link rel="http://identifiers.emc.com/linkrel/network-locations"
        href="/acme-rest/repositories/REPO/network-locations"/>
    <link rel="http://identifiers.emc.com/linkrel/relations"
        href="/acme-rest/repositories/REPO/relations"/>
    <link rel="http://identifiers.emc.com/linkrel/relation-types"
        href="/acme-rest/repositories/REPO/relation-types"/>
    <link rel="http://identifiers.emc.com/linkrel/checked-out-objects"
        href="/acme-rest/repositories/REPO/checked-out-objects"/>
    <link rel="http://identifiers.emc.com/linkrel/types"
        href="/acme-rest/repositories/REPO/types"/>
    <link rel="http://identifiers.emc.com/linkrel/dql"
        hreftemplate="/acme-rest/repositories/REPO{?dql,page,items-per-page}"/>
    <link rel="http://identifiers.emc.com/linkrel/search"
        hreftemplate="/acme-rest/repositories/REPO/search{?q,collections,locations,facet,
            inline,page,items-per-page,include-total,sort,view}"/>
    <link rel="http://identifiers.emc.com/linkrel/batches"
        href="/acme-rest/repositories/REPO/batches"/>
    <link rel="http://identifiers.emc.com/linkrel/batch-capabilities"
        href="/acme-rest/repositories/REPO/batch-capabilities"/>
</links>
...
</repository>
```



7.14 Overriding specific resources

When a new resource overrides an existing resource, the new resource will become active, and the existing resource will become inactive. All link relations that were pointing to the existing resource, which is the one that is overridden, will now be pointing to new resource.

This configuration to develop new resources to deprecate OpenText Documentum CM Core resources.

You may be wondering why resources that have been overridden must implement the same URI patterns as Core resources. This is because the Core REST Link Builder checks whether the target URI pattern is active before writing that link relation to the resource. When the new resource URI is not implemented in the same way as the Core resource, the target URI in the link relation of the sourcing resource is inactive, and therefore it is not pointing to the new (overridden) resource.

To get around this limitation, you can introduce a new registry entry in YAML and explicitly tell a resource to override another resource. At runtime, you can also make the link relation builder understand the overriding relationship between two resources. Then the link relation targeting the original resource can be redirected

dynamically to the new resource. This approach avoids the need for any hardcoded URI availability checking logic.

7.14.1 Resource Overriding Configuration

The `override-resource-registry` key in YAML causes the overridden resource to be replaced by the overriding resource. The resource that is overridden is disabled. Here is a code sample showing the override configuration key in YAML:

```
override-resource-registry:
- new: new-document
  origin: document
- new: new-user
  origin: user
```

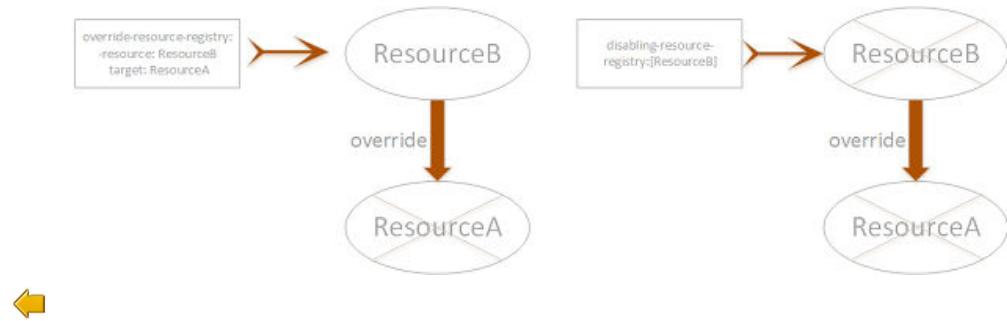
Key	Description
new	The name of the overriding resource
origin	The name of the resource that is overridden

7.14.1.1 Scenarios and Expected Results

Here are some examples of scenarios along with their expected results

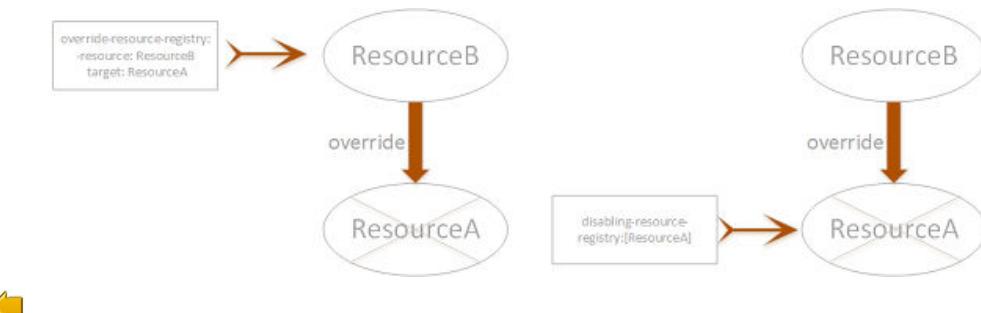
➤ Example 7-95: Disable a Resource that Overrides Another Resource

When disabling an overriding resource, the overriding (Resource B) is disabled and the resource that is overridden resource (Resource A) is also disabled.



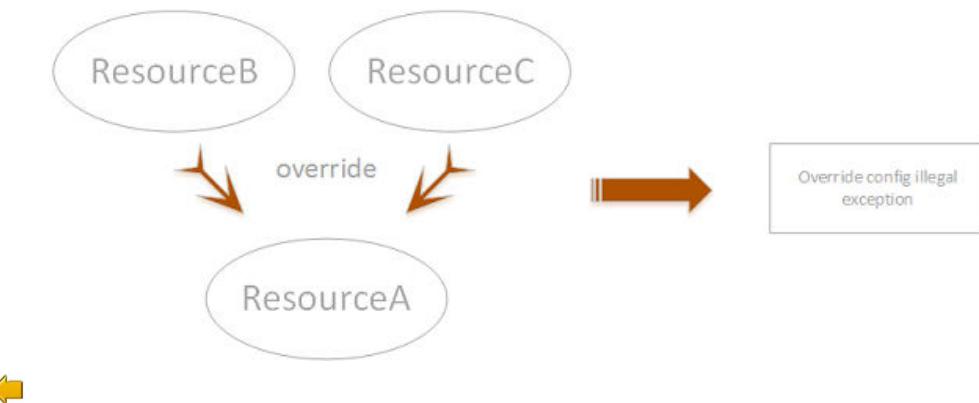
➤ Example 7-96: Disable a Resource that is Overridden by Another Resource

In this example, no change takes place in the `disabling-resource-registry` setting since `<Resource A>` has been disabled in the `override-resource-registry` setting. The overriding resource (Resource B) remains active and the overridden resource (Resource A) remains disabled.



➡ **Example 7-97: Two New Resources Incorrectly Override the Same Resource**

In this example, two new resources (Resource B and Resource C) are incorrectly configured to override the same resource (resource A). This configuration error leads to an `override config illegal` exception when deployed.



7.14.1.2 Working With a URI Template When Overriding

When you override a resource, the new (overriding) resource must have the same parameters, in the same order, as the resource that is being overridden. Foundation REST API validates the URL template of new resources to ensure that the new resource meets the preceding constraint. When the new resource being overridden does not meet the preceding constraint, initialization fails with an `override config illegal` exception and an exception detail message is written to the log.

7.14.1.3 How Resource Overriding Impacts Batchable Resources

When a resource is overridden:

- It is no longer in batchable-resources
- It is no longer in non-batchable-resources

The overriding resource can use the `<BatchProhibiton>` annotation. Here is a listing of the impact that the `<BatchProhibition>` annotation has on the resource.

- The `<BatchProhibiton>` is *true*:
 - The resource is no longer in batchable-resources
 - The resource is in non-batchable-resources
- The `<BatchProhibiton>` is *false*:
 - The resource is in batchable-resources
 - The resource is no longer in non-batchable-resources

7.15 Turning off XML, JSON, or HAL media types



Note: HAL is expressed using JSON. In this section, when we mention JSON, we specifically mean `application/vnd.emc.documentum+json`.

Foundation REST API supports XML, JSON, and HAL media types out-of-the-box. Foundation REST API extensibility allows you to build application specific Foundation REST API. Typically, resources are developed in one of XML, JSON, or HAL but not all three. This section discusses how to turn off the media type (XML, JSON, or HAL) that you are not using.

An additional YAML configuration parameter has been exposed during the deployment phase so you can choose the media type that is supported. Here's a code sample to help illustrate the point:

```
---
##### SET SUPPORTED MEDIA TYPES (FOR INTERNAL USE) #####
# # Sets the supported media types at runtime. XML only, JSON only, or HAL
# # only support can be configured. The setting applies to both core resources
# # and extended resources.
# # The following shows the syntax for a resource view definition:
# #
# #   - media-type: <media type name>
# #
# # There are four choices:
# #   - default, supporting all of XML, JSON, and HAL at runtime.
# #   - xml, supporting XML media type only at runtime.
# #   - json, supporting JSON media type only at runtime.
# #   - hal, supporting HAL media type only at runtime.
# #
# # The default media type support is 'default'.
#####
media-type-registry:
# - media-type: default
```

By default, all of XML, JSON, and HAL media types are supported:

- When *media-type:xml*, the only and default media type is XML. Any request with a JSON or HAL message body will get a 415 status code, and any request for a JSON or HAL Response will get a 406 status code. The error message is always in XML.
- When *media-type:json*, the only and default media type is JSON. Any request with a XML or HAL message body will get a 415 status code, and any request for an XML or HAL Response will get a 406 status code. The error message is always in JSON.
- When *media-type:hal*, the only and default media type is HAL. Any request with a XML or JSON message body gets a 415 status code, and any request for an XML or JSON Response will get a 406 status code. The error message is always in HAL JSON.
- When *media-type:default*, all of XML, JSON, and HAL are supported.



Caution

The setting has no impact on resource controller implementations. It just directs the Spring marshalling framework to choose a supported media type when resolving an inbound or outbound HTTP message.

You can declare a specific media type to support in resource controllers by using the following declarations:

- *@RequestMapping* consumes
- *@RequestMapping*. produces

Here are a two examples to help illustrate the point.

➡ Example 7-98: Declare all Three Media Types

This example shows you how to declare all three XML, JSON, and HAL media types:

```
@RequestMapping(
    method = RequestMethod.POST,
    produces ={ SupportedMediaTypes.APPLICATION_VND_DCTM_JSON_STRING,
    SupportedMediaTypes.APPLICATION_VND_DCTM_XML_STRING,
    SupportedMediaTypes.APPLICATION_VND_DCTM_HAL_STRING,
    MediaType.APPLICATION_JSON_VALUE,
    MediaType.APPLICATION_XML_VALUE
    MediaType.APPLICATION_HAL_VALUE
    },
    consumes ={ SupportedMediaTypes.APPLICATION_VND_DCTM_JSON_STRING,
    SupportedMediaTypes.APPLICATION_VND_DCTM_XML_STRING
    SupportedMediaTypes.APPLICATION_VND_DCTM_HAL_STRING
    })
@RequestBody
@ResponseStatus(HttpStatus.CREATED)
public CabinetObject createCabinet()
```



➡ Example 7-99: Declare Only XML Media Type

This example shows you how to declare just the XML media type: Shown in bold in the code sample is the *produces* attribute, which sets the supported media type for messages that are created on the server. In this case, we have chosen to have XML messages created and sent from the server.

There is also a *consumes* attribute, which defines the media type. that is supported by your Foundation REST API client. In this case, the supported media type is XML of course.

```
@RequestMapping(
    method = RequestMethod.POST,
    produces ={
        SupportedMediaTypes.APPLICATION_VND_DCTM_XML_STRING,
        MediaType.APPLICATION_XML_VALUE
    },
    consumes ={
        SupportedMediaTypes.APPLICATION_VND_DCTM_XML_STRING
})
@ResponseBody
@ResponseStatus(HttpStatus.CREATED)
public CabinetObject createCabinet()
```



Tip: The preferred solution is to declare that all three media types (XML, JSON, and HAL) are supported in resource controllers, and use your YAML file to set the specific media type supported. There is no additional development cost when you declare your media type support in resource controllers. However, you gain the flexibility to use your YAML file to configure specific media type support.

7.16 Creating custom error code mapping files

Foundation REST API allows the client to create error code mapping JSON files. The name of a custom error code mapping JSON file must follow this pattern: `rest-*error-mapping.json`

Where * can be any string of characters. For example: `rest-myresource-error-mapping.json`.

You must ensure that the package where you create your custom error code mapping files is specified in the property `rest.ext.error.code.mapping.packages` within the `rest-api-runtime.properties` file.

7.17 Creating custom message files

Foundation REST API allows you to create custom message files. You can read custom messages with `MessageBundle` as shown in the following code snippet:

```
public String feedTitle() {  
    return MessageBundle.INSTANCE.get("TITLE_OF_MYRESOURCE_COLLECTION");  
}
```

The name of a custom message file must follow this pattern:

```
rest-*messages*.properties
```

* stands for any string of characters.

Example: `rest-myresource-messages-new.properties`

Additionally, the package where you create custom message files must be listed in the `rest.extension.message.packages` runtime property, which is found within the `rest-api-runtime` properties file.

7.18 HTTP compression

Many application servers support server-to-client compression, where the server returns a compressed Response body to client. However, different servers have different capabilities for HTTP compression support with specific or distinct configurations being required to enable this kind of support and client to server compression is rarely supported.

Foundation REST API supports a standalone HTTP compression solution for generic web applications. This HTTP compression solution supports server-to-client and client-to-server compression. Client-to-server compression, where a client uploads a compressed request body to the server, is only supported by the POST and PUT HTTP methods.

7.18.1 Supported compression schemas

The following widely used compression and decompression algorithms are supported by the Foundation REST API:

- `<gzip>`
- `<x-gzip>`
- `<deflate>`

7.18.1.1 Compression Notes

- When a specific compression schema has not been defined while attempting to compress a response body, for example <Accept-Encoding: *>, the Foundation REST API server defaults to using the <gzip> schema to compress the Response body.
- When an unacceptable compression schema, such as <Accept-Encoding: compress>, is defined while attempting to compress a response, the Foundation REST API server ignores the invalid schema that was defined and returns the Response without any compression.
- When multiple algorithms, such as <Accept-Encoding: deflate;q=0.4,gzip;q=0.6>, are defined while attempting to perform compression, the Foundation REST API server selects the best encoding schema to compress the Response body. In the preceding example, the server would select the <gzip> compression schema.
- When the Foundation REST API server receives an unacceptable decompression schema, such as <Content-Encoding: compress>, from client while attempting to decompress a request body, the Foundation REST API server will throw an <HTTP status code 400> exception.
- When the specified Header <Content-Encoding> schema is not compatible with the actual compressed schema used for the Request body, an HTTP error status code <400> exception is returned by the Foundation REST API server.

For example, the Request body is compressed using the <DEFLATE> algorithm and the client specifies <Content Encoding: gzip>. This results in a <400> HTTP error status code being returned by the Foundation REST API server.

7.18.2 Supported configurations

In order to provide the greatest level of application server compatibility for compression, the Foundation REST API server supports the following runtime configurations within the runtime properties file.

Example 7-100: REST Compression configuration

Here is a code sample that shows the properties for REST compression configuration:

```
#-----+  
#      Compression Service Configuration      |  
#-----+  
#  
# This section contains the configuration of server Response message compression, the  
# Response message includes both the xml/json message and the content binaries. The  
# compression service is only triggered when the Request header Accept-Encoding matches  
# any one of the following supported compression schemas:  
# - gzip  
# - x-gzip  
# - deflate  
#  
# In addition to the above, the Request and Response must also match the conditions  
# listed  
# in all of the properties shown below.  
#
```

```
# This property specifies the size, in bytes, of the smallest Response that will be
# compressed. When less than the value of ${rest.compression.threshold} bytes are written
# to the Response, it is not compressed. In this case, the Response goes back to the
client
# unmodified. When the value for this property is 0, compression always begins
immediately.
# This property defaults to 10485760 bytes (10MB).
rest.compression.threshold=
#
# When specified, this property is treated as a comma-separated list of content types
# For example, application/xml,application/json
# The REST server only attempts to compress responses whose content type is compatible
# with one of the values set for this property.
# This property defaults to all media types.
rest.compression.include.content.types=
#
# This property is the same as {rest.compression.include.content.types}, but specifies a
# list of content types not to compress. By default no content type is excluded.
# However note that any content type that indicates a compressed format, such as the
# following content types are never compressed:
# - application/gzip
# - application/x-compress
# - application/deflate
# - application/x-gzip)
#
# Also note, when any value is specified in both {rest.compression.include.content.types}
# and here in this property, the specific content type is removed from the included
content
# types, which causes the Response not to be compressed.
rest.compression.exclude.content.types=
#
# When specified, this property is treated as a comma-separated list of regular
expressions
# of the types that are accepted by the regular expression pattern. The pattern must
match
# those paths that should be compressed. Anything else will not be compressed.
# This property defaults to all paths.
# "Paths" here means values returned by HttpServletRequest.getRequestURI().
# Note that the regular expression must match the file name exactly.
# For example the regular expression pattern "/static/" does not match everything
# containing the string "/static/". To do that, use ".*/static/.*" instead.
rest.compression.include.path.patterns=
#
# This property is same as {rest.compression.include.path.patterns}, but specifies a list
# of patterns that match paths that should not be compressed. Note the archived-contents
# path pattern is always excluded, specifically the value ".*/archived-contents.*" is
# always set in this property, and when any value is specified both in
# {rest.compression.include.path.patterns} and here simultaneously, the specific path is
# removed from the included paths, and the Response with that path is not compressed.
rest.compression.exclude.path.patterns=
#
# When specified, this property is treated as a comma-separated list of regular
# expressions, and only those Requests with User-Agent headers whose value matches one of
# these regular expressions is compressed.
# This property defaults to all user agents.
rest.compression.include.user.agent.patterns=
#
# Similar to {rest.compression.include.user.agent.patterns}, all requests whose User-
Agent
# header matches one of the patterns set for this property are not compressed. By default
# no user agent is excluded, but when any value is specified in
# {rest.compression.include.user.agent.patterns} and here at the same time, the specific
# user agent is removed from the list of included agents, and the Response with that user
# agent is not compressed.
rest.compression.exclude.user.agent.patterns=
#
# When specified, this property is treated as a comma-separated list of regular
# expressions. A Request with User-Agent headers whose value matches one of these regular
```

```
# expressions causes the Response to be returned without the Vary header.
rest.compression.no.vary.header.agent.patterns=
```



7.18.3 Client and Server compression samples

▶ Example 7-101: Client-to-server compression

The following code sample shows you how to use the Content-Encoding Header to upload document content while using a specific compression schema:

```
POST /dctm-rest/repositories/REPO/objects/09000005800051ae/contents?
overwrite=true HTTP/1.1
Host: localhost:8080
Authorization: Basic ZG1hZG1pbjpwYXNzd29yZA==
Content-Encoding: gzip

... (The remainder of the Request-body) ...
... (The compressed binary file<middle.log.gz>) ...

Status 201 Created
Content-Type application/vnd.emc.documentum+json;charset=UTF-8
Location http://localhost:8080/dctm-rest/repositories/REPO/objects/09000005800051ae/
contents/content?format=unknown&modifier=&page=0
Vary Accept-Encoding

... [The remaining Response body] ...
```



▶ Example 7-102: Server-to-client compression

This sample demonstrates how to use the Accept-Encoding Header to download compressed document content with a specified compression algorithm.

The Response Header Content-Encoding defines the actual compression schema used for the Response body, which in this case is <gzip>. The Foundation REST API server returns the compressed file as a formatted stream.

```
GET /dctm-rest/repositories/REPO/objects/09000005800051ae/content-media?
format=unknown&modifier=&page=0 HTTP/1.1
Host: localhost:8080
Accept-Encoding: deflate;q=0.4,gzip;q=0.6
Authorization: Basic ZG1hZG1pbjpwYXNzd29yZA==

Status 200 OK
Content-Encoding gzip
Content-Type application/octet-stream
Vary Accept-Encoding
ETag W/"K1cw+IwVjnDwltiMr2KGuCm8HyQzcMvm7V1KhtWWtJM="

... [The compressed binary file] ...
```



7.19 Tutorial: Foundation REST API extensibility development

This tutorial walks through a reference implementation of custom resources using the Documentum REST Services Extensibility feature. It is a showcase for bringing various Core REST and Spring framework eco-system technologies together to implement REST resources for your domain types. You'll learn these technologies from this tutorial:

- Design and implement new resources
- Add new links to Core resources
- Package custom and Core resources into a single WAR package

7.19.1 What to Build First

Assume that you need to create an alias set collection resource for the `dm_alias_set` type objects in the repository. The resource is feed-based and accepts HTTP GET method. The alias set collection resource is designed as follows:

Table 7-9: Resource design of the alias set collection

Resource	URI	HTTP Methods	Media Types
Alias Set Collection	/repositories/{repositoryName}/alias-sets{?view, filter, links, inline, page, items-per-page, include-total, sort}	GET	application/atom+xmlapplication/vnd.emc.documentum+json

The resource supports both XML and JSON representations in conformity with Core REST representations.

XML representation for the alias set collection resource

```
<feed xmlns="http://www.w3.org/2005/Atom"
  xmlns:dm="http://identifiers.emc.com/vocab/documentum"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <id>http://localhost:8080/acme-rest/repositories/REPO/alias-sets</id>
  <title>Alias Sets</title>
  ...
  <link rel="self" href="http://localhost:8080/acme-rest/repositories/REPO/alias-sets"/>
  <entry>
    <id>http://localhost:8080/acme-rest/repositories/REPO/alias-sets/6600208080000105</id>
    <title>AdminAccess</title>
  ...

```

```
<content type="application/xml"
src="http://localhost:8080/acme-rest/repositories/REPO/alias-sets/
6600208080000105"/>
<link rel="edit" href="http://localhost:8080/acme-rest/repositories/
REPO/alias-sets/6600208080000105"/>
</entry>
...

```

JSON representation for the alias set collection resource

```
{
  id: "http://localhost:8080/acme-rest/repositories/REPO/alias-sets",
  title: "Alias Sets",
  ...
  links: [{rel: "self", href: "http://localhost:8080/acme-rest/
repositories/REPO/alias-sets"}],
  entries: [
    {
      id: "http://localhost:8080/acme-rest/repositories/REPO/alias-sets/
6600208080000105",
      title: "AdminAccess",
      ...
      content: {
        type: "application/vnd.emc.documentum+json",
        src: "http://localhost:8080/acme-rest/repositories/REPO/alias-sets/
6600208080000105"
      },
      links: [{rel: "edit", href: "http://localhost:8080/acme-rest/
repositories/REPO/alias-sets/6600208080000105"}]
    },
    ...
  ]
}
```



Note: XML representation for collections must conform to the atom feed; and JSON representation for collections must conform to EDAA.

With the alias set collection resource in your mind, it's quite natural to think about building single alias set resources. The single alias set resource can be embedded in the feed representation of the alias set collection resource. The alias set resource is designed as follows:

Table 7-10: Resource design of the alias set resource

Resource	URI	HTTP Methods	Media Types
Alias Set	/repositories/{repositoryName}/alias-sets/{aliasSetId} {?view,filter,links }	GET	application/vnd.emc.docuemntu m+xml application/vnd.emc.documentum +json

Just like the alias set collection, single alias set resources support both XML and JSON representations.

XML representation for the alias set resource

```
<?xml version='1.0' encoding='UTF-8'?>
<alias-set xmlns="http://identifiers.emc.com/vocab/documentum"
xmlns:dm="http://identifiers.emc.com/vocab/documentum"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```

xsi:type="dm_alias_set" definition="http://localhost:8080/acme-rest/
repositories/REPO/types/dm_alias_set">
<properties>
<owner_name>Administrator</owner_name>
<object_name>AdminAccess</object_name>
...
</properties>
<links>
<link rel="self" href="http://localhost:8080/acme-rest/repositories/
REPO/alias-sets/6600208080000105"/>
<link rel="author" href="http://localhost:8080/acme-rest/repositories/
REPO/users/Administrator"/>
</links>
</alias-set>

```

JSON representation for the alias set resource

```

{
  "name": "alias-set",
  "type": "dm_alias_set",
  "definition": "http://localhost:8080/acme-rest/repositories/REPO/
  types/dm_alias_set",
  "properties": {
    "owner_name": "Administrator",
    "object_name": "AdminAccess",
    ...
  },
  "links": [
    {"rel": "self", "href": "http://localhost:8080/acme-rest/repositories/
  REPO/alias-sets/6600208080000105"}, 
    {"rel": "author", "href": "http://localhost:8080/acme-rest/repositories/
  REPO/users/Administrator"}
  ]
}

```

7.19.2 Quickstart

Foundation REST API SDK provides you with a convenient way to set up your first custom resource project: using the Maven archetype. Refer to [Get Started With the Development Kit](#) for details. The code structure we will build for both resources is shown as follows.

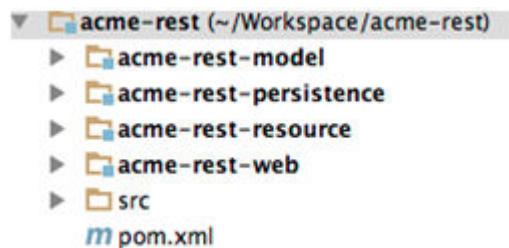


Figure 7-9: Code Structure

As the diagram shows, we create four modules for the project. The model module creates the serializable alias set data model. The persistence module creates the persistence API to get and update the alias set object using Foundation Java API. The resource module creates the resource controllers. And the web module creates the WAR file for the entire custom services.

7.19.3 Creating Resource Model

The data model class defines the data structure of the resource representation for both input and output. With Foundation REST API annotations, the model Java class has a direct mapping to the resource representation messages.

The Maven archetype project contains a sample model class AliasSet for the alias set. This is the only class added to the model module.

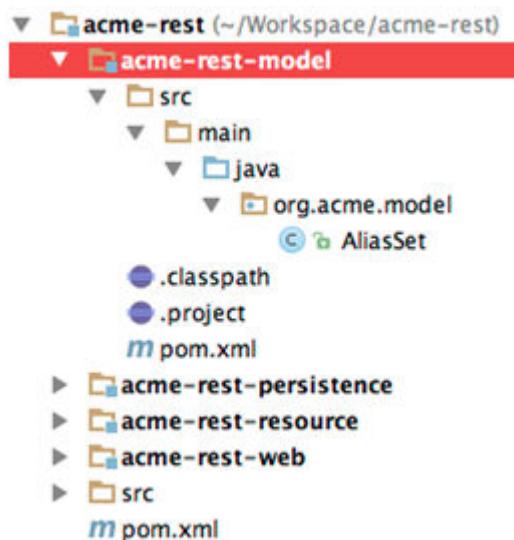


Figure 7-10: Structure of Model

The class definition for AliasSet is shown:

```
/**
 * Data model for dm_alias_set
 */
@SerializableType(value = "alias-set",
xmlNSPrefix = "dm",
xmlNS = "http://identifiers.emc.com/vocab/documentum")
public class AliasSet extends PersistentObject {
public AliasSet() {
setType("dm_alias_set");
}
}
```

@SerializableType is the Foundation REST API Java annotation introduced to design the representation model. And the PersistentObject class is the out-of-box model class that implements properties and links, and allows inheritance. So with this simple class definition, the alias set representation model is built up.

[Foundation REST API Marshalling Framework](#) provides more information about the Foundation REST API annotations.

7.19.3.1 Validating the annotated resource model

Foundation REST API SDK provides a Maven plugin to validate the REST model annotations during the design time. The plugin can be added to the pom file to validate the models during the build process. You can also run the scanner with command lines. A sample of the Maven plugin configuration in the pom file is shown:

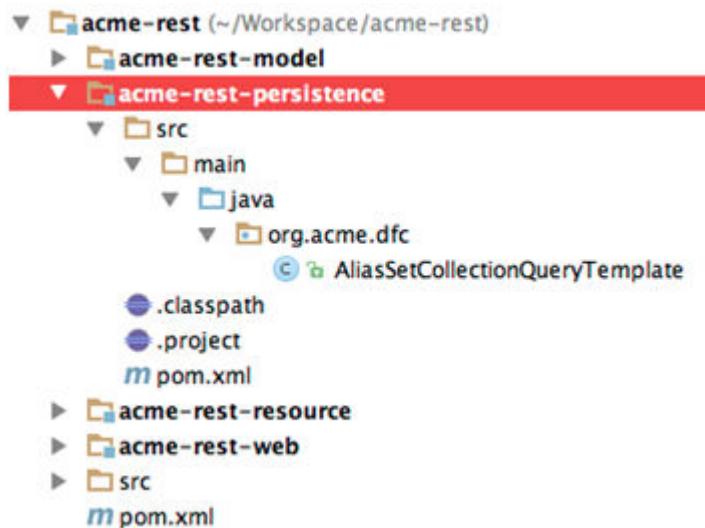
```
<plugin>
<groupId>com.emc.documentum.rest</groupId>
<artifactId>documentum-rest-extension-validating</artifactId>
<version>1.0.</version>
<inherited>true</inherited>
<executions>
<execution>
<id>validate-annotation</id>
<phase>verify</phase>
<goals>
<goal>check-annotation</goal>
</goals>
<configuration>
<input>${project.basedir}/target/${scan.artifactId}-${version}.war</input>
<outputDir>${project.basedir}/target</outputDir>
<isDebug>false</isDebug>
</configuration>
</execution>
</executions>
</plugin>
```

The section [Annotation Scanner](#) provides more information about Foundation REST API annotation validation.

7.19.4 Creating Persistence

The persistence module defines the Foundation Java API interface and implementation for the operations. This module accepts the data model input. Output of this module is also a data model. Foundation REST API SDK library provides a lot of out-of-box persistence APIs that can be used to communicate with Foundation Java API. In this sample project, you can fully leverage Core persistence library to manipulate the alias set object in the repository.

The Maven archetype project only has one class `AliasSetCollectionQueryTemplate`, which customizes the DQL query for the alias set collection.

**Figure 7-11: Structure of Persistence**

This query template is the only implementation class to retrieve a collection of alias set objects. The SDK provides other APIs to facilitate the object collection retrieval. The class definition of the `AliasSetCollectionQueryTemplate` is shown as follows.

```
/*
 * A query template to get alias set collection.
 */
public class AliasSetCollectionQueryTemplate extends PagedQueryTemplate {
@Override
protected List<String> defaultFields() {
return Arrays.asList( "r_object_id", "alias_category",
"alias_name", "alias_value", "object_name", "owner_name");
}

@Override
protected String qualification() { return ""; }

@Override
protected String from() { return "dm_alias_set"; }

@Override
protected List<SortOrder> defaultSorts()
{ return Arrays.asList(new SortOrder("object_name", true)); }

@Override
protected boolean supportRepeatingAttributeQuery() {return true; }
}
```

`PagedQueryTemplate` is an out-of-the-box class that defines the abstraction for the DQL query string construction for a type-specific collection. A collection resource's persistence API can extend this class to build the DQL query string for the returning collection. The next section shows an example about how it is used in the resource controller.

7.19.5 Creating Resource Controller

The resource controller defines the REST resource mapping for the custom resource and is the end point for a resource operation.

The Maven archetype project defines two controllers, for the alias set collection resource and the alias set resource, respectively. The code structure of the resource module is shown as follows.

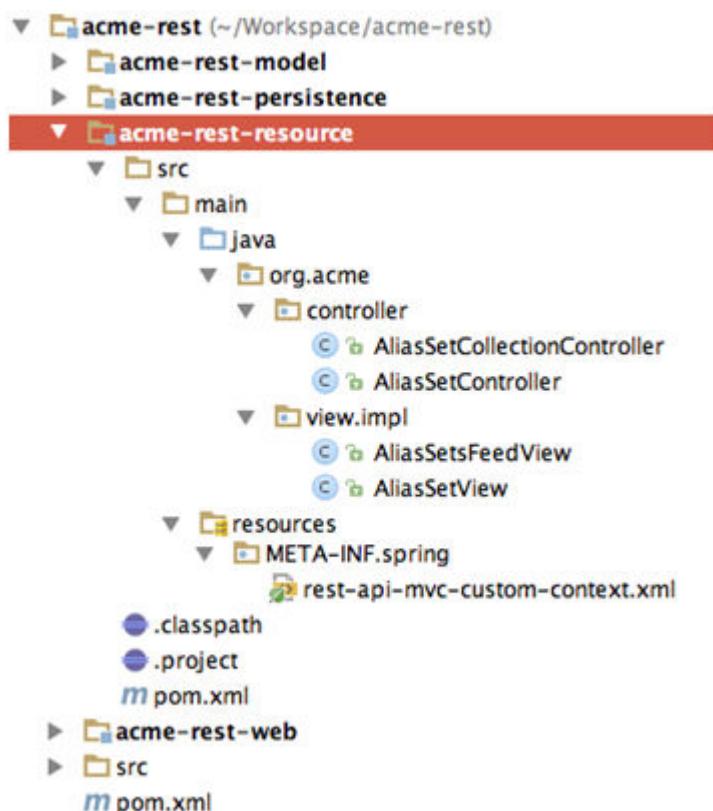


Figure 7-12: Structure of Controller

7.19.5.1 AliasSetCollectionController

The AliasSetCollectionController class is the definition for the alias set collection resource.

```
/**
 * Collection of alias sets in a repository.
 */
@Controller("acme#alias-sets")
@RequestMapping("/repositories/{repositoryName}/alias-sets")
@ResourceViewBinding(AliasSetsFeedView.class)
public class AliasSetCollectionController extends AbstractController {
    @RequestMapping()
    method = RequestMethod.GET,
    produces = {
        SupportedMediaTypes.APPLICATION_VND_DCTM_JSON_STRING,
        MediaType.APPLICATION_ATOM_XML_VALUE,
        MediaType.APPLICATION_JSON_VALUE,
        MediaType.APPLICATION_XML_VALUE
    }
    @ResponseBody
    @ResponseStatus(HttpStatus.OK)
    public AtomFeed getAliasSets(
        @PathVariable("repositoryName") final String repositoryName,
        @TypedParam final CollectionParam param,
        @RequestUri final UriInfo uriInfo)
        throws Exception {
        PagedQueryTemplate template = new AliasSetCollectionQueryTemplate()
            .filter(param.getFilterQualification())
            .order(param.getSortSpec().get())
            .page(param.getPagingParam().getPage(),
                param.getPagingParam().getItemsPerPage());
        PagedDataRetriever<AliasSet> dataRetriever =
            new PersistentDataRetriever<AliasSet>(
                template,
                param.getPagingParam().getPage(),
                param.getPagingParam().getItemsPerPage(),
                param.getPagingParam().isIncludeTotal(),
                param.getAttributeView(),
                AliasSet.class);
        return getRenderedPage(
            repositoryName,
            dataRetriever.get(),
            param.isLinks(),
            param.isInline(),
            uriInfo,
            null);
    }
}
```

The resource controller uses a number of Java annotations. They are mainly divided into two categories.

Spring annotations

- `@Controller` - a Spring annotation to define a resource controller. It is strongly recommended to assign a unique name for the controller as Core REST runtime uses this name to identify a resource.
- `@RequestMapping` - a Spring annotation to define the URI, Headers, and parameters for a resource. This annotation can be applied to the controller class level or individual controller method level. Spring has a complicated match pattern to compare whether two controller methods or controller classes use the same URI mapping. For example, you can define two controller methods with

the same URI, but with different HTTP methods; even that you can define two controller methods with the same URI and HTTP methods, but with different HTTP Headers or query parameters.

- `@ResponseBody` and `@ResponseStatus` - Spring annotations to automatically resolve the model and view and HTTP status for the resource. `@ResponseBody` is mandatory for custom resource development.
- `@PathVariable` – a Spring annotation to extract variables from a path.

Foundation REST API annotations

- `@TypedParam final CollectionParam param` - a Foundation REST API annotation customizing Spring query parameters which assemble feed resource-related query parameters together, e.g., inline, page, items-per-page.
- `@RequestUri` - A Foundation REST API annotation customizing Spring query parameters which assembles feed resource related query parameters together, e.g., inline, page, items-per-page.
- `@ResourceViewBinding` - a Foundation REST API annotation to bind a resource to a default view. With the view binding, the correct view class is instantiated to resolve the links and atom attributes for the alias set resource representation. This annotation can be put on the controller methods, too.

The Foundation REST API SDK also provides abstraction, model, and utility classes for code reuse in custom resource development.

Foundation REST API classes

- `AbstractController` - the Core controller abstraction. All custom resources should extend this abstract class.
- `PagedDataRetriever` - the Core persistence API to execute a query for a paged query template and return a page for the query.
- `AtomFeed` - the Core model class for Atom feeds. All collection resources are returned as an atom feed.

In summary, there aren't many lines to implement an alias set collection resource, but each piece provides very useful information.

7.19.5.2 AliasSetController

Let's move on to the controller for the alias set resource. The class definition is shown as follows.

```
/**  
 * An alias set.  
 */  
@Controller("acme#alias-set")  
@RequestMapping("/{repositoryName}/alias-sets/{aliasSetId}")  
@ResourceViewBinding(AliasSetView.class, queryTypes="dm_alias_type")  
public class AliasSetController extends AbstractController {  
    @Autowired  
    private SysObjectManager sysObjectManager;
```

```

    @RequestMapping(
        value = ALIAS_SET_URI_PATTERN,
        method = RequestMethod.GET, produces = {
            SupportedMediaTypes.APPLICATION_VND_DCTM_JSON_STRING,
            MediaType.APPLICATION_ATOM_XML_VALUE,
            MediaType.APPLICATION_JSON_VALUE,
            MediaType.APPLICATION_XML_VALUE
        })
    @ResponseBody
    @ResponseStatus(HttpStatus.OK)
    public AliasSet getAliasSet(
        @PathVariable("repositoryName") final String repositoryName,
        @PathVariable("aliasSetId") final String aliasSetId,
        @TypedParam final SingleParam param,
        @RequestUri final UriInfo uriInfo)
        throws Exception {
        AliasSet aliasSet = sysObjectManager.getPyObjectByQualification(
            String.format("dm_alias_set where r_object_id='%s'", aliasSetId),
            param.getAttributeView(),
            AliasSet.class
        );
        return getRenderedObject(repositoryName, aliasSet, param.isLinks(),
            uriInfo, null);
    }
}

```

The controller is implemented similarly to the `AliasSetCollectionController`. It calls `SysObjectManager` to manipulate the sysobjects. Since an alias set is nothing special other than a sysobject, you can reuse existing Core REST APIs to operate the alias set.

7.19.6 Spring Context Configuration

In addition to the resource controller definition, you need to define the Spring context configuration to scan the controller classes during the server startup. This Spring configuration file tells the Spring framework where to load the controllers.



Note: You must use the `com.emc.documentum.rest.context.ComponentScanExcludeFilter` exclude filter when you use the Spring `@ComponentScan` annotation in your custom defined configuration class. This exclude filter ensures that the Spring framework loads all of the resources that you have defined.

Here is a code sample that demonstrates this technique:

```

@Configuration
@ComponentScan(basePackages = "org.acme",
    excludeFilters = { @ComponentScan.Filter(type = FilterType.CUSTOM,
        value = { com.emc.documentum.rest.context.ComponentScanExcludeFilter.class }) })
public class ExtensionContextConfig {
}

```

You must ensure that the package where you created the `ExtensionContextConfig` class is specified in the `rest.context.config.location` property in the `rest-api-runtime.properties` file.

The preceding information is equivalent to the following XML configuration:

```

<context:component-scan base-package="org.acme" use-default-filters="true">
    <context:exclude-filter type="custom">

```

```
    expression="com.emc.documentum.rest.context.ComponentScanExcludeFilter"/>
</context:component-scan>
```

7.19.7 Creating Resource View

Till now, the resource implementation has been able to return the alias set feed and alias set resource following the URIs, but the link relations and atom attributes for them are still missing. In this section, we are going to create views for both resources to customize the links and atom attributes.

7.19.7.1 AliasSetsFeedView

The `AliasSetsFeedView` class defines the atom feed attributes and links for the alias set collection resource.

View definition of AliasSets

```
/**
 * View for the alias set feed.
 */
@FeedViewBinding(AliasSetView.class)
public class AliasSetsFeedView extends FeedableView<AliasSet> {
    public AliasSetsFeedView(Page<AliasSet>
        page, UriInfo uriInfo, String repositoryName,
        boolean returnLinks, Map<String, Object> others) {
        super(page, uriInfo, repositoryName, returnLinks, others);
    }

    @Override
    public String feedTitle() { return "Alias Sets"; }

    @Override
    public Date feedUpdated() { return new Date(); }
}
```

The feed view class must extend `FeedableView<T>` and implement its own feed attributes.

`FeedableView<T>` - the Core REST feed view abstraction. It provides built-in link relations for an atom feed. By implementing this view, the alias set collection resource obtains the built-in self link and pagination links. There are also default implementations for several atom feed attributes, like atom authors, atom ID, and so on.

`@FeedViewBinding` - a Core REST annotation to bind a feed view to the corresponding entry views.

7.19.7.2 AliasSetView

The AliasSetView class defines the atom entry attributes and links for the alias set.

View definition of AliasSet

```
/*
 * View for the alias set.
 */
@DataViewBinding(AliasSet.class)
public class AliasSetView extends Persistent DataView<AliasSet> {
    public AliasSetView(AliasSet aliasSet, UriInfo uriInfo, String repositoryName,
        boolean returnLinks, Map<String, Object> others) {
        super(aliasSet, uriInfo, repositoryName, returnLinks, others);
    }

    @Override
    public String entryTitle() { return (String) serializableData.
        getMandatoryAttribute("object_name"); }

    @Override
    public String entrySummary() { return (String) serializableData.
        getMandatoryAttribute("object_name"); }

    @Override
    public Date entryUpdated() { return new Date(); }

    @Override
    public Date entryPublished() { return new Date(); }

    @Override
    public List<AtomAuthor> entryAuthors() {
        String ownerLink = ResourceUriBuilder
            .onResource("user")
            .pathVariables((String)serializableData.getAttributeByName("owner_name"));
        return Arrays.asList(new AtomAuthor(owner, ownerLink, null));
    }

    @Override
    public AliasSet entryContent() { return data(); }

    @Override
    public String entrySrc() { return canonicalResourceUri(boolean validate); }

    @Override
    public void customize() {
        String ownerLink = ResourceUriBuilder
            .onResource("user")
            .pathVariables((String)serializableData.getAttributeByName("owner_name"));
    }

    @Override
    public String canonicalResourceUri(boolean validate) {
        return UriFactory(validate).buildUriByTemplateName(
            "X_ALIAS_SET_URI_TEMPLATE",
            Collections.singletonMap("aliasSetId", serializableData.getId()));
    }
}
```

The domain model view must extend `EntryableView<T>` and implement its own entry attributes and resource links. In this sample class, `AliasSetView` extends `Persistent DataView` which provides the default implementation for making links. When an object is accessible through multiple resource URIs, the canonical resource URI represents the primary resource URI for this object. The method `canonicalResourceUri` returns the canonical resource URI dynamically with a validation option.

- When validate is *true* and the corresponding resource for the URI is inactive, the returned URI is {@link com.emc.documentum.rest.http.UriFactory#INACTIVE_URL}
- When validate is *false*, the returned URI is a static URI defined by this method.

@DataViewBinding - a Core REST annotation to mark on a single data object resource view. This view binding binds a view to a resource model.

Refer to the section [Foundation REST API MVC](#) for more information about the Foundation REST API MVC programming pattern.

7.19.8 Adding More HTTP Methods

The Maven archetype project only demonstrates the GET method on the alias set collection resource and the alias set resource. Obviously in a real application, object creation, modification, and deletion are needed, too. This section walks through the design and the implementation of object creation and deletion.

7.19.8.1 Creating an Alias Set Object

A good practice to create a new object is to POST the new resource to the corresponding collection resource. As you have the alias set collection resource, let's create alias set resources with the HTTP POST method on the alias set collection resource.

Table 7-11: Resource design of the alias set resource

Resource	URI	HTTP Methods	Media Types
Alias Set	/repositories/{repositoryName}/alias-sets{?view,filter,links,inline,page,items-per-page,include-total,sort}	GET	application/atom+xml;application/vnd.emc.documentum+json
POST	application/vnd.emc.documentum+xml application/vnd.emc.documentum+json		

The new controller method is defined as follows.

```
@Controller("acme#alias-sets")
@RequestMapping("/repositories/{repositoryName}/alias-sets")
@ResourceViewBinding(AliasSetsFeedView.class)
public class AliasSetCollectionController extends AbstractController {
...
@RequestMapping(method = RequestMethod.POST,
produces = {
```

```

SupportedMediaTypes.APPLICATION_VND_DCTM_JSON_STRING,
SupportedMediaTypes.APPLICATION_VND_DCTM_XML_STRING,
MediaType.APPLICATION_JSON_VALUE,
MediaType.APPLICATION_XML_VALUE
})
@ResponseBody
@ResponseStatus(HttpStatus.CREATED)
@ResourceViewBinding(AliasSetView.class)
public AliasSet createAliasSet(
@PathVariable("repositoryName") final String repositoryName,
@RequestBody final AliasSet aliasSet,
@RequestUri final UriInfo uriInfo)
throws DfException {
AliasSet createdRelation = sysObjectManager.
createObject(aliasSet, false, AttributeView.ALL);
Map<String, Object> others = new HashMap<String, Object>();
others.put(ViewParams.POST_FROM_COLLECTION, true);
return getRenderedObject(repositoryName, createdRelation,
true, uriInfo, others);
}

@Autowired
private SysObjectManager sysObjectManager;
}

```



Notes

- A `ViewParams.POST_FROM_COLLECTION` parameter is needed on the view parameters for any controller create method
- With the annotation `@ResourceViewBinding` on the controller method, the same `AliasSetView` is reused to render alias set resource responses
- A create operation Response must have a Location Header pointing to the new resource URI. Foundation REST API adds the Location Header automatically for controller methods in case following conditions are met:
 - The method returns a `Linkable` object.
 - The method has the annotation `@ResponseBody`
 - The method has the annotation `@ResponseStatus(HttpStatus.CREATED)`

7.19.8.2 Deleting an Alias Set Object

It is nature to perform an HTTP `DELETE` method on the alias set resource to delete an alias set. A delete method `removeAliasSet` is defined on `AliasSetController` as shown in the following sample:

```

@Controller("acme#alias-set")
@RequestMapping("/repositories/{repositoryName}/alias-sets/{aliasSetId}")
@ResourceViewBinding(AliasSetView.class)
public class AliasSetController extends AbstractController {
...
@RequestMapping(method = RequestMethod.DELETE)
@ResponseStatus(HttpStatus.NO_CONTENT)
public void removeAliasSet(
@PathVariable("repositoryName") final String repositoryName,
@PathVariable("aliasSetId") final String aliasSetId,
@RequestUri final UriInfo uriInfo)
throws Exception {
sysObjectManager.deleteSysObject(aliasSetId, true, true,
DeleteVersionPolicy.ALL);
}

```

```
}
}
```

7.19.9 Making Resources Queryable

Now the alias set resource is implemented with the `dm_alias_set` type. But how to make Core DQL resource to return resource links for the query result of `dm_alias_set` type? The answer is to add a new attribute to the `@ResourceViewBinding` annotation on the `AliasSetController` class.

```
@Controller("acme#alias-set")
@RequestMapping("/repositories/{repositoryName}/alias-sets/{aliasSetId}")
@ResourceViewBinding(value = AliasSetView.class, queryTypes = "dm_alias_set")
public class AliasSetController extends AbstractController {
...
}
```

`queryTypes` makes the alias set resource linkable from DQL query result. Here is an example.

```
<?xml version='1.0' encoding='UTF-8'?>
<feed xmlns="http://www.w3.org/2005/Atom"
      xmlns:dm="http://identifiers.emc.com/vocab/documentum"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
...
<link rel="self" href="http://localhost:8080/acme-rest/repositories/
REPO?dql=select%20*%20from%20dm_alias_set"/>
<entry>
<id>http://localhost:8080/acme-rest/repositories/REPO?
dql=select%20*%20from%20dm_alias_set&index=0</id>
...
<content>
<dm:query-result definition="http://localhost:8080/acme-rest/
repositories/REPO/types/dm_alias_set">
<dm:properties>
<dm:r_object_id>6600208080000100</dm:r_object_id>
<dm:owner_name>Administrators</dm:owner_name>
<dm:object_name>Smart Container</dm:object_name>
<dm:object_description></dm:object_description>
</dm:properties>
</dm:query-result>
</content>
<link rel="edit" href="http://localhost:8080/acme-rest/repositories/
REPO/alias-sets/6600208080000100"/>
</entry>
...
```

7.19.10 Making Resources Linkable

With the preceding implementation, the alias set collection resource and the alias set resource are almost finished. But wait – how does a Foundation REST API client discover the links for the new resources at runtime? The solution is to customize Core resources to add links for your new resources. Here are the links we want to design.

- The alias set collection resource can be found on a repository resource
- The alias set resource can be found from the feed of alias set collection resource

For the second one, as we have implemented the Location Header on alias set creation operation, the POST on the alias set collection resource has been able to

return the link for the newly created alias set resource. So what you actually need is to customize the link relations for the Core repository resource. To do this, simply add a YAML file under the right path and then modify the YAML file as shown:

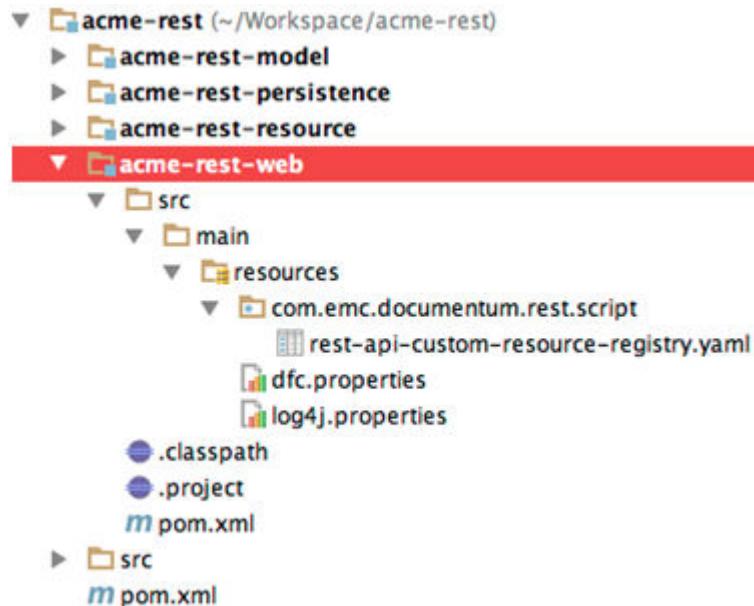


Figure 7-13: YAML File Location

Adding a link relation in repository for the alias set collection:

```
resource-link-registry:
- resource:repository
link-relation: 'http://identifiers.emc.com/linkrel/alias'
uri-template:X_ALIAS_SETS_URI_TEMPLATE
value-mapping: []
```

The section [Adding Links to Core Resources](#) has all the details of this function. Here is the sample.

By doing this, the repository resource now has one more link relation that points to the alias set collection resource.

```
<repository xmlns="http://identifiers.emc.com/vocab/documentum"
xmlns:dm="http://identifiers.emc.com/vocab/documentum"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<id>8320</id>
<name>REPO</name>
<description/>
<server>
<name>REPO</name>
<host>CS71P01</host>
<version xml:space="preserve">7.1.0010.0158Win64.SQLServer</version>
<docbroker>CS71P01</docbroker>
</server>
<links>
<link rel="self" href="http://localhost:8080/acme-rest/repositories/REPO"/>
<link rel="http://identifiers.emc.com/linkrel/alias"
href="http://localhost:8080/acme-rest/repositories/REPO/alias-sets"/>
```

```
...  
</links>  
</repository>
```

7.19.11 Making Resources Non-Batchable (Optional)

By default, all custom resources are batchable. If you want to make a specific custom resource or a certain method non-batchable, put the `@BatchProhibition` annotation on the resource controller or the controller method.

Similarly, you can use the `@TransactionProhibition` annotation to remove the transaction support from a custom resource. Like the `@BatchProhibition` annotation, `@TransactionProhibition` can be applied to both the controller class level and the controller method level.

This tutorial does not demonstrate how to manage the batch property of a custom resource as we want the Alias Set and Alias Set collection resources to stay batchable. You can find more details on making them non-batchable from [Resource: Deciding Whether to be Batchable](#).

7.19.12 Packaging and Deploying Resources

Finally, we need to package the new resources into a WAR file. This can be achieved by the Maven WAR overlay plugin in the web module. The YAML file, which customizes resources, must be put into the class-path of the web module `com/emc/documentum/rest/script`.

Here is the WAR overlay plug-in configuration sample in the web module:

```
<build>  
  <plugins>  
    <plugin>  
      <groupId>org.apache.maven.plugins</groupId>  
      <artifactId>maven-war-plugin</artifactId>  
      <version>2.4</version>  
      <configuration>  
        <overlays>  
          <overlay>  
            <groupId>com.emc.documentum.rest</groupId>  
            <artifactId>documentum-rest-web</artifactId>  
            <excludes />  
          </overlay>  
        </overlays>  
      </configuration>  
    </plugin>  
  </plugins>  
</build>
```

A single `acme-rest-web-0.0.1-SNAPSHOT.war` is built in the directory `/acme-rest-web/target`, which contains both Core resources and new resources.

7.19.13 Troubleshooting

During development, you can enable the DEBUG level log4j logging for the package com.emc.documentum.rest. Resource controller and view information about both core and custom resources will be printed.

7.20 OpenText Documentum Content Management (CM) Accelerated Content Services and OpenText Documentum Content Management (CM) Branch Office Caching Services content upload

For the creation of a new sysobject or the checkout of an existing sysobject, the Foundation REST API server can optionally provide an Accelerated Content Services or Branch Office Caching Services content write URL for the Foundation REST API client to upload the primary of the sysobject to the Accelerated Content Services or Branch Office Caching Services server in a separate call.

The advantage for this solution is that when multiple Branch Office Caching Services servers are deployed in the environment, contents of sysobjects can be uploaded to the local geographic location in the organization. This accelerates the content upload in wide area networks where the bandwidth is critical.

7.20.1 Link relation for Accelerated Content Services and Branch Office Caching Services content upload URLs

The Accelerated Content Services and Branch Office Caching Services content upload URL is provided to Foundation REST API clients as a new link relation **http://identifiers.emc.com/linkrel/distributed-upload** on a Sysobject Resource, at the time when the Sysobject is newly created or checked out. The title of the link relation depends on the location pointed to by the content upload URL.

Accelerated Content Services content upload URL:

```
{  
    "rel": "http://identifiers.emc.com/linkrel/distributed-upload",  
    "title": "ACS",  
    "href": "http://ACS-SERVER:9080/ACS/servlet/ACS?command=write&XXX"  
}
```

Branch Office Caching Services content upload URL:

```
{  
    "rel": "http://identifiers.emc.com/linkrel/distributed-upload",  
    "title": "BOCS-Abuja",  
    "href": "http://BOCS-SERVER:9080/ACS/servlet/BOCS?command=write&XXX"  
}
```

7.20.2 Resources

These three existing resource methods that can produce the Accelerated Content Services and Branch Office Caching Services content upload URLs for sysobjects.

Table 7-12: Resource methods

Resource Method	Impact
POST on Folder Child Documents	When creating a new document under a folder, optional query parameters can be supplied to request the Accelerated Content Services Branch Office Caching Services content upload URL.
POST on Folder Child Objects	When creating a new sysobject under a folder, optional query parameters can be supplied to request the Accelerated Content Services Branch Office Caching Services content upload URL.
PUT on Lock	When checking out an existing sysobject, optional query parameters can be supplied to request the Accelerated Content Services Branch Office Caching Services content upload URL.

Constraints to be noted for the Accelerated Content Services and Branch Office Caching Services content upload URLs.

- There could be multiple Accelerated Content Services servers or Branch Office Caching Services servers deployed for a Documentum CM Server, but only one Accelerated Content Services or Branch Office Caching Services content upload URL can be requested at a time.
- Only a contentless (newly or existed) Sysobject can be requested for Accelerated Content Services and Branch Office Caching Services content upload.
- Only the primary content at page 0 can be requested for an Accelerated Content Services and Branch Office Caching Services content upload.



Note: During the content upload operation using Foundation REST API server and Accelerated Content Services/Branch Office Caching Services, by default, Accelerated Content Services/Branch Office Caching Services returns a response to Documentum REST clients with status code 302 redirection including the location header information. To avoid the default behavior for the Documentum REST clients that cannot handle the 302 redirection, a new boolean type custom header called X-NO-REDIRECTION is introduced. The Documentum REST clients sends this header with a value set to true to the Foundation REST API server where these clients requires a response status code 200 from either Accelerated Content Services/Branch Office Caching Services including the location header information. The location header

information contains an URL to Documentum CM Server to access the sysobject for which the content upload was successful.

7.20.3 Common query parameters

These common parameters can be applied to three relevant resources for the Accelerated Content Services and Branch Office Caching Services content upload.

Table 7-13: Common query parameters

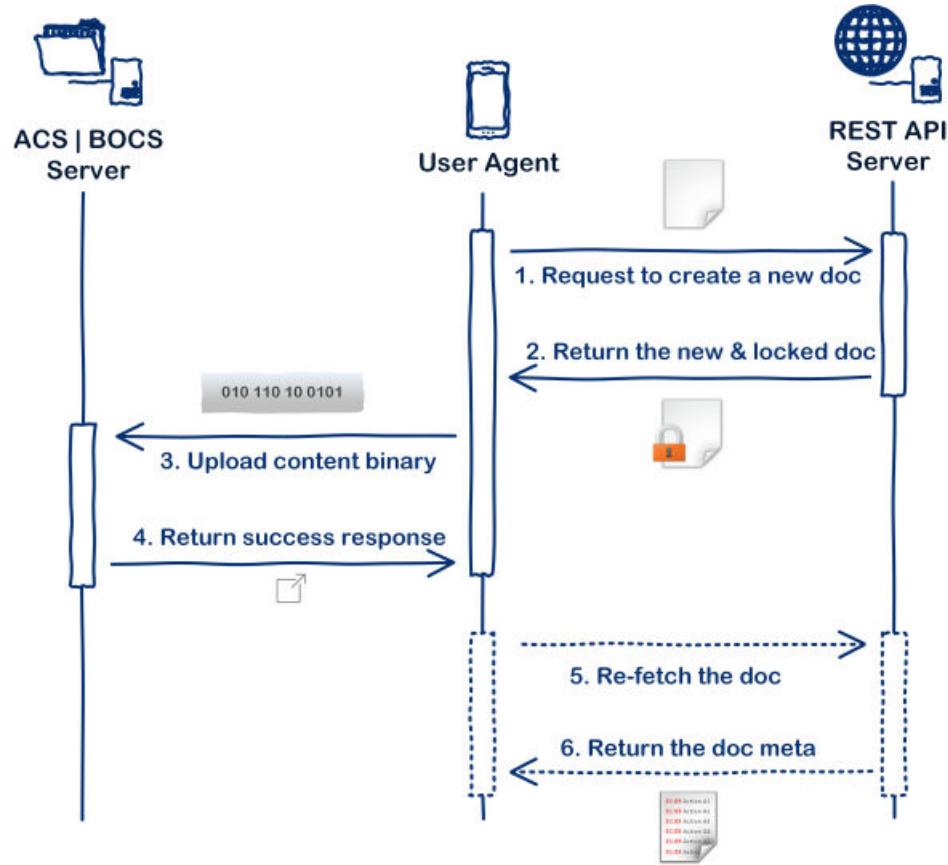
Parameter Name	Data Type	Default	Description
<i>require-dc-write</i>	boolean	false	Specifies whether to request a Distributed Content upload URL during the Sysobject creation or checkout. True returns the Accelerated Content Services or Branch Office Caching Services content upload URL in the Response link relations.
<i>format</i>	string	N/A	Required when <i>distributed-upload</i> is set to true. Content format for the new content to upload. For Folder Child Documents or Folder Child Objects, the existing <i>format</i> parameter is reused. But it does not support a multi-content pattern (comma-separated formats). The format must be an existing <i>dm_format</i> name.
<i>content-length</i>	long	N/A	Required when <i>distributed-upload</i> is set to true. Content length for the new content to upload. For Folder Child Documents or Folder Child Objects, the existing <i>content-length</i> parameter is reused. But it does not support a multi-content pattern (comma-separated lengths).

Parameter Name	Data Type	Default	Description
<i>network-location</i>	string	N/A	<p>Optional</p> <p>Network location for the new content to upload. The server takes the network location to determine which Branch Office Caching Services server to serve for the content upload. When the network location definition exists and has a mapping in a Branch Office Caching Services server configuration, the corresponding Branch Office Caching Services server content upload URL is returned; otherwise, the default Accelerated Content Services server content upload URL is returned.</p>

Parameter Name	Data Type	Default	Description
<i>require-ip-network-location</i>	boolean	false	<p>Optional</p> <p>From the 22.1 release, Foundation REST API client applications can choose to return Branch Office Caching Services URLs depending on the IP address. To use this feature, you must set the value of <i>require-ip-network-location</i> to true.</p> <ul style="list-style-type: none">• Scenario 1: If you set the value of <i>require-ip-network-location</i> to true and you do not provide the information for <i>network-location</i>, then Branch Office Caching Services URLs are returned for content upload for all the network location objects having the IP address range defined for each client IP address. However, if a network location is not found for a client IP address, then an Accelerated Content Services URL is returned.• Scenario 2: If you provide the information for <i>network-location</i> explicitly and in addition you set the value of <i>require-ip-network-location</i> to true, then Branch Office Caching Services URLs are returned for content upload for both the explicitly provided information for <i>network-location</i> and also for all the network locations that are valid for the client IP address range.• Scenario 3: If you do not provide any information for <i>network-location</i> and in addition you set the value of <i>require-ip-network-location</i> to false, then the Accelerated Content Services URL is returned for content upload.

Parameter Name	Data Type	Default	Description
<i>async</i>	boolean	false	<p>Optional</p> <p>Asynchronous mode for the Branch Office Caching Services content upload. The server takes this parameter to determine when to replicate the content binary from the Branch Office Caching Services server cache to the Documentum CM Server storage:</p> <ul style="list-style-type: none">• When <i>async=true</i>, the upload success Response is returned to the client immediately after the client side upload completes. The content replication between the Branch Office Caching Services server and Documentum CM Server is performed behind the scene asynchronously.• When <i>async=false</i>, the upload success Response is returned after the content on the Branch Office Caching Services server is completely replicated to the Documentum CM Server storage.

7.20.4 Workflow for Content Upload



Workflow for distributed upload in document creation

Figure 7-14: Workflow for distributed upload during Document creation

1. Foundation REST API client creates a new sysobject by making a POST request to Folder Child Documents resource or Folder Child Objects resource, in which it requires to get the Accelerated Content Services or Branch Office Caching Services content upload URL.

```
POST /dctm-rest/repositories/REPO/folders/0c00000c80000105?require-dc-
write=true&format=text
&content-length=13&network-location=Toronto&async=true HTTP/1.1
Host: localhost:8080
Content-Type: application/vnd.emc.documentum+json
Authorization: Basic ZG1hZG1pbjpwYXNzd29yZA==

{
    "properties": {
        "object_name": "demodoc.txt"
    }
}
```

2. The newly created sysobject resource is returned with the Accelerated Content Services or Branch Office Caching Services content upload URL in the link relations. Its server object is locked by the user.

```
Status code: 201 Created
Location: http://localhost:8080/dctm-rest/repositories/REPO/objects/0900000d10000983
Content-Type: application/vnd.emc.documentum+json; charset=UTF-8
{
  "properties":
  {
    "r_object_id": "0900000d10000983",
    "r_content_size": 0,
  },
  "links": [
    {
      "rel": "self",
      "href": "http://localhost:8080/dctm-rest/repositories/
REPO/objects/0900000d10000983"
    }
    , ...
    {
      "rel": "http://identifiers.emc.com/linkrel/distributed-upload",
      "title": "BOCS-Toronto",
      "href": "http://<BOCS-SERVER>:9080/ACS/servlet/BOCS?
command=write&XXX"
    }
  ]
}
```

3. Foundation REST API client makes another POST request on the content upload URL to complete the upload.

```
POST http://<BOCS-SERVER>:9080/ACS/servlet/BOCS?command=write&XXX HTTP/1.1
Content-Type: plain/text

Hello, world!
```

4. When the content upload is completed, Foundation REST API client gets a 302 Moved Temporarily Response from the Accelerated Content Services or Branch Office Caching Services server, where there is a Location Header telling the Sysobject resource URL. The server object is also unlocked upon the success of the content upload.

```
HTTP/1.1 302 Moved Temporarily
Location: http://localhost:8080/dctm-rest/repositories/REPO/objects/0900000d10000983
```

By this step, the Distributed Content upload has been completed.

5. (Optional) Foundation REST API client can make the URL redirect request to the sysobject resource again.
6. In the sysobject resource this time, content related metadata (i.e. *a_content_type,r_content_size*) are returned with correct values.

The following code sample explains the creation of contentless object by taking the contentless object URL. For example, `https://<host>:<port>/dctm-rest/repositories/<repository name>/folders/<folderID>/documents?require-dc-write=true&format=pdf&content-length=<content-length>&content-charset=UTF-8&async=false` and fetches the ACS URL and uploads the content by using the ACS URL.

```
/** 
 * Creates a new document and fetches the corresponding ACS upload URL.
```

```

    *
     * @param contentLessDocURL dctm-rest url for creating the document with metadata
     * and without content.
     * @return the ACS URL content upload
     * @throws ContentTransferException if there is a failure in document creation or
     * response parsing
    */
    static String createDocumentAndFetchAnACSURL (String contentLessDocURL) throws
ContentTransferException {
    int responseCode = 0;

    HttpURLConnection httpURLConnection =
URLConnection.getHttpURLConnection(contentLessDocURL, "POST");
    httpURLConnection.setRequestProperty("Content-Type", "application/json");
    httpURLConnection.setRequestProperty("Accept", "application/json");
    httpURLConnection.setDoOutput(true);
    String payLoad = getObjectPayLoad();
    try (OutputStream os = httpURLConnection.getOutputStream()) {
        os.write(payLoad.getBytes());
        os.flush();
        responseCode = httpURLConnection.getResponseCode();
    } catch (IOException exception) {
        ExceptionUtil.createWithAppropriateMessage(exception);
    }
    String acshref = null;

    try {
        InputStream is = responseCode == 201 ? httpURLConnection.getInputStream() :
httpURLConnection.getErrorStream();
        if (is == null) {
            throw new ContentTransferException(responseCode, "No response stream
available from server");
        }
        if (responseCode == 201) {
            JsonNode json = JsonReader.createJsonNode(is);
            httpURLConnection.disconnect();
            // reading the links
            JsonNode linksNode = null;
            if (JsonReader.isObjectIdFound(json)) {
                linksNode = json.has("links") ? json.get("links") : null;
                acshref = JsonReader.getUploadACSURL(linksNode);
            }
        } else {
            ErrorStreamReader.readErrorStream(is, responseCode);
        }
    } catch (IOException ioException){
        throw new ContentTransferException(responseCode,"Error while reading the
json output during doc create",ioException);
    }
    return acshref ;
}

static String getObjectPayLoad () {
    String jsonProps = null;
    try {
        // JSON metadata
        ObjectMapper mapper = new ObjectMapper();
        Map<String, Object> props = new HashMap<>();
        props.put("object_name", "rest-api-test-file");
        props.put("subject", "sample subject");
        props.put("title", "upload-title");
        props.put("authors", new String[]{"author1", "author2"});

        Map<String, Object> payload = new HashMap<>();
        payload.put("properties", props);
        jsonProps = mapper.writeValueAsString(payload);
    } catch (JsonProcessingException jsonProcessingException) {
        jsonProcessingException.printStackTrace();
    }
}

```

```

        return jsonProps;
    }

    /**
     * Uploads content to ACS and retrieves the redirection URL upon success.
     *
     * @param acshref the ACS URL where content needs to be uploaded
     * @param filePath the local file path to upload its content
     * @return the redirection URL returned by ACS if the upload succeeds
     * @throws ContentTransferException if an I/O or HTTP error occurs during upload
     */
    static String uploadContent(String acshref, String filePath) throws
ContentTransferException {

    String redirectURL = null;
    File file = new File(filePath);

    if (!file.exists()) {
        System.err.println("File not found: " + filePath);
        return null;
    }

    // Create connection
    System.out.println("Uploading the content has started ..... ");
    HttpURLConnection conn = URLConnection.getHttpURLConnection(acshref,"POST");
    conn.setDoInput(true);

    // Set headers
    conn.setRequestProperty("User-Agent", "client");
    conn.setRequestProperty("Content-Type", "application/octet-stream");
    conn.setRequestProperty("Accept", "*/*");
    conn.setInstanceFollowRedirects(false);
    int responseCode = 0;
    try (OutputStream out = conn.getOutputStream();
         FileInputStream fileInputStream = new FileInputStream(file)) {

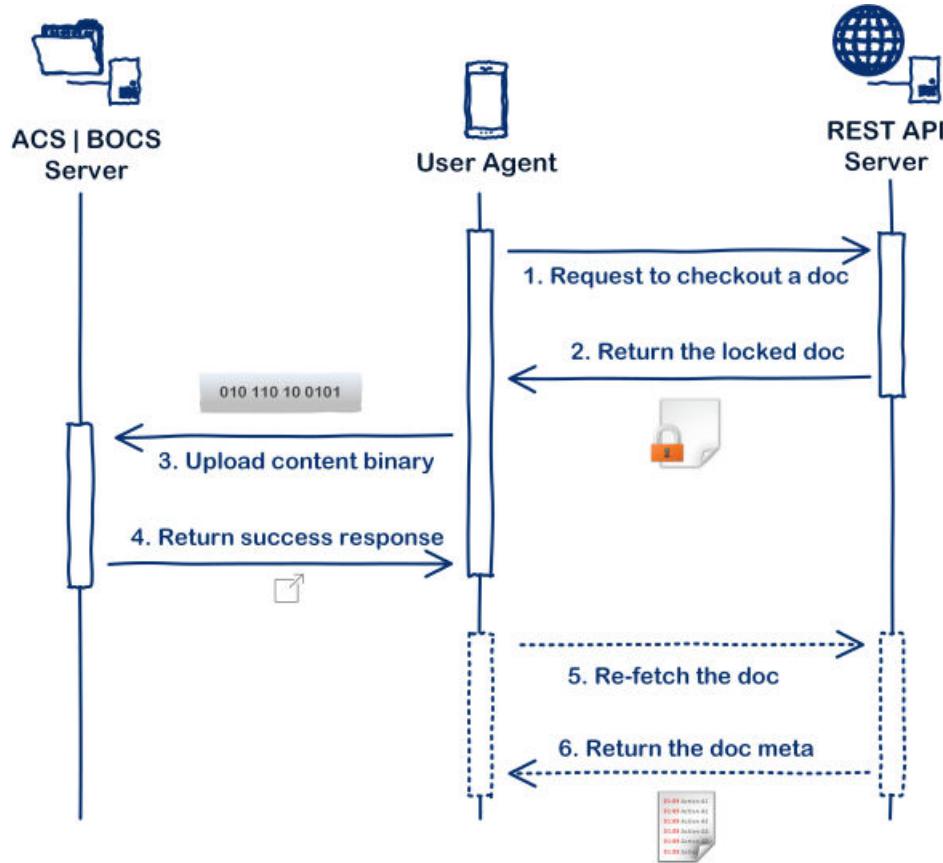
        byte[] buffer = new byte[1024*1024];
        int bytesRead;
        while ((bytesRead = fileInputStream.read(buffer)) != -1) {
            out.write(buffer, 0, bytesRead);
        }
        out.flush();
        // Handle response
        responseCode = conn.getResponseCode();
    } catch (IOException exception) {
        ExceptionUtil.createWithAppropriateMessage(exception);
    }

    // Redirection url received from ACS
    if (responseCode == HttpURLConnection.HTTP_MOVED_TEMP ) {

        for (int i = 1; ; i++) {
            String headerKey = conn.getHeaderFieldKey(i);
            String headerVal = conn.getHeaderField(i);
            if (headerKey == null && headerVal == null) break;
            if (headerKey.equals("Location")) {
                redirectURL = headerVal;
            }
        }
    } else {
        ErrorStreamReader.readErrorStream(conn.getErrorStream(),responseCode);
    }
    conn.disconnect();
    return redirectURL;
}

```

7.20.5 Workflow for distributed upload in document checkout



Workflow for distributed upload in document checkout

Figure 7-15: Workflow for distributed upload during Document checkout

1. Foundation REST API client creates a new sysobject by making a PUT request to Lock resource, in which it requires to get the Accelerated Content Services or Branch Office Caching Services content upload URL.

```
PUT /dctm-rest/repositories/REPO/objects/0900000d1000983/lock?require-dc-
write=true&
format=text&content-length=13&network-location=Toronto&async=true HTTP/1.1
Host: localhost:8080
Authorization: Basic ZG1hZG1pbjpwYXNzd29yZA==
```

2. The checked out sysobject resource is returned with the Accelerated Content Services or Branch Office Caching Services content upload URL in the link relations.

```
Status code: 200 OK
Content-Type: application/vnd.emc.documentum+json; charset=UTF-8
{
  "properties":
  {
```

```

    "r_object_id": "0900000d10000983",
    "r_lock_owner": "dmadmin",
    "r_content_size": 0,
},
"links": [
{
    "rel": "self",
    "href": "http://localhost:8080/dctm-rest/repositories/REPO/objects/
        0900000d10000983/lock"
}
,
...
{
    "rel": "http://identifiers.emc.com/linkrel/distributed-upload",
    "title": "BOCS-Toronto",
    "href": "http://<BOCS-SERVER>:9080/ACS/servlet/BOCS?
        command=write&XXX"
}
]
}

```

3. Foundation REST API client makes another POST request on the content upload URL to complete the upload.

```

POST http://<BOCS-SERVER>:9080/ACS/servlet/BOCS?command=write&XXX HTTP/1.1
Content-Type: plain/text

Hello, world!

```

4. When the content upload is completed, Foundation REST API client gets a 302 Moved Temporarily Response from the Accelerated Content Services or Branch Office Caching Services server, where there is a Location Header telling the Sysobject resource URL. The server object is also unlocked upon the success of the content upload.

```

HTTP/1.1 302 Moved Temporarily
Location: http://localhost:8080/dctm-rest/repositories/REPO/objects/0900000d10000983

```

By this step, the Distributed Content upload has been completed.

5. Optionally, Foundation REST API client can make the URL redirect request to the sysobject resource again.
6. In the sysobject resource this time, content related metadata (that is, a_content_type, r_content_size) are returned with correct values.

7.21 Conditional Request for Foundation REST API server

Foundation REST API server solves these two scenarios as follows:

- Support client cache: At times, the Foundation REST API server does not return the requested resource to client endpoint again if the identical requested resource is cached locally and not updated during the last acquisition. Otherwise, the Foundation REST API server handles the client's request as normal. With conditional request mechanism, it makes sense to keep client's local cache not stale.
- Avoid lost update: In this scenario, multiple clients get an identical resource simultaneously from Foundation REST API server. So, when one of the client

updates the resource successfully, the subsequent updates from others intended for this resource would fail. With conditional request mechanism, one client, without any acknowledgement that the resource has been modified, would not override others' update, avoiding lost update. For more information about conditional Requests, see RFC 7232.

7.21.1 Conditional Get Request for a single resource

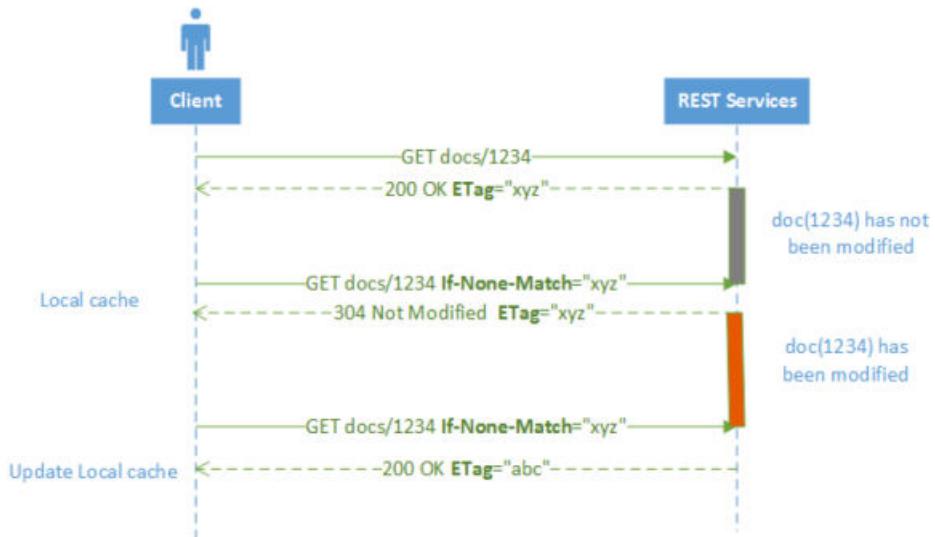


Figure 7-16: Conditional Get Request for a single resource

1. Foundation REST API server returns the requested resource with an ETag in the Response for a GET request.
2. Client specifies the HTTP Header, If-None-Match, in a GET request with the previous obtained ETag value.
3. With the Request, the Foundation REST API server determines whether the ETag value in the *If-None-Match* Header matches the current ETag of the instance of the requested resource.
 - When it does not match: 304 (Not Modified) status code is returned without sending back the resource.
 - Matched: Send back the resource as normal with a new value in the ETag Header.



Note: REST APIs such as the GET operation on content and contentMedia uses or returns only content-media.

7.21.2 Conditional Update Request for a single resource

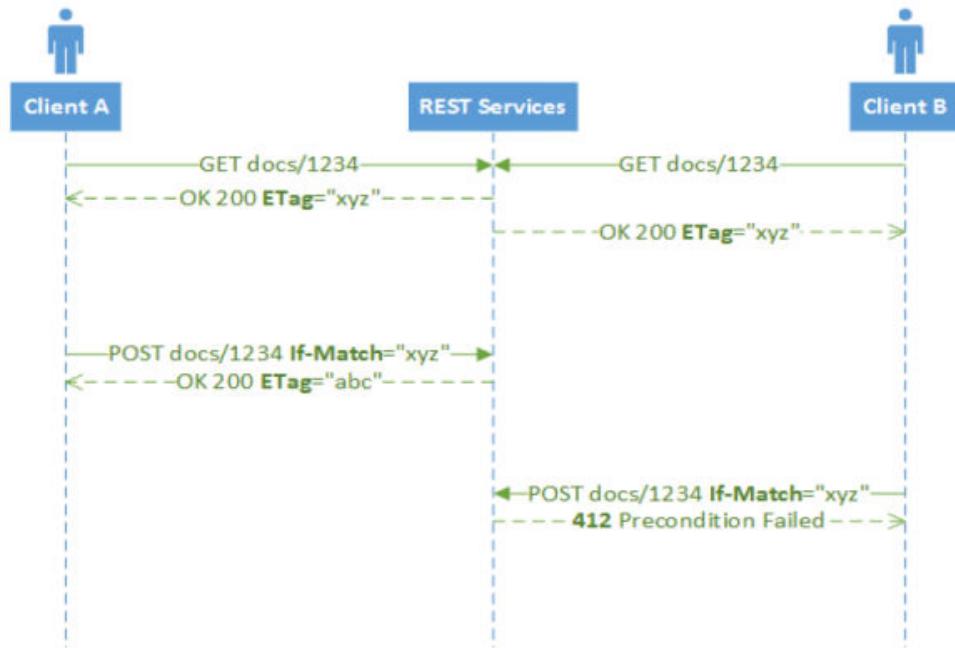


Figure 7-17: Conditional Update Request for a single resource

1. Multiple clients. Let's consider client A and B, get the identical resource with the same ETag value in respective responses at the same time.
2. Client A updates the resource successfully because the ETag in the `If-Match` HTTP Header of the Request matches. The Foundation REST API server generates a new ETag for the updated instance of the resource and then client A gets this updated ETag value.
3. Now the Foundation REST API server receives an update Request from client B with the previously obtained ETag value specified in the `If-Match` Header. The Foundation REST API server determines that this ETag value is not matched, therefore it fails to update the resource and returns status code 412 (`PreCondition Failed`).

7.21.3 Customize resources with conditional Request

To support a conditional Request for a customized resource, the Foundation REST API publishes the `@EnableConditionalRequest` annotation and the `Digestible` interface, which allows you to extend these resources.

- `EnableConditionalRequest.java`

```
/*
 * Annotation that annotates a controller. Once annotated by
@EnableConditionalRequest,
 * the controller would handle a HTTP request, mainly GET, PUT or POST, as
conditional
 * request.
 *
 * @see com.emc.documentum.rest.context.ConditionalRequestBodyAdvice
 * @see com.emc.documentum.rest.context.ConditionalResponseBodyAdvice
 */
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface EnableConditionalRequest { }
```

- `Digestible.java`

```
/*
 * Allows a persistent object to calculate its own digest. In the meanwhile, Enables
to
*update a persistent object conditionally based on if the {@code request digest}is
matched
* with another digest, which is commonly* the current digest of the identical
persistent*
* object with newest version.
* <p>
* {@code request digest}, is obtained from a request with certain mechanism and
stored
* temporarily in an object before real update. By comparing with current digest of
* corresponding persistent object with newest version, conditional update comes
* into realization.
*/
public interface Digestible {
    /**
     * Calculates digest for a persistent object itself.
     * <p>It's up to the persistent model which implements this interface to
determine how
     * to calculate
     * a distinct digest for a model instance, i.e., making the digest based on
distinct pro    * perties.</p>
     *
     * @return the digest
     */
    String calculatesDigest();

    /**
     * Sets the Request digest.
     *
     * @param reqDigest
     */
    void setReqDigest(String reqDigest);

    /**
     * Get the Request digest.
     *
     * @return the Request digest
     */
    String getReqDigest();

    /**
```

```

        * Determines whether the Request digest is matched with another digest
instance.
        *
        * @param digest another digest
        * @return TRUE, if the Request digest is matched with the input, otherwise FALSE
        */
boolean isMatchedWithReqDigest(String digest);
}

```

7.21.4 Examples

Assuming a Model View Controller approach:

- Implement the Digestible interface in the Model:

```

AliasSet.java

@SerializableType(value = "alias-set", xmlNSPrefix = "dm",
xmlNS = "http://identifiers.emc.com/vocab/documentum")
public class AliasSet implements Digestible{
    public AliasSet() {
        setType("dm_alias_set");
    }
    public List <String> getMandatoryFields() {
        return Arrays.asList("r_object_id",
                            "object_name",
                            "owner_name",
                            "i_vstamp");
    }
    @Override
    public String calculatesDigest() {
        return ReqDigestHelper.generateDigest(Arrays.asList(getId(),
            String.valueOf(getMandatoryAttribute("i_vstamp"))));
    }
    @Override
    public void setReqDigest(String reqDigest) {
        this.reqDigest = reqDigest;
    }
    @Override
    public String getReqDigest() {
        return reqDigest;
    }
    @Override
    public boolean isMatchedWithReqDigest(String digest) {
        if ("*".equals(reqDigest)) {
            return true;
        }
        if (reqDigest != null) {
            return reqDigest.equals(digest);
        }
        return false;
    }
}

```

- Enable a conditional request in the controller:

```

AliasSetController
/**
 * An alias set.
 */
@Controller("acme#alias-set")
@RequestMapping("/repositories/{repositoryName}/alias-sets/{aliasSetId}")
@ResourceViewBinding(value = AliasSetView.class, queryTypes = "dm_alias_set")
@EnableConditionalRequest
public class AliasSetController extends AbstractController {
    ...
}

```

7.22 Using the AppWorks Gateway to integrate the Foundation REST API

This section discusses Foundation REST API integration using the OpenText AppWorks Gateway. The AppWorks Gateway is used for hybrid application rollouts and upgrades, push notification services, lightweight mobile devices management, REST API access, user authorization, and more. The AppWorks Gateway also performs functions that allow you to access an OpenText EIM server platform. This access gives you the ability to manage desktop and mobile client communications.

You can use the Documentum REST API, augmented by the features of the AppWorks Gateway, to perform the following tasks.

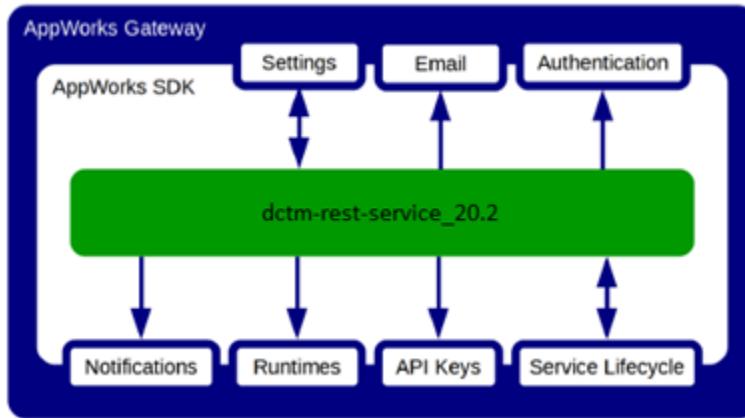
You can:

- Use the extensibility features of Foundation REST API to customize the REST API and use the AppWorks SDK to access your OpenText server platforms.
- Create a custom deployment that can be used to combine the resources of various products. For example, SOAP web services combined with REST APIs (and possibly other products), can provide a consolidated API as a Facade.
- Use the Foundation REST API to develop apps that have mobile or desktop user interfaces with heightened capabilities.
- Simplify your app deployments, upgrades, and rollback procedures.
- Centralize your app's configuration and management settings.

7.22.1 Design Overview

The discussions in this section apply to the **AppWorks Gateway 16.7.0** and later Software Development Kits (SDK). Any design description is limited to the task of integrating the Documentum REST API with the AppWorks Gateway as a service.

The hosted Foundation REST API has been specifically packaged to allow hosting by the OpenText AppWorks Gateway. The following figure illustrates the interactions between integration components.



7.22.2 AppWorks Component Classes

To enable Document REST API deployment in AppWorks Gateway, any required packaging modifications have been made. The deployable AppWorks artifact; `dctm-rest-appworks-service_<release version>.zip`, can be downloaded from the download portal. If you want to modify the `dfc.properties` and `rest-api-runtime.properties` files, this must be done prior to deployment to the AppWorks Gateway.

 **Note:** To get shorter URLs after deployment, you can rename the artifact to `dctm-rest_<release version>.zip` before you deploy it.

The following classes have been added to the REST API SDK to support the consumption of AppWorks SDK in custom controllers. These controllers extend from a default REST API provided controller class called `AWSdkControllerBase`.

- `AwSdkBean`

A Java bean that is included in the controller classes to provide access to AppWorks SDK clients and provide the utility functions required to send emails, notifications, and so on.

- `AwSdkControllerBase`

A reference implementation of a controller that you can extend to handle URLs. It is auto wired to provide the `AwSdkBean` Java bean.

7.22.3 AppWorks authorization

AppWorks Gateway provides OTDS based authentication using the AuthClient Class and `http://<GatewayHost>:8080/v3/admin/auth` URL endpoint. AuthClient has been made available through the `AwSdkBean` Java bean. This allows the custom controller code to authorize with AppWorks Gateway. “[SDK interfaces](#)” on page 309 provides more details.

DCTM-REST service implements AppWorks AuthRequestHandler interface to enable delegation of AppWorks Gateway authentication to Foundation REST API server. The Foundation REST API server forwards auth requests to Documentum CM Server which authenticates using the OTDS server. This enables handling of all AppWorks Gateway username/password authentications by DCTM-REST service (via Documentum CM Server OTDS integration). This unification of AppWorks Gateway and OpenText Documentum CM users enables AppWorks Gateway users to access REST resources. Ensure that the AppWorks Gateway resource and OpenText Documentum CM resource are synchronized in the OTDS server prior to selecting REST as primary auth provider. *OpenText Directory Services Development Guide* provides more details about resource synchronization.

To enable Foundation REST API as primary authentication provider in AppWorks Gateway, select **dctm-rest** as primary provider.

In AppWorks Gateway Admin UI, select **Configuration > AuthProviders**.

Here is the new addition to `rest-api-runtime.properties` property configuration file:

```
rest.appworks.otds.repository=repo_name
```

The repository that is integrated with OTDS to perform OTDS password authentication. This should be the same OTDS with AppWorks Gateway resource partition and users. The default value is empty.

7.22.4 AppWorks Settings Management

Foundation REST API deployed on AppWorks Gateway may need (frequently modified) configuration parameters to be configurable by end users. Common uses for settings include:

- Foundation Java API properties changes such as docbroker host, port, and so on
- Enabling verbose REST Inbound/Outbound HTTP requests
- Change REST log level to DEBUG, TRACE, and so on

You can edit these settings from the AppWorks Gateway's UI after you deploy the DCTM-REST service in the Gateway server. This enables you to edit the properties using the GUI and to avoid modifying the properties files manually. To access this feature, login to AppWorks Gateway Admin UI, select **Services**, select the **gear** icon under Foundation REST API and select the **Settings** tab.

A service restarted is needed to see the effect of the changes made. The following commonly used properties are exposed using the UI:

- Foundation Java API: Add message identifier in the exception message (dfc.exception.include_id)
- Foundation Java API: Add extra embellishments to the exception message (dfc.exception.include_decoration)
- Foundation Java API: Determines TCP/IP socket type. Valid values: native, try_native_first, secure, try_secure_first (dfc.session.secure_connect_default)
- Foundation Java API: Global registry repository name (dfc.globalregistry.repository)
- Foundation Java API: Global registry repository password (dfc.globalregistry.password)
- Foundation Java API: Global registry repository user (dfc.globalregistry.username)
- Foundation Java API: The hostname or IP address of the dockroker (dfc.docbroker.host[0])
- Foundation Java API: The port number of the docbroker (dfc.docbroker.port[0])
- LOG4J2: Specify the logging level for REST Service (logger.package.level)
- REST: Enable the REST request and response message logging at the server side (rest.message.logging.enabled)
- REST: Specifies the logging buffer size in byte for requests and responses. The default value is 1048567. (rest.message.logging.buffer)

7.22.5 SDK interfaces

The AwSdkBean instance is available to all controllers extending from the AwSdkControllerBase class, which includes the following API methods.

```
/**
 * @return Instance of Gateway's auth mechanism
 */
AuthClient getAuthClient()

/**
 * @return Instance of the Runtimes registered with the Gateway
 */
RuntimesClient getKnownRuntimes()

/**
 * @return Retrieve trusted provider (API) that wants to use a limited subset of the
 * Gateways APIs
 */
TrustedProviderClient getTrusterProviders()

/**
 * @return Instance of Gateway's settings client to create, update, and be notified of
 * Gateway wide parameters
 */
SettingsClient getSettingsClient()

/**
```

```
* @return MailClient to enable sending emails
*/
MailClient getMailClient()

/**
* @return NotificationClient to enable sending notifications to users or apps
*/
NotificationsClient getNotificationsClient()

/**
* @return Instance of Gateway's service client to create a complete deployment and
register connectors
*/
ServiceClient getServiceClient()
```

Refer to the sample implementation found in the DCTM REST <release version> SDK. The sample project is located in the .../samples/resource-appworks-samples folder.

7.22.6 Resources

OpenText AppWorks Developer (<https://developer.opentext.com/awd/intro>).

7.23 Checksum for Foundation REST APIs

Foundation REST API provides the checksum to validate the integrity of all the synchronous and asynchronous content transfer operations.

Synchronous content transfer operation

Checksum can be requested for the import and export synchronous content transfer operations performed using:

- Foundation REST API
- Foundation REST API and Accelerated Content Services
- Foundation REST API and Branch Office Caching Services

Asynchronous content transfer operations

Checksum can be requested for the import and export asynchronous content transfer operations using REST Services and Branch Office Caching Services or when you know the object ID.

In asynchronous operations to fetch the checksum, when an object ID is known, clients can retrieve the checksum using the following checksum endpoint:

repositories/{{repository-name}}/objects/{{object-id}}/checksum

Use one of the following methods to request the checksum from the appropriate Foundation REST API endpoints:

- At runtime, append the require-checksum=true query parameter to the Foundation REST API endpoint requests.
- Set the rest.checksum.enabled property in the rest-api-runtime.properties file to true. The default value is false.



Note: Checksum requests at runtime using the `require-checksum` query parameter takes precedence over the value provided for the `rest.checksum.enabled` property.

Foundation REST API sends the checksum as part of the `Content-Digest` response header. When multiple contents are part of the import or export operations, including the renditions of primary content, the checksum values for the import or export operations are represented as comma-separated values in the `Content-Digest` response header, following the order of imported contents.

Clients that use Foundation REST API can request the checksum for the content transfer operations with the following Foundation REST API endpoints:

- Folder child documents
- Folder child objects
- Content
- Contents
- Content media
- Versions
- Lock
- Checksum



Note: The link relations reference to the checksum API with the `link rel checksum` is available as part of the link relations for the preceding list of REST resources when the value of the `require-checksum` query parameter is set to `true`.

For more information about the checksum API, see *OpenText Documentum Content Management - Foundation REST API Reference Guide (EDCPKRST250400-PRE)*.

Chapter 8

Authentication extensibility

You can use authentication extensibility to intercept the authentication flow or to create a custom authentication scheme for Foundation REST API.

8.1 Anonymous access

Anonymous Access is an extension based on Foundation REST API security framework. This extension can be used to configure anonymous access for custom resources or static resource files in your custom REST services. There are two user scenarios for this feature:

1. Add new static resource files into your REST application. An example of this is adding custom CSS style sheets or JavaScript files.
 - When a static resource file URL is not configured with authentication or anonymous access, that Request is rejected with a 403 status code.
 - Anonymous configuration provides users with a way to bypass the authentication process.
2. Add custom anonymous REST resources into a custom REST service. An example of this is a resource that contains a built-in login user.

Your particular REST service may have a built-in login user on the server side and you want to skip the usual authentication schema that is already configured.

8.1.1 Implementation

You must create a custom Java controller that uses the `@AnonymousAccess` annotation to get the functionality that you want. There are two ways to set anonymous access in your custom REST service:

1. By Java annotation using the `@AnonymousAccess` annotation.
2. By configuration using the `rest-api-runtime.properties` file.



Note: The two approaches shown here do not need to be implemented at the same time for one resource. They can take effect individually.

8.1.1.1 By Java annotation

In the Spring framework, you can connect add-on features to resource controllers as annotations, which allows you to do many things using Spring itself while using Reflection to reduce the amount of work you ultimately have to do.

Typically, all REST service functionality is implemented into one controller that uses the `@AnonymousAccess` annotation. Your controller can provide anonymous access functionality to your REST application. The `@AnonymousAccess` annotation allows:

- All resource controllers to be excluded from a regular configured authentication schema.
- All resource controllers to be provided with their own repository security context, initialization, and cleanup procedures.

`@AnonymousAccess` contains two fields:

- `contextInitializerClass`

This is an implementation of the `contextInitializerClass` interface. This interface defines the customized details for your context, such as username and password.

- `contextCleanerClass`

This field extends from `DefaultRepositoryContextCleaner`, and is used to clean up any security related context.

➡ Example 8-1: Typical usage scenario

```
@AnonymousAccess(  
    contextInitializerClass = MyRepositoryContextInitializer.class,  
    contextCleanerClass = DefaultRepositoryContextCleaner.class  
)  
@Controller("acme#custom-resource")  
public class MyAnonymousResourceController extends AbstractController {...}
```



The `initialize()` and `clean()` methods can be implemented as needed:

- For example, you can use the `initialize()` method during initialization to set the repository name, login name and password in `RepositoryContextHolder`. The `authType` can be set to `<AuthType.PASSWORD>`, which indicates that user authentication is done using a login name and password.
- The default `clean()` method is used to clean the context in `RepositoryContextHolder`. When you have more complex logic for cleaning context, you can override this method with your own custom code to perform the cleaning.

8.1.1.2 By runtime configuration

Runtime configuration is not recommended for the implementation of anonymous REST services, but it is the only way to configure anonymous static resource files.

A runtime property called `<rest.security.anonymous.url.patterns=>` is added to the `rest-api-runtime.properties` file. This property value must be written using `<Ant>` style URL patterns. Multiple URL patterns can be created as comma-separated values.

 **Example 8-2: Configuration in rest-api-runtime.properties**

```
rest.security.anonymous.url.patterns=/another-anonymous-resource,
/one-more-resource
```



The patterns defined in it are added to any other anonymous patterns.

8.1.2 Extension samples

There are samples named `<documentum-rest-anonymous-samples>` available in the `<documentum-rest-anonymous-samples>` package. These samples illustrate how to use this security extension, and both approaches are demonstrated in them. To get these samples, see the `README.md` file in the package and follow the instructions provided in it.

8.2 Generic servlet filter customization

In Foundation REST API the `<SpringSecurityFilterChain>` is created by Java code. Your REST applications may want to inject more HTTP servlet filters into your REST application to handle the Requests with additional business logic, such as counting Requests, logging specific Requests, and more.

In Foundation REST API 7.3 and later, an XML configuration file is exposed to allow you to inject custom (non-security) filters before or after `<SpringSecurityFilterChain>` is created.



Note: Modifying the `web.xml` directly in the WAR file is not recommended because the filter sequence cannot be guaranteed.

Here's a code sample that shows you how to edit the `<dctm-rest.war>\WEB-INF\classes\META-INF\spring\extension\rest-api-custom-filters.xml` file for customizing servlet filters:

 **Example 8-3: /META-INF/spring/extension/rest-api-custom-filters.xml**

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.1.xsd">
```

```

<!-- allow users to add custom filters into the REST servlet; all filters are
     mapped to the root context '*' -->
<bean id="restFilterFactory" class="com.emc.documentum.rest.filter.FilterFactory">
    <property name="customBeforeSecurityFilters">
        <list>
            <!-- an ordered list of filters added before springSecurityFilterChain;
                 empty by default -->
        </list>
    </property>
    <property name="customAfterSecurityFilters">
        <list>
            <!-- an ordered list of filters added after springSecurityFilterChain;
                 empty by default -->
        </list>
    </property>
</bean>
</beans>

```



In a custom REST application, you can inject filters by updating the `rest-api-custom-filters.xml` file. The REST SDK provides a sample that shows you how to do this.

➡ Example 8-4: Sample `rest-api-custom-filters.xml`

Here's a code sample that illustrates a custom `rest-api-custom-filters.xml` file:

```

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.1.xsd">

    <!-- allow users to add custom filters into the REST servlet; all filters are
         mapped to the root context '*' -->
    <bean id="restFilterFactory" class="com.emc.documentum.rest.filter.FilterFactory">
        <property name="customBeforeSecurityFilters">
            <list>
                <!-- an ordered list of filters added before springSecurityFilterChain;
                     empty by default -->
            </list>
        </property>
        <property name="customAfterSecurityFilters">
            <list>
                <!-- an ordered list of filters added after springSecurityFilterChain;
                     empty by default -->
                <bean class="com.emc.documentum.rest.sample.filter.RequestCountingFilter"/>
            </list>
        </property>
    </bean>
</beans>

```



➡ Example 8-5: Entire filter chain sequence

Here is the entire filter chain sequence, ordered as the following:

```

MessageLoggingFilter
↓
CharacterEncodingFilter

```

```
↓  
RepositoryNamingFilter  
↓  
<customBeforeSecurityFilters ..>  
↓  
SpringSecurityFilterChain  
↓  
<customAfterSecurityFilters ..>  
↓  
HiddenHttpMethodFilter  
↓  
ApplicationFilter
```



8.3 Using Foundation REST API as the Authentication Handler for the AppWorks Gateway

This section describes using Foundation REST API as the authentication handler for the AppWorks Gateway.

8.3.1 Overview

Making Foundation REST API the primary authentication handler is useful when there are several Foundation REST API resources, that are able to authenticate requests, deployed on the AppWorks gateway. In this case, requests are authenticated with the OpenText Directory Service (OTDS) using Documentum CM Server.

Making Foundation REST API the primary authentication handler has the following advantages:

- You can eliminate the need for any user synchronization between AppWorks and Documentum CM Server on OTDS. This is because Foundation REST API that are deployed on AppWorks no longer depend on the AppWorks Gateway for authentication. Among other advantages, you can avoid searching for Gateway users that may not be present in Documentum CM Server.
- You can combine multiple existing Foundation REST API resources while using Documentum CM Server as a single resource for all authentication requests.
- Upon a successful login, Foundation REST API can obtain a user session from Documentum CM Server.

8.3.2 Some Important Information

This information is meant for users of AppWorks Platform 20.2, or later. The descriptions in this section are limited to the integration of Foundation REST API in AppWorks Gateway as a primary authentication handler that uses a username and password combination to authenticate requests that are forwarded to Documentum CM Server for validation.

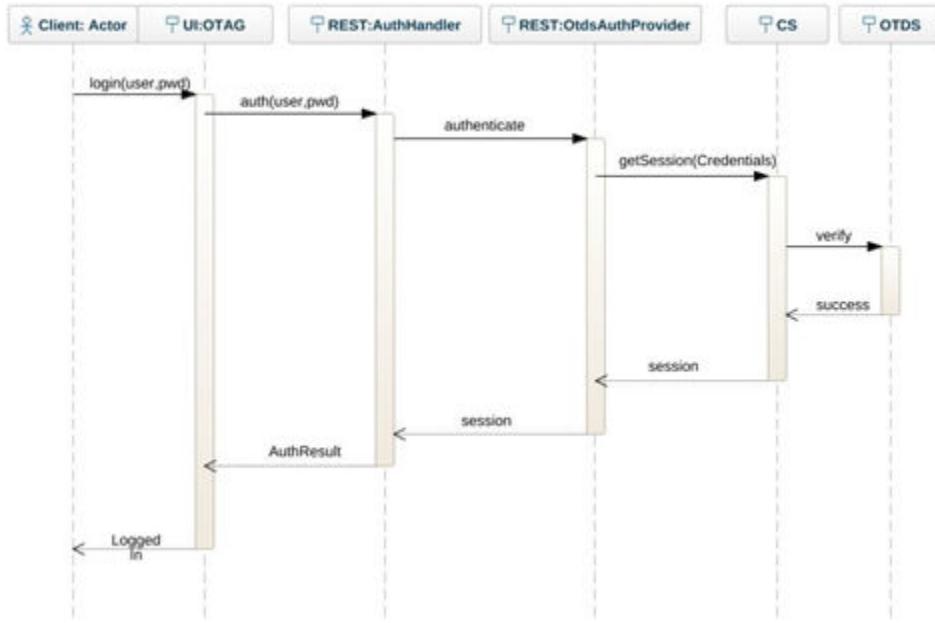
The following REST authentication schemes are supported:

- *basic*: Supports HTTP basic authentication for Documentum CM Server inline users.
- *otds_password*: Supports using a password for OTDS integration with Documentum CM Server.
- *otds_token*: Supports using a token for OTDS integration with Documentum CM Server.

8.3.3 Using a Hosted Document REST API Service

The hosted Foundation REST API gets registered as the primary authentication handler. This allows all authentication requests that arrive at the AppWorks Gateway to be forwarded to Foundation REST API for validation.

The *AwAuthRequestHandler* class is the main REST handler class that processes authentication requests and forwards them to the REST authentication layer for validation. When validation succeeds, a success *AuthHandlerResult* instance with user information is returned. This interaction is shown in the following sequence diagram:



8.4 Custom authentication development

This section discusses how to develop a custom authentication scheme in Foundation REST API 7.3 and later. Foundation REST API provides you with a rich set of authentication schemes. However, you may want to add your own authentication schemes, such as *OAuth*, *OpenID*, *Two-factor authentication*, and others, that are not supported out-of-the-box.

There are various requirements for custom authentication deployment. Therefore, to accommodate many of your specific requirements, we have exposed the authentication extension development framework, which allows you to develop a custom authentication scheme according to your specific needs.

From Spring Security 6.5 release onwards, `SecurityFilterChain` must have unique `RequestMatcher` definitions. If there are more than one `SecurityFilterChain` classes annotated with `EnableWebSecurity` that overrides the default `WebSecurity` configuration that creates a conflict while resolving the `RequestMatcher` uniquely. The REST client applications must exclude Foundation REST API default `WebSecurity` class to avoid conflicts with default `WebSecurity` configurations.

To support the conditional exclusion, configure a class that implements `org.springframework.context.annotation.Condition` to the `rest.security.auth.override.condition.class` property in the Foundation REST API runtime properties file. This condition is evaluated for each `@EnableWebSecurity` annotated class to decide whether it should be included.

8.4.1 Understanding the security filter flows

The REST authentication framework leverages the security found in the Spring framework to setup the servlet security context.

For more information, see “[Generic servlet filter customization](#)” on page 315 in Foundation REST API to understand how the Spring Security Filter Chain takes place within the entire servlet filter chain.

Specific to the security filter chain itself, the Spring Security framework puts multiple HTTP security configurations into a chain called `<SpringSecurityFilterChain>`. The high level flow of the security filter chain in Core REST is as shown:

Anonymous access URL interceptors



*Core or Custom Authentication Filter (s) for URL pattern /repositories/***



Access denial security filter

For any request that is sent to the Foundation REST API server, the following applies:

- When the Request URL maps to an anonymous access URL pattern, the Request is bypassed by the resource controller without security handling. Some static resources, such as js and css files, follow this pattern. The Core REST authentication framework exposes an extension for you to configure anonymous access resources or URL patterns.

For more information, see “[Anonymous access](#)” on page 313

- The out-of-the-box authentication schemes or custom authentications manage the Request authentication for the resource URL pattern `/repositories/**` by default.

This URL pattern can be configured globally in the `rest-api-runtime.properties` file using the `<rest.security.auth.root.url>` property.

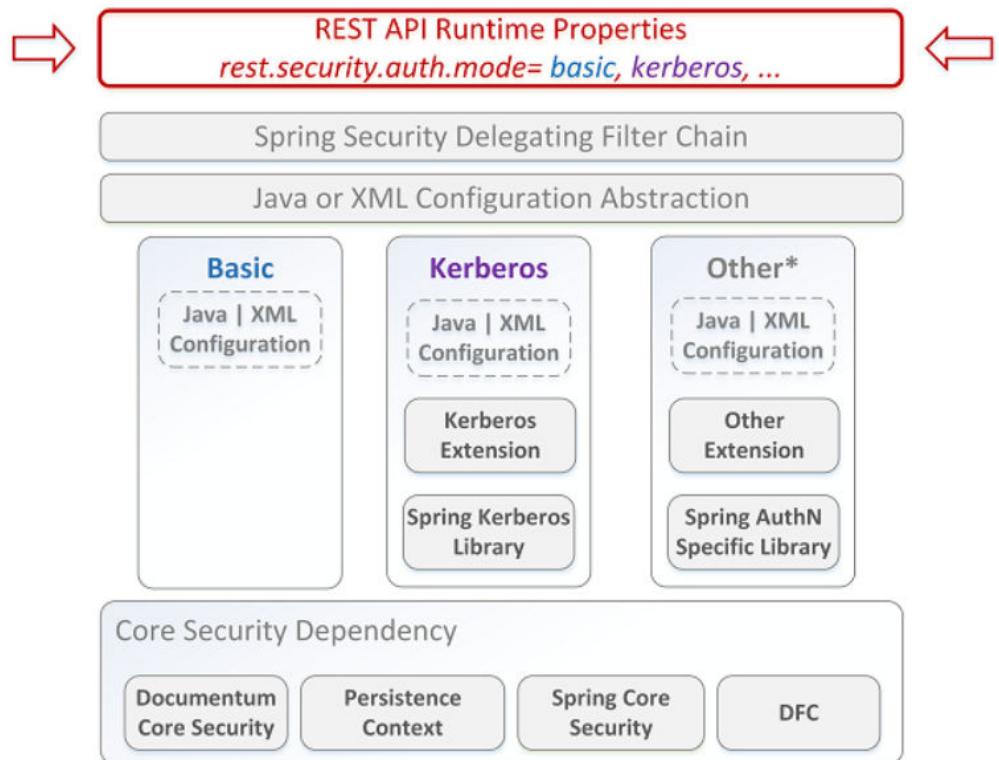
- For other cases not shown in preceding lists, the Request is rejected with a `<403>` status code.

8.4.2 Understanding the authentication framework

As a Core REST custom authentication developer, it is assumed that you are familiar with the following:

- Spring Security (supported versions)
- Maven
- Documentum REST Services Extensibility

The Foundation REST API authentication framework has a layered and extensible architecture, as illustrated in the following diagram. An individual authentication scheme in Foundation REST API is taken as a configuration of Spring HTTP Security:



- Configuration parameters, such as. `<rest.security.auth.mode>`, in the REST `rest-api-runtime.properties` file provides the uniform interface for you to choose which authentication scheme to use at runtime.
- Following the runtime configuration, Spring Security loads the corresponding configurations from the class-path.
- More specifically, Spring security looks for an effective Java or XML configuration to load a configured authentication scheme.

- There may be multiple authentication scheme implementations deployed to one server, but only the one that is specified in the runtime properties is used.
- Each authentication scheme has its own Java or XML configuration, its own implementation, and dependent libraries.
- Foundation REST API provides a common set of security libraries to facilitate the development of authentication, such as client token integration, session manager utility, and more.

The authentication flow for a Foundation REST API authentication scheme is similar to any typical Spring security authentication flow. The following diagram shows the flow of a typical authentication:

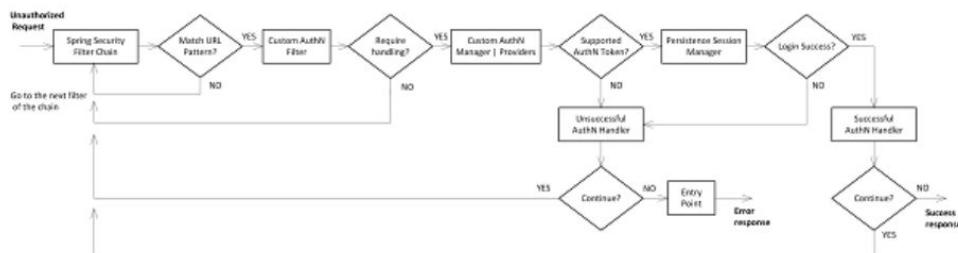


Diagram-2 A Typical Authentication Flow

- When an unauthorized Request arrives, Spring Security examines the Request and finds a matching security filter to handle the Request.
- Within the authentication filter, there is the logic to determine whether or not to process the Request. This is usually determined by examining the Request information and the local security context.
- When the authentication filter determines that it should authenticate the Request, it extracts the Request information into a request token, such as an Authorization Header, and calls its wired authentication manager to handle that Request.
- Each authentication manager may have more than one authentication provider. Each provider calls the persistence authentication manager to authenticate the Request token.
- The persistence authentication manager calls the Foundation Java API session manager to authenticate the user in the Request. For OpenText Documentum CM, the authentication success is the same as the successful retrieval of the Foundation Java API session.
- When the token is not supported or authenticated, an unsuccessful authentication handler handles the error. The handler usually clears the local security context and delegates to an entry point to return the error Response, such as a 401 status code.
- When the token is authenticated, a successful authentication handler can return the Response directly or it can continue to delegate it to other filters in the chain.

- After a successful authentication, the authenticated token is stored in the local security context to be used by the controller that handles the business logic of the Request. For Foundation REST API, `<RepositoryContextHolder>` stores the authenticated user credential during the handling of a request.
- After the Request handling finishes, the security context can determine whether to delete the authenticated token or cache it for a stateful session. The Foundation REST API `<RepositoryContextHolder>` always cleans up the local security context. This behavior is by design.

During the deployment phase, usually you are only required to configure REST runtime properties in order to enable a specific authentication scheme. For example, `rest.security.auth.mode=basic`.

8.4.3 Authentication extension

The custom authentication scheme provides the same convenience of deployment as other Core authentication schemes. It is important to follow the Core REST authentication extension development model when developing a custom authentication scheme. Similar to Documentum REST Services Extensibility, authentication extension development requires you to use Documentum REST Services Archetype to create a sub module for your custom authentication scheme. After you've created your custom authentication scheme, you must package it as a dependent library of the Foundation REST API WAR. The development lifecycle of a custom authentication scheme is illustrated as follows:

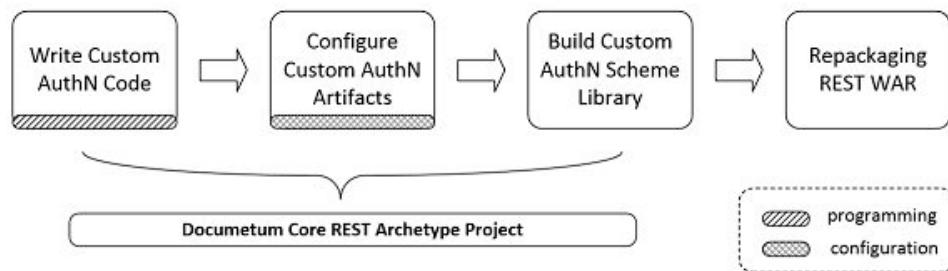


Diagram-3 Lifecycle of Building Custom Authentication Schemes

The amount of work required to develop a custom authentication scheme depends on third-party software used and OpenText Documentum CM dependency support. The following diagram shows the implementation stack of a custom authentication scheme and some possible extension points:

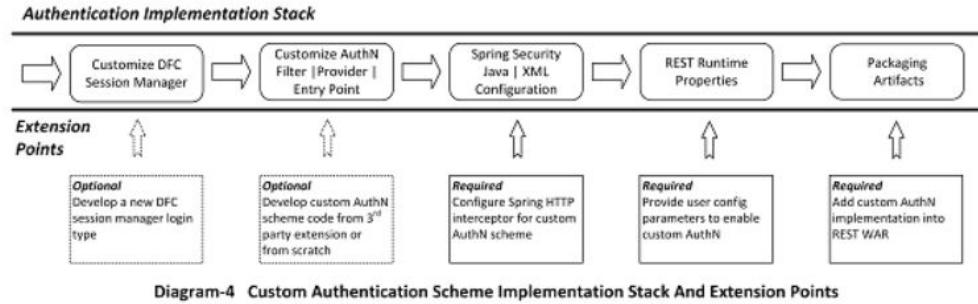


Diagram-4 Custom Authentication Scheme Implementation Stack And Extension Points

8.4.3.1 Customize Foundation Java API session manager

The login for Documentum CM Server requires a Foundation Java API session manager, which must be set up. Foundation REST API provides an interface called `com.emc.documentum.rest.dfc.RepositorySessionManager` to use for Foundation REST API server code that instantiates the Foundation Java API session manager. This interface provides the following `IDfSessionManager` public method:

```
public IDfSessionManager get(String repositoryName, String user, Object credential,
AuthType authType)
```

Depending on the authentication type used, the credentials are different for each logged in user. By default, the following authentication types are supported:

Authentication Type	Description	Credential
DEFAULT	Default login , same to PASSWORD.	User password
PASSWORD	Password login	User password
LOGIN_TICKET	OpenText Documentum CM Login Ticket login	Login ticket in BASE64 string
PRINCIPAL	Documentum CM Server principal login	No credentials
CUSTOM	Preserved for user defined login	User defined credential

Foundation REST API provides a default implementation for the `com.emc.documentum.rest.dfc.RepositorySessionManager` interface. The implementation class is `com.emc.documentum.rest.dfc.impl.MemoryRepositorySessionManager`. As its name implies, this class puts the initialized session manager object into server memory (`Ehcache`) for reuse. Caching gives it the best performance with Documentum CM Server login.



Note: To avoid possible session leaks, the `com.emc.documentum.rest.dfc.impl.MemoryRepositorySessionManager` class never caches the Foundation Java API session . It is recommended that you use the default implementation to initialize Foundation Java API sessions.

There are two ways to get the implementation instance:

1. Call `MemoryRepositorySessionManager.INSTANCE`. The implementation is a Singleton, so the easiest way to use it is to call its `INSTANCE` Singleton.
2. Get it from `com.emc.documentum.rest.security.provider.AbstractAuthProvider`. You can get the session manager instance by extending from the abstract provider.

8.4.3.1.1 When to customize

In most cases, you do not need to customize Foundation Java API session managers because the default implementation already has support for most authentication types that are also supported by Documentum CM Server. A custom authentication scheme should first examine the existing authentication types to see whether any of them can be used for the custom authentication scheme. You can use the `CUSTOM` authentication type when none of the default implementations meet your requirements. The default implementation creates an empty Foundation Java API session manager for the authentication type `CUSTOM`. You must set up custom identities for the obtained session manager.

Example 8-6: A custom Foundation Java API manager

Here's a code sample that shows you a custom Foundation Java API manager:

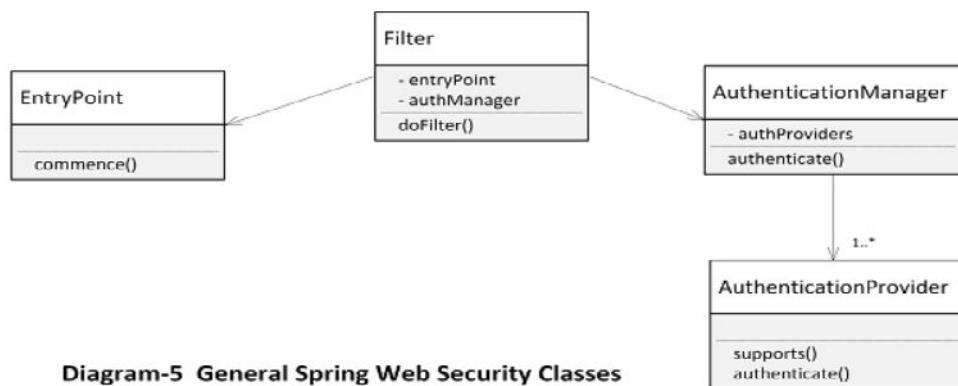
```
// Get memory repository session manager
RepositorySessionManager repoSessionManager = MemoryRepositorySessionManager.INSTANCE;
IDfSessionManager dfcSessionManager = repoSessionManager.get("ACME", "dave",
"5XHR6ftZ==",
AuthType.CUSTOM);

// Check whether it is initialized -- if it is not the first time you login
// then it may already be initialized
if (!dfcSessionManager.hasIdentity("ACME")) {
    // Set identity
    IDfLoginInfo login = new DfLoginInfo();
    login.setUser("dave");

    // This is a dummy password plugin sample -- set a valid password plugin which is
    // supported by DFC login.setPassword("DM_PLUGIN=dm_custom/" + "5XHR6ftZ==");
    dfcSessionManager.setIdentity("ACME", login);
}
```

8.4.3.2 Customize authentication filter, provider, and entry point

Spring Security introduces the concepts of authentication filter, manager, provider, and entry point. They work together to authenticate a web Request. Here's a diagram that shows the Spring web security classes:

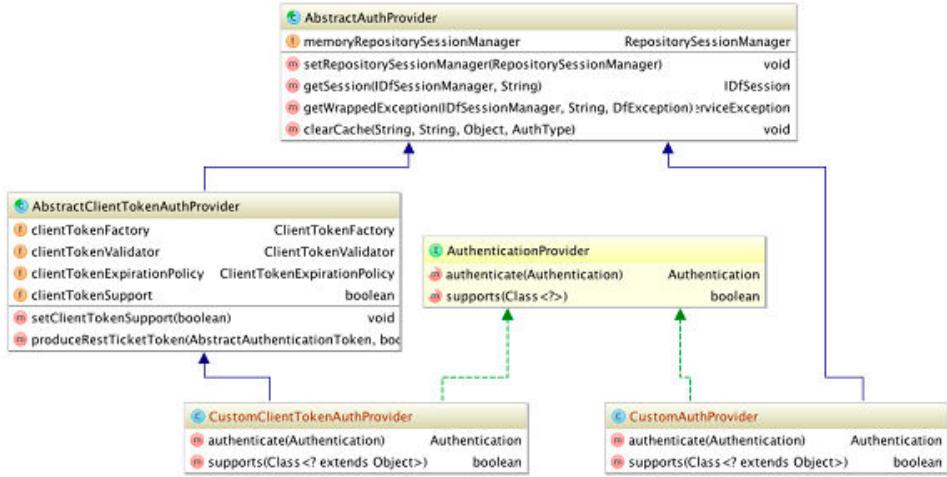


A custom authentication scheme may directly use an out-of-the-box security filter, such as a Spring dependency, to authenticate the Request, or it may extend from third-party software. Spring Security provides a number out-of-the-box security filters. You can also add your own filters. Authentication managers provide the authentication services for filters. You cannot customize the authentication managers, but you can register different authentication providers. Each authentication provider authenticates a specific authentication token.

8.4.3.2.1 Authentication Provider

Developers often write their own authentication providers, because the provider must call the Core REST session manager to authenticate the user. Spring Security provides an interface called *AuthenticationProvider*, which is used to implement the provider. Foundation REST API additionally provides two abstract classes for extension, the *AbstractAuthProvider* class and the *AbstractClientTokenAuthProvider* class.

- *AbstractClientTokenAuthProvider* provides methods to call the Foundation Java API session manager to authenticate the user. Extend it when the authentication requires a Documentum CM Server login.
- *AbstractClientTokenAuthProvider* provides methods to exchange the authenticated token with a Client Token. Extend it when authentication success requires a Documentum CM Server login, then return a Client Token cookie.



Example 8-7: The CustomAuthenticationProvider class

```

public class CustomAuthenticationProvider extends AbstractAuthProvider
    implements AuthenticationProvider {
    @Override
    public boolean supports(Class<? extends Object> authentication) {
        return UsernamePasswordAuthenticationToken.class.isAssignableFrom(authentication);
    }
    @Override
    public Authentication authenticate(Authentication authentication)
        throws AuthenticationException {
        if (authentication == null || Strings.isNullOrEmpty(authentication.getName())) {
            throw new UsernameNotFoundException(MessageBundle.INSTANCE
                .get("E_BAD_CREDENTIALS_ERROR"));
        }
        String repositoryId = RepositoryContextHolder.getRepositoryName();
        String user = (String) authentication.getPrincipal();
        String password = (String) authentication.getCredentials();
        IDfSessionManager sm = null;
        IDfSession session = null;
        try {
            sm = memoryRepositorySessionManager.get(repositoryId, user, password,
                AuthType.PASSWORD);
            session = getSession(sm, repositoryId);
            Collection<GrantedAuthority> grantedAuthorities = new
            ArrayList<GrantedAuthority>();
            grantedAuthorities.add(new SimpleGrantedAuthority("ROLE_USER"));
            AbstractAuthenticationToken authorizedToken = new
            UsernamePasswordAuthenticationToken(
                authentication.getPrincipal(),
                authentication.getCredentials(),
                grantedAuthorities);
            if (!authorizedToken.isAuthenticated()) {
                authorizedToken.setAuthenticated(true);
            }
            return authorizedToken;
        } catch (DfException e) {
            clearCache(repositoryId, user, password, AuthType.PASSWORD);
            throw getWrappedException(sm, repositoryId, e);
        } finally {
            DfcSessions.release(session);
        }
    }
}
  
```

```
}
```



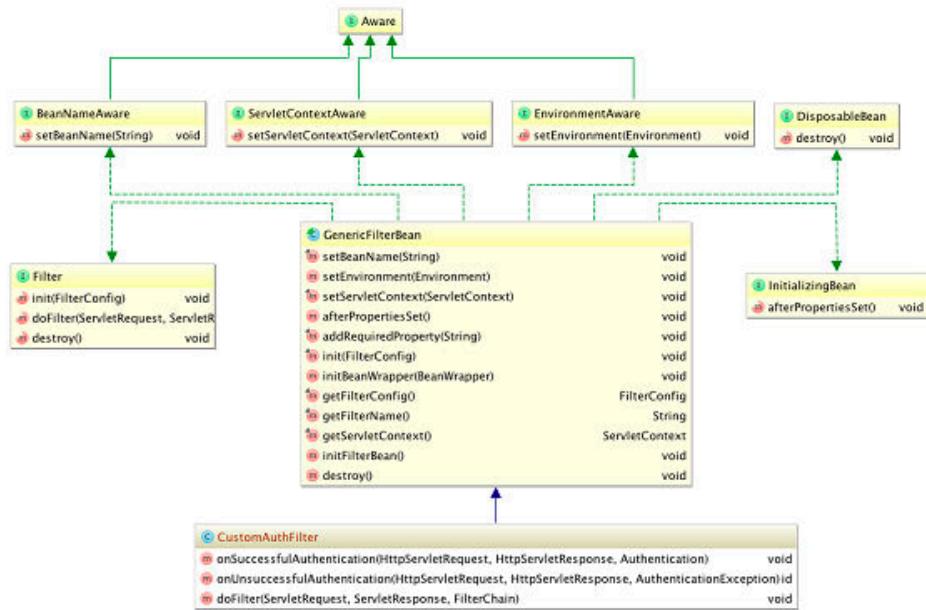
Example 8-8: The CustomAuthenticationProvider class

```
public class CustomAuthenticationProvider extends AbstractClientTokenAuthProvider
    implements AuthenticationProvider {
    ...
    @Override
    public Authentication authenticate(Authentication authentication)
        throws AuthenticationException {
        ...
        try {
            ...
            AbstractAuthenticationToken authorizedToken = new
                UsernamePasswordAuthenticationToken(
                    authentication.getPrincipal(),
                    authentication.getCredentials(),
                    grantedAuthorities);
            if (clientTokenSupport) {
                authorizedToken = produceRestTicketToken(authorizedToken, false, session);
            }
            authorizedToken.setDetails(authentication.getDetails());
            if (!authorizedToken.isAuthenticated()) {
                authorizedToken.setAuthenticated(true);
            }
            return authorizedToken;
        } catch (DfException e) {
            ...
        } finally {
            ...
        }
    }
}
```



8.4.3.2.2 Authentication filter

You can extend from an existing third-party security filter to create your custom authentication filter, or you can implement your custom filter by using an abstract class or interface such as *javax.servlet.Filter*. Here's a diagram of the filter UML:



In the previous section we said that the authentication provider calls the Foundation Java API session manager to login. The session is actually released immediately after the login success. It is important that your filter implementation have the logic to store and clear the security context. The security context is used by the resource controller to obtain the Foundation Java API session manager in the following cases:

- When an authentication succeeds, the authentication filter must store the user credential into *RepositoryContextHolder*.
- When an authentication fails, the authentication filter must clean up the *RepositoryContextHolder*.

Example 8-9: Custom security filter

Here's a code sample that shows you how to implement a custom security filter:

```

// Authenticate the servlet request
@Override
public void doFilter(ServletRequest request, ServletResponse Response, FilterChain chain)
    throws IOException, ServletException {
    if (!requireHandle(request)) {
        chain.doFilter(request, Response);
        return;
    }

    try {
        AuthenticationToken authToken = parseAuthToken(request);
        Authentication authResult = authenticationManager.authenticate(authToken);
        SecurityContextHolder.getContext().setAuthentication(authResult);
        onSuccessfulAuthentication(request, Response, authResult);
    }
    catch (AuthenticationException failed) {
        SecurityContextHolder.clearContext();
    }
}

```

```

        onUnsuccessfulAuthentication(request, Response, failed);

        if (ignoreFailure) {
            chain.doFilter(request, Response);
        }
        else {
            authenticationEntryPoint.commence(request, Response, failed);
        }
        return;
    }

    chain.doFilter(request, Response);
}

// success handler
protected void onSuccessfulAuthentication(HttpServletRequest request,
    HttpServletResponse Response,
    Authentication authResult) throws IOException {
    RepositoryContextHolder.setLoginName(authResult.getPrincipal().toString());
    RepositoryContextHolder.setPassword(authResult.getCredentials());
    RepositoryContextHolder.setAuthType(AuthType.PASSWORD);
}

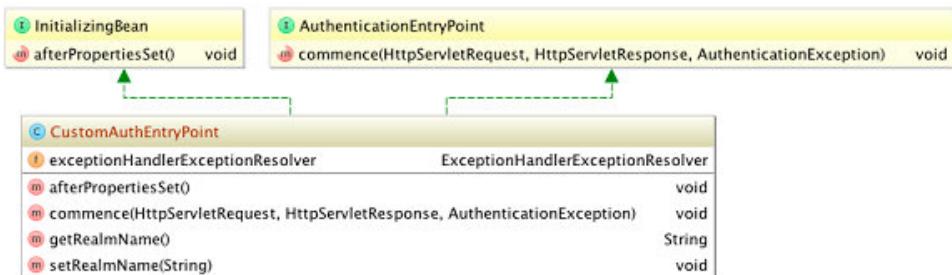
// failure handler
protected void onUnsuccessfulAuthentication(HttpServletRequest request,
    HttpServletResponse Response,
    AuthenticationException failed) throws IOException {
    RepositoryContextHolder.clearUserCredentials();
}
}

```



8.4.3.2.3 Authentication Entry Point

Authentication errors, such as HTTP status 401 for error or status 302 for redirecting, must be returned to the client in the correct manner. Using an entry point helps to implement this logic.



An entry point must at least implement the *Commence* method, and then use that method to set the Response Header and status, as well as handle the Response body.

Example 8-10: Custom Entry Point

Here is a code sample that shows you a custom authentication entry point:

```

@Override
public void commence(HttpServletRequest request, HttpServletResponse Response,
    AuthenticationException authException)
    throws IOException, ServletException {
}

```

```

        Response.setStatus(HttpStatus.UNAUTHORIZED);
        Response.addHeader(ChallengeConfig.WWW_AUTHENTICATE, getChallenge(request,
AuthSchemes.BASIC.value())); exceptionHandlerExceptionResolver
.resolveException(request, Response, null, authException);
}

```



8.4.3.3 Documentum client token integration

The custom authentication scheme can integrate with a DOCUMENTUM-CLIENT-TOKEN in a scenario where a successful custom authentication login results in a DOCUMENTUM-CLIENT-TOKEN cookie being sent back in the Response.

After receiving the Response, the Foundation REST API client can access Foundation REST API by using the Client Token cookie for subsequent requests. This approach avoids having to go through the custom authentication scheme to authenticate the Foundation REST API client every time. When a custom authentication is used with a DOCUMENTUM-CLIENT-TOKEN cookie, a log out must be set up. This can be achieved by putting Client Token filters and a custom authentication filter into an ordered filter chain. The following are the instructions for the filter sequence:

others

↓

ClientTokenPreAuthFilter

↓

*Custom Authentication Filter for URL pattern /repositories/***

↓

ClientTokenPostAuthFilter

↓

others

- *ClientTokenPreAuthFilter* is a Core REST security filter that validates the client token from the Request cookie Header. It must filter Requests prior to the other two filters.
- *Custom authentication filter* is the developer implemented authentication filter that authenticates the client token from the Request and stores a *ClientTokenAuthenticationToken* into *Spring SecurityContextHolder*. The custom authentication filter must use a provider implementation of *AbstractClientTokenAuthProvider* to produce the *ClientTokenAuthenticationToken* client token after a successful login.
- *ClientTokenPostAuthFilter* is the other Core REST security filter that sends back the Client Token cookie and Headers for the Response after the custom login succeeds or when the client token must be refreshed.

8.4.3.4 Spring security Java configuration

Spring Security provides Java Configuration and Security Namespace (XML) Configuration to configure the security filters for the web application. Both approaches provide the same capabilities for you to configure Spring security. Foundation REST API supports both configurations, but the Java configuration is preferred.

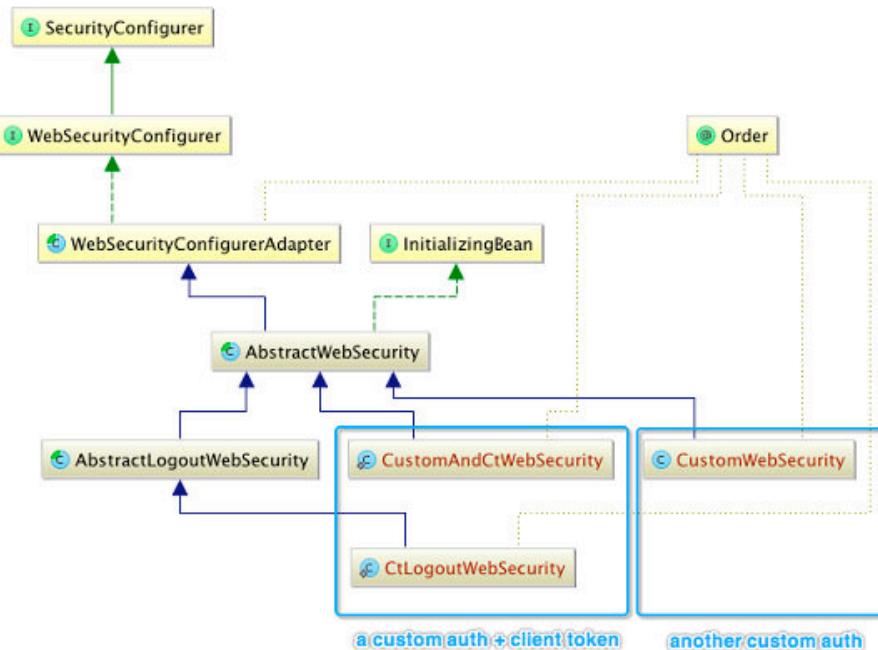
In this section, we introduce the optimized Java configuration for developing new authentication schemes.

8.4.3.4.1 Implement web security

For Spring Security Java configuration, you typically implement a specific authentication scheme by extending a class from `WebSecurityConfigurerAdapter` and annotating it with `@EnableWebSecurity`.

You can do the same when implementing a OpenText Documentum CM custom authentication. To ease the effort required for development, Foundation REST API provides an optimized abstraction for you to develop the custom web security in the `documentum-rest-security-core.jar` file.

The following diagram shows the flow of web security integration:



8.4.3.4.1.1 AbstractWebSecurity

Core REST API *AbstractWebSecurity* implements *InitializingBean* and leaves five methods for the implementation to override.



Note: There are two annotations applied to the custom web security: *@AuthSchemeProfile* and *@Order*.

Example 8-11: A custom web security sample

```
/*
 * Custom authentication
 */
@AuthSchemeProfile(schemes = CustomWebSecurity.MODE)
@Order(1)
public class CustomWebSecurity extends AbstractWebSecurity {

    public static final String MODE = "custom";

    @Override
    protected AuthenticationProvider[] authenticationProviders() {
        return new AuthenticationProvider[] { customAuthProvider() };
    }

    @Override
    protected AuthenticationEntryPoint entryPoint() {
        return customAuthEntryPoint();
    }

    @Override
    protected void configureSecurityFilters(HttpSecurity http) throws Exception {
        http.addFilterBefore(customAuthFilter(), BasicAuthenticationFilter.class);
    }

    @Bean
    public CustomAuthFilter customAuthFilter() {
        return new CustomAuthFilter(authenticationManagerBean());
    }

    @Bean
    public CustomAuthProvider customAuthProvider() {
        return CustomAuthProvider();
    }

    @Bean
    public CustomAuthEntryPoint customAuthEntryPoint() {
        return CustomAuthEntryPoint();
    }
    @Bean
    public SecurityFilterChain customWebSecurityFilterChain(HttpSecurity http) throws
Exception {
        return super.securityFilterChain(http);
    }

    @Bean
    public WebSecurityCustomizer customWebSecurityCustomizer() {
        return super.webSecurityCustomizer();
    }
}
```



In the preceding example, the two Beans (in bold) *customWebSecurityFilterChain* and *customWebSecurityCustomizer* must be registered with *SecurityFilterChain* and

WebSecurityCustomizer respectively for the supported version of Spring where *custom* (in italics) is your authentication mechanism name.

OpenText recommends that you follow the naming convention (prefixing the authentication mechanism name with *WebSecurityFilterChain* and *WebSecurityCustomizer*) for the beans to be unique and to register the beans with SecurityFilterChain and WebSecurityCustomizer.

8.4.3.4.1.2 **@AuthSchemeProfile**

This is a Core REST annotation to name your custom authentication implementation. It is used, in conjunction with the Spring annotations *@EnableWebSecurity*, *@Configuration*, and *@Conditional*, to make this class work as a Spring security configuration. When the annotation is named, you can enable the authentication scheme at runtime by using the REST runtime properties. Here is a code sample that shows you how to do this:

```
@AuthSchemeProfile("custom") --> rest.security.auth.mode=custom
```

8.4.3.4.1.3 **@Order**

Multiple configurations can be enabled for an authentication scheme. When multiple configurations are enabled for an authentication scheme, the annotation *@Order* specifies the order of priority in which the configurations work, relative to each other. The lower the value, the higher the priority. The number range for *@Order* is from 1 to 98 for a custom configuration. All other numbers are reserved.

8.4.3.4.2 **Client token integration**

For client token integration, a web security implementation must create multiple inner classes. One of those inner classes is used to configure the HTTP logout security. Here is a code sample that shows you how to do this:

➡ Example 8-12: Custom web security sample with client token integration

```
/**  
 * Custom + client token authentication.  
 */  
@AuthSchemeProfile(schemes = CustomCtCtWebSecurity.MODE)  
public class CustomCtCtWebSecurity {  
  
    public static final String MODE = "custom-ct";  
  
    @AuthSchemeProfile(schemes = CustomCtCtWebSecurity.MODE)  
    @Order(1)  
    public static class CtLogoutWebSecurity extends AbstractLogoutWebSecurity {  
    }  
  
    @AuthSchemeProfile(schemes = CustomCtCtWebSecurity.MODE)  
    @Order(2)  
    public static class CustomAndCtWebSecurity extends AbstractWebSecurity {  
  
        @Autowired  
        protected CtAuthBeans ctBeans;  
  
        @Override  
        protected AuthenticationProvider[] authenticationProviders() {  
            return new AuthenticationProvider[] {
```

```
        customAuthProviderWithClientToken(),
        ctBeans.clientTokenAuthProvider()
    );
}

@Override
protected AuthenticationEntryPoint entryPoint() {
    return customEntryPoint();
}

@Override
protected void configureSecurityFilters(HttpSecurity http) throws Exception {
    http
        .addFilterBefore(customAuthFilter(), BasicAuthenticationFilter.class)
        .addFilterAfter(clientTokenPreAuthFilter(), BasicAuthenticationFilter.class)
        .addFilterAfter(clientTokenPostAuthFilter(),
ClientTokenPreAuthFilter.class);
}

@Bean
public CustomAuthFilter customAuthFilter() {
    return new CustomAuthFilter(authenticationManagerBean());
}

@Bean
public CustomAuthProvider customAuthProvider() {
    return CustomAuthProvider();
}

@Bean
public CustomAuthEntryPoint customAuthEntryPoint() {
    return CustomAuthEntryPoint();
}

@Bean
public ClientTokenPreAuthFilter clientTokenPreAuthFilter() {
    return ctBeans.clientTokenPreAuthFilter(authenticationManagerBean());
}

@Bean
public ClientTokenPostAuthFilter clientTokenPostAuthFilter() {
    return ctBeans.clientTokenPostAuthFilter();
}
}
```

- The first inner class `CtLogoutWebSecurity` extends from `AbstractLogoutWebSecurity` and does not need any customization. This configuration handles client token logout. It is annotated with `@AuthSchemeProfile(ct - "custom")` and `@Order(1)`.
 - The second inner class `CustomAndCtWebSecurity` extends from `AbstractWebSecurity` and performs customizations similar to `CustomWebSecurity`. This configuration performs custom login with client token integration. The difference is it must add the client token filters into its `configureSecurityFilters(HttpSecurity http)` override method. It is also annotated with the `@AuthSchemeProfile(ct - "custom")` annotation, but has a lower prioritized order of `@Order(2)`.



Note: Core REST API *CtAuthBeans* provides client token authentication beans for you to import into your custom web security configuration.

**Caution**

For client token integration, when the custom authentication filter wants to send a redirect URI to the Response after a successful authentication, it must not call the Response redirect directly. Instead, it must pass the redirected URL to the authenticated token and let the next filter, the *ClientTokenPostAuthFilter* filter, redirect the URI. Otherwise, the Response does not set the client token cookie correctly.

**Example 8-13: Set a client token cookie**

Here is a code sample that shows you how to set a client token cookie:

```
// Delegate to Core ClientTokenPostAuthFilter to do redirect
ClientTokenAuthToken authenticated = (ClientTokenAuthToken)
    getAuthenticationManager().authenticate(token);
if (authenticated.isAuthenticated()) {
    authenticated.setRedirectUri(redirectUrl);
    SecurityContextHolder.getContext().setAuthentication(authenticated);
}
```

**8.4.3.4.3 Externalize configuration parameters**

In a custom authentication configuration, some variables must be externalized so that they are only specified at runtime, not compile time. You can implement a runtime property class that imports variables from *rest-api-runtime.properties*. This class can be accessed by your implementation classes to import external runtime variables.

**Example 8-14: Runtime properties**

```
/**
 * Default custom security runtime properties values.
 * Properties set in 'rest-api-runtime.properties' will override the default.
 */
@Configuration
@PropertySource("classpath:rest-api-runtime.properties")
public class CustomRuntime {

    /**
     * A static place holder to load properties from the file 'rest-api-
     * runtime.properties'.
     *
     * @return placeholder configurer
     */
    @Bean
    public static PropertySourcesPlaceholderConfigurer propertyPlaceholderConfigurer() {
        return new PropertySourcesPlaceholderConfigurer();
    }

    /**
     * Property value for 'rest.security.custom.key'. Defaults to empty.
     */
    @Value("${rest.security.custom.key:}")
    public String key;

    /**
     * Property value for 'rest.security.custom.max_age'. Defaults to 3600.
     */
}
```

```

    @Value("${rest.security.custom.max_age:3600}")
    public Integer maxAge;
}

```



8.4.3.4.4 Configure runtime properties

In the custom REST war deployment, you can enable the custom authentication scheme and configure each parameter with the *rest-api-runtime.properties* file.

➤ Example 8-15: Runtime properties for custom authentication

```

set authentication mode (required)
rest.security.auth.mode=ct-custom

# set realm name
rest.security.realm.name=ACME.COM

# set Java configuration package (required)
rest.context.config.location=com.acme.rest.security.config

# set other parameters
rest.security.custom.max_age=60
rest.security.custom.key=A42gHx47X

```



Note: The runtime property *rest.context.config.location* requires you to specify the Java package of the configuration classes that extend from *AbstractWebSecurity* or *AbstractLogoutWebSecurity*. This is a requirement for Java configuration.

8.4.3.5 Spring security XML configuration

For developers that are more familiar with XML configuration, it is possible to configure the security in XML. This is not the recommended solution because additional security extensions in Foundation REST API are based on Java configurations.

For XML configuration, you write the XML file to configure HTTP security for OpenText Documentum CM custom authentication. The XML file must be packaged with the jar file. The content of the XML configuration is a Spring *<http>* configuration and relevant bean definitions.

For more information on how to configure a custom filter, see the *Spring Security Reference* called Adding in Your Own Filters.

8.4.3.5.1 Write XML configuration

Here is a code sample that shows you the `<http>` configuration for a custom authentication sample that is the same as the Java configuration `CustomWebSecurity`. It is assumed that the custom file name is `rest-api-ct-custom-security.xml`.

Example 8-16: Custom authentication XML configuration sample 1

```

<!-- import properties -->
<beans:bean id="propertyConfigurer" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <beans:property name="locations">
        <beans:list>
            <beans:value>classpath:rest-api-runtime.properties</beans:value>
        </beans:list>
    </beans:property>
</beans:bean>

<!-- HTTP security configuration: custom basic authentication -->
<http use-expressions="true" create-session="never"
      entry-point-ref="customAuthEntryPoint">
    <custom-filter ref="customAuthFilter" before="BASIC_AUTH_FILTER"/>
    <intercept-url pattern="/repositories/**" access="hasRole('ROLE_USER')"/>
    <csrf disabled="true"/>
</http>

<!-- filter definition: customized HTTP authentication filter -->
<beans:bean id="customAuthFilter" class="com.acme.security.filter.CustomAuthFilter">
    <beans:constructor-arg ref="customAuthManager"/>
</beans:bean>

<!-- authentication manager definition: custom authentication providers -->
<authentication-manager alias="customAuthManager" erase-credentials="false">
    <authentication-provider ref="customAuthProvider"/>
</authentication-manager>

<!-- provider definition -->
<beans:bean id="customAuthProvider"
           class="com.acme.security.provider.CustomAuthProvider"/>

<!-- entry point definition -->
<beans:bean id="customEntryPoint" class="com.acme.security.entry.CustomAuthEntryPoint">
    <beans:property name="realmName" value="${rest.security.realm.name}"/>
</beans:bean>

```



Example 8-17: Custom authentication XML configuration sample 2

Here is an XML code sample of the `<http>` configuration for the custom authentication that is the same as the Java configuration `CustomCtWebSecurity`.

```

<!-- import properties -->
<beans:bean id="propertyConfigurer" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <beans:property name="locations">
        <beans:list>
            <beans:value>classpath:rest-api-runtime.properties</beans:value>
        </beans:list>
    </beans:property>
</beans:bean>

<import resource="classpath*:META-INF/spring/template/rest-api-ct-security" />

<!-- HTTP security configuration: ct sign out -->

```

```

<http use-expressions="true" create-session="never" entry-point-ref="basicEntryPoint"
      pattern="/logout">
    <custom-filter ref="dctmClientTokenSignOutFilter" position="LOGOUT_FILTER"/>
    <intercept-url pattern="/logout" access="ROLE_USER"/>
    <csrf disabled="true"/>
</http>
<!-- HTTP security configuration: custom basic authentication -->
<http use-expressions="true" create-session="never"
      entry-point-ref="customAuthEntryPoint">
    <custom-filter ref="customAuthFilter" before="BASIC_AUTH_FILTER"/>
    <custom-filter ref="clientPreTokenFilter" position="BASIC_AUTH_FILTER"/>
    <custom-filter ref="clientPostTokenFilter" after="BASIC_AUTH_FILTER"/>
    <intercept-url pattern="/repositories/**" access="hasRole('ROLE_USER')"/>
    <csrf disabled="true"/>
</http>

<!-- filter definition: customized HTTP authentication filter -->
<beans:bean id="customAuthFilter" class="com.acme.security.filter.CustomAuthFilter">
    <beans:constructor-arg ref="customAuthManager"/>
</beans:bean>

<!-- authentication manager definition: custom authentication providers -->
<authentication-manager alias="customAuthManager" erase-credentials="false">
    <authentication-provider ref="customAuthProvider"/>
    <authentication-provider ref="clientTokenAuthProvider"/>
</authentication-manager>

<!-- filter definition -->
<beans:bean id="customAuthFilter" class="com.acme.security.provider.CustomAuthFilter">
    <beans:constructor-arg index="0" name="authenticationManager"
        ref="customAuthManager"/>
</beans>

<!-- provider definition -->
<beans:bean id="customAuthProvider" class="com.acme.security.provider
    .CustomClientTokenAuthProvider">
    <beans:property name="clientTokenSupport" value="true"/>
</beans>

<!-- entry point definition -->
<beans:bean id="customEntryPoint" class="com.acme.security.entry.CustomAuthEntryPoint">
    <beans:property name="realmName" value="${rest.security.realm.name}"/>
</beans:bean>
<!-- filter definition: Documentum REST Services client token authentication filters -->
<beans:bean id="clientTokenPreFilter" class="com.emc.documentum.rest.security.filter
    .ClientTokenPreAuthFilter">
    <beans:property name="authenticationManager" ref="customAuthManager"/>
</beans:bean>
<beans:bean id="clientTokenPostFilter" class="com.emc.documentum.rest.security.filter
    .ClientTokenPostAuthFilter"/>

<!-- filter definition: customized Client Token sign out filter -->
<beans:bean name="dctmClientTokenSignOutFilter"
    class="com.emc.documentum.rest.security.filter.ClientTokenSignOutFilter">
    <beans:constructor-arg index="0" name="logoutSuccessUrl"
        value="${rest.security.logout.success.url}"/>
</beans:bean>

```



Note: The rest-api-ct-security.xml file is imported into the XML file. The rest-api-ct-security.xml file provides beans for client token authentication, and it is the same as the *CtAuthBeans* Java configuration class.

8.4.3.5.2 Configure Security XML Extension

To register your custom authentication configuration XML file into the Foundation REST API, you must update the \${war}/META-INF/spring/extension/rest-api-custom-security.xml file.

Example 8-18: Modify security extension for custom authentication

```
<beans:beans>
    <!-- ===== -->
    <!-- CAUTION: XML CONFIGURATION IS DISCOURAGED.
        USE JAVA CONFIGURATION INSTEAD. -->
    <!-- ===== -->

    <!-- Legacy XML configuration for adding custom authentication beans -->
    <!-- Adds a custom authentication XML configuration in two steps: -->

    <!-- Step 1: Import system default anonymous URL patterns -->
    <beans:import resource="rest-api-anonymous-security"/>

    <!-- Step 2: Import oauth2 XML configuration file -->
    <beans:import resource="classpath*:rest-api-ct-custom-security"/>
</beans:beans>
```



8.4.3.5.3 Configure runtime properties

Similar to the Java configuration, in the custom REST war deployment, you can enable the custom authentication scheme and configure each parameters through *rest-api-runtime.properties*.

Example 8-19: Modify security extension for custom authentication

```
# set authentication mode (required)
rest.security.auth.mode=ct-custom

# set realm name
rest.security.realm.name=ACME.COM

# set other parameters
rest.security.custom.max_age=60
rest.security.custom.key=A42gHx47X
```



8.4.3.6 Packaging artifacts

The packaging process produces a custom authentication jar file that includes all implemented classes and Spring XML configurations. You can package the jar file into the Foundation REST API war file manually or by using the war overlay of Documentum REST Services Archetype project.

8.4.3.7 Samples

The Foundation REST API SDK provides two samples that show you how to build custom authentications. One sample is for a custom login form with a client token, the other sample is for an OAuth2 login.

Refer to the *Foundation REST API SDK* for the sample code.

8.4.4 Tutorial: Foundation REST API authentication extensibility development

In this section, we discuss how to setup a Maven project to develop a custom authentication scheme. It is recommended that you create a custom authentication project based on the Documentum REST Services Archetype.

For more information, see [Get Started With the Development Kit](#).

The project structure for the extended authentication scheme is shown here. In the following sample we give you the Java configuration, and we name the new authentication scheme *ct-custom*.

For XML configuration samples, see the Foundation REST API SDK.

Example 8-20: Maven project layout

```

acme-rest (from archetype)
  └── acme-rest-custom-security (new)
    └── src
      └── main
        └── java
          └── com
            └── acme
              └── security
                └── config
                  └── CustomRuntime.java
                  └── CustomWebSecurity.java
                └── entry
                  └── CustomEntryPoint.java
                └── filter
                  └── CustomAuthFilter.java
                └── provider
                  └── CustomClientTokenAuthProvider.java
      └── resources
    └── test
      └── java
      └── resources
    └── pom
  └── acme-rest-web (from archetype)
    └── src
      └── main
        └── resources
          └── rest-api-runtime.properties
    └── pom
  └── pom

```

acme-rest-custom-security

This is a new Maven sub module that is added to the Archetype project. It includes all the Java code and configurations for the custom authentication scheme. The build output of the module is a jar file.

► Example 8-21: CustomRuntime.java

Custom runtime allows you to read custom runtime properties in Java configuration.

```
/**  
 * Default custom security runtime properties values.  
 * Properties set in 'rest-api-runtime.properties' will override the default.  
 */  
@Configuration  
@PropertySource("classpath:rest-api-runtime.properties")  
public class CustomRuntime {  
  
    /**  
     * A static place holder to load properties from the file 'rest-api-  
     * runtime.properties'.  
     *  
     * @return placeholder configurer  
     */  
    @Bean  
    public static PropertySourcesPlaceholderConfigurer propertyPlaceholderConfigurer() {  
        return new PropertySourcesPlaceholderConfigurer();  
    }  
  
    /**  
     * Property value for 'rest.security.custom.key'. Defaults to empty.  
     */  
    @Value("${rest.security.custom.key:}")  
    public String key;  
}
```

CustomWebSecurity.java

Follow the instructions in [Java Configuration](#) to implement custom Web Security. The *CustomWebSecurity* sample code shows the logic to implement multiple HTTP security for an authentication scheme.

CustomEntryPoint.java

Follow the instructions in [“Authentication Entry Point” on page 330](#) to write the *CustomEntryPoint* code. HTTP status 401 and 302 are the most often used codes to indicate an authentication failure.

- According to RFC 7235, when an HTTP status 401 is used, a response Header *WWW-Authenticate <challenge>* should also be returned.

The authentication exception must be transformed into an XML or JSON error. You can access Spring resolver *org.springframework.web.servlet.mvc.method.annotation.ExceptionHandlerExceptionResolver* into the *CustomEntryPoint* class to resolve the authentication exception in the servlet Response.

For more information, see the sample code at [“Authentication Entry Point” on page 330](#).

- When an HTTP status 302 is used to indicate a user login failure, the user must be redirected to a login page or the other authentication service. Here is a code sample that shows HTTP redirection:

 **Example 8-22: HTTP Redirection on Login Failure**

```

@Override
public void commence(HttpServletRequest request, HttpServletResponse Response,
                     AuthenticationException authException)
                     throws IOException, ServletException {
    if (Response.isCommitted()) {
        return;
    }
    // use Spring redirect strategy
    new DefaultRedirectStrategy().sendRedirect(request, Response, "http://acme.com/
sign-on");
}

```



`CustomAuthFilter.java`

Follow “[Authentication filter](#)” on page 328 to write the `CustomAuthFilter` code.

`CustomClientTokenAuthProvider.java`

Follow the instructions at “[Authentication Provider](#)” on page 326 to write the `CustomClientTokenAuthProvider` code.

An authentication provider does two jobs:

- Authenticate the user with Foundation Java API session manager
- Produce and return an authenticated token

To view a code sample, see “[Authentication Provider](#)” on page 326.

Client Token Support

When a custom authentication scheme is used with a DOCUMENTUM-CLIENT-TOKEN cookie, the provider must return the authenticated token as a `ClientTokenAuthToken` token.

 **Example 8-23: Example of Custom Authentication Provider**

```

@Override
public Authentication authenticate(Authentication authentication)
                     throws AuthenticationException {
    if (authentication == null || Strings.isNullOrEmpty(authentication.getName())) {
        throw new UsernameNotFoundException(MessageBundle
                     .INSTANCE.get("E_BAD_CREDENTIALS_ERROR"));
    }
}

String repositoryId = RepositoryContextHolder.getRepositoryName();
String user = (String) authentication.getPrincipal();
String password = (String) authentication.getCredentials();
IDfSessionManager sm = null;
IDfSession session = null;
try {

```

```

        sm = memoryRepositorySessionManager.get(repositoryId, user, password,
AuthType.PASSWORD);
        session = getSession(sm, repositoryId);
        Collection<GrantedAuthority>
        grantedAuthorities = new ArrayList<GrantedAuthority>();
        grantedAuthorities.add(new SimpleGrantedAuthority("ROLE_USER"));
        AbstractAuthenticationToken authorizedToken =
            new UsernamePasswordAuthenticationToken(
                authentication.getPrincipal(),
                authentication.getCredentials(),
                grantedAuthorities);
        ///// CHANGE CODE BELOW
        // ClientTokenAuthToken is produced
        if (clientTokenSupport) {
            authorizedToken = produceRestTicketToken(authorizedToken, false, session);
        }
        ///// ADD CODE ABOVE
        if (!authorizedToken.isAuthenticated()) {
            authorizedToken.setAuthenticated(true);
        }
        return authorizedToken;
    } catch (DfException e) {
        clearCache(repositoryId, user, password, AuthType.PASSWORD);
        throw getWrappedException(sm, repositoryId, e);
    } finally {
        DfcSessions.release(session);
    }
}
}

```



pom.xml (custom-security)

The custom authentication scheme module must add dependencies on both Core REST Security and Spring Security. Here is a code sample that shows you the minimal dependencies.

Example 8-24: Custom Authentication Scheme Pom

```

<dependency>
    <groupId>com.emc.documentum.rest</groupId>
    <artifactId>documentum-rest-config</artifactId>
</dependency>
<dependency>
    <groupId>com.emc.documentum.rest</groupId>
    <artifactId>documentum-rest-security-core</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-core</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-web</artifactId>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <scope>provided</scope>
</dependency>

```



acme-rest-web

It is the existing Maven sub module in the Archetype project. Its purpose is to overwrite the WAR packaging and settings in the extended Foundation REST API. So after resources files are put into the module location, the files overwrite the default files in the same location within the WAR file.

rest-api-runtime.properties

Follow the instructions in “[Configure runtime properties](#)” on page 337 to set the runtime properties for the custom authentication scheme. During the build of the web module, this file overwrites the default (empty) REST Runtime Properties file in the WAR file.

➡ Example 8-25: Runtime Properties for Custom Authentication

```
# set authentication mode (required)
rest.security.auth.mode=ct-custom

# set realm name
rest.security.realm.name=ACME.COM

# set Java configuration package (required)
rest.context.config.location=com.acme.rest.security.config
```



pom.xml (web)

The web module must adds dependency for the module acme-rest-custom-security and the custom authentication jar file packages everything into the WAR file. Here is a code sample that shows you the pom file:

➡ Example 8-26: Web module POM

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                             http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <parent>
    <artifactId>acme-rest</artifactId>
    <groupId>com.acme.rest</groupId>
    <version>1.0.0-SNAPSHOT</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>
  <artifactId>acme-rest-web</artifactId>
  <packaging>war</packaging>
  <build>
    <resources>
      <resource>
        <directory>src/main/resources</directory>
        <includes>
          <include>*.properties</include>
        </includes>
        <filtering>true</filtering>
      </resource>
```

```

        </resources>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-war-plugin</artifactId>
            <version>${org.apache.maven.plugins.war.version}</version>
            <configuration>
                <overlays>
                    <overlay>
                        <groupId>com.emc.documentum.rest</groupId>
                        <artifactId>documentum-rest-web</artifactId>
                        <excludes />
                    </overlay>
                </overlays>
            </configuration>
        </plugin>
    </plugins>
</build>

<dependencies>
    <dependency>
        <groupId>com.acme.rest</groupId>
        <artifactId>acme-rest-custom-security</artifactId>
        <version>1.0.0-SNAPSHOT</version>
    </dependency>
    <dependency>
        <groupId>com.emc.documentum.rest</groupId>
        <artifactId>documentum-rest-web</artifactId>
        <version>${com.emc.documentum.rest.version}</version>
        <type>war</type>
        <scope>runtime</scope>
    </dependency>
    <!-- other new dependencies -->
</dependencies>
</project>

```



pom.xml (root)

This is the Archetype project's root module. It must add acme-rest-custom-security as a sub module. Here is a code sample that shows you the root pom file:

➡ Example 8-27: The root POM

```

<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>acme-rest</artifactId>
        <groupId>com.acme.rest</groupId>
        <version>1.0.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <packaging>pom</packaging>
    <modules>
        <module>acme-rest-custom-security</module>
        <!-- other sub modules -->
    </modules>
</project>

```



Chapter 9

Advanced security configuration

This section contains configurations for advanced security configuration items and features, such as *Default Security Headers*, *Cross Site Request Forgery (CSRF) protection with Client token*, *Request Sanitization*, and *Cross-Origin Resource Sharing or (CORS)*.

9.1 Default security Headers

Foundation REST API version 7.3 and later provides the following configurable security defaults to help protect your Foundation REST API:

Vulnerability	Solution	Runtime properties	Defaults
Strict Transport Security Misconfiguration	Add the Strict Transport Security Response Header. RFC 6797 provides more details.	<p>Specifies whether to disable the Strict-Transport-Security header. When false, the HTTP Strict Transport Security (HSTS) is enabled when the network protocol is HTTPS and the Response adds the following header:</p> <ul style="list-style-type: none"> - Strict-Transport-Security: max-age=31536000 ; includeSubDomains <p>For more information, see IETF Tools website.</p> <p>Defaults to false.</p> <pre>rest.security.headers.hsts.disabled=</pre> <p>Specifies whether to include sub domains for the Strict-Transport-Security header. When true, sub domains should be considered HSTS Hosts. For more information, see IETF Tools website for defaults to true.</p> <pre>rest.security.headers.hsts.include_sub_domains=</pre> <p>Specifies the value (in seconds) for the max-age directive of the Strict-Transport-Security header. For more information, see IETF Tools website.</p> <p>Defaults to one year.</p> <pre>rest.security.headers.hsts.max_age_in_seconds=</pre>	<pre>Strict -Transport-Security: max-age= 31536000 ; includeSub Domains</pre>

Vulnerability	Solution	Runtime properties	Defaults
Cacheable HTTPS Response	Add <i>Cache-Control</i> , <i>Pragma</i> , <i>Expires</i> Header	Specifies whether to disable the Cache-Control header. When false, the Response adds the following headers: - Cache-Control: no-cache, no-store, max-age=0, must-revalidate - Pragma: no-cache - Expires: 0 Defaults to false. rest.security.headers.cache_control.disabled=	Cache-Control: no-cache, no-store, max-age=0, must-revalidate Pragma: no-cache Expires: 0
Content Sniffing Not Disabled	Add Content-Type-Options Header	Specifies whether to disable the X-Content-Type-Options header. When false, the Response adds the following header: - X-Content-Type-Options: nosniff For more information, see MSDN on Microsoft website. Defaults to false. rest.security.headers.content_type_options.disabled=	X-Content-Type-Options: nosniff

Vulnerability	Solution	Runtime properties	Defaults
Missing Cross-Frame Scripting Protection	Add X-Frame-Options Header	Specifies whether to disable the X-Frame-Options header. When false, the Response adds the following headers: - X-Frame-Options: DENY For more information, see Mozilla website. Defaults to false. rest.security.headers.x_frame_options.disabled= Specifies the X-Frames-Options policy. Allowed values are DENY and SAMEORIGIN. Defaults to DENY. rest.security.headers.x_frame_options.policy=	X-Frame-Options: DENY

Vulnerability	Solution	Runtime properties	Defaults
Browser Cross-Site Scripting Filter Misconfiguration	Add X-XSS-Protection Header	<p>Specifies whether to disable the X-XSS-Protection header. When false, the Response adds the following header: - X-XSS-Protection: 1; mode=block For more information, see MSDN blog Defaults to false.</p> <p>rest.security.headers.xss_protection.disabled= Specifies whether to enable X-XSS-Protection explicitly. When true, the Response adds the following header: - X-XSS-Protection: 1</p> <p>When false, the Response adds the following header: - X-XSS-Protection: 0</p> <p>Defaults to true.</p> <p>rest.security.headers.xss_protection.explicit_enable= Specifies whether to set block mode for the X-XSS-Protection header. When true, the Response adds the block mode to the X-XSS-Protection header: - X-XSS-Protection: 1; mode=block When false, the Response does not add a mode. Defaults to true.</p> <p>rest.security.headers.xss_protection.block=</p>	x-xss-protection: 1; mode=block

9.2 CSRF Protection

Cross Site Request Forgery (CSRF) is an attack by a malicious Web site, email, blog, instant message, or application. The attacker uses a visitor's Web browser and authentication credentials to perform an unwanted action on a trusted site that the user is visiting.

Foundation REST API supports CSRF protection using a Client Token cookie for ticked sign on, and this functionality can be integrated with OTDS or custom SSO.

The CSRF token is used to protect the DOCUMENTUM-CLIENT-TOKEN. The client must provide the token in a request, and the server validates the token provided. When validation fails, the server returns a response with a 403 (Permission Denied) HTTP status code. This can still occur even when the initial authentication succeeds.

9.2.1 Enabling CSRF Protection

When CSRF protection is enabled, the server requires the Client to send a Client Token cookie together with the CSRF token to validate a user login. Since a CSRF attack is considered to be a severe security issue, Foundation REST API since 7.3 enables CSRF by default. This means that a Foundation REST API client that authenticates with Client Token cookies against older Foundation REST API servers does not work against a new Foundation REST API server, unless the Foundation REST API client implements CSRF protection. Foundation REST API runtime properties provide a configuration parameter that you can use to enable or disable CSRF protection. Here a code sample showing you how to disable CSRF protection in the `rest-api-runtime.properties` file:

```
// Disable CSRF protection for Client Token cookie.  
// Its default value is true.  
rest.security.csrf.enabled=false
```

9.2.2 Generating a Token

Foundation REST API uses the *Synchronizer Token Pattern* to protect the Client Token cookie. A CSRF token must be generated either by the Client or by the Server. Foundation REST API runtime properties provides two ways to generate a token.

1. Using the Client: In authentication schemes that use a Client Token cookie, the Client sends a defined CSRF token and parameter names to the Foundation REST API server during the initial authentication (Basic, SSO, and so on) phase. The Foundation REST API server sends back a Client Token cookie that all future requests, that authenticate with Client Token cookie, must use in their Request Header or Request query parameter.
2. Using the Server: In the authentication schemes that use a Client Token cookie, when clients pass the initial authentication (Basic, SSO, and so on) phase, the Foundation REST API server sends back a Client Token cookie together with a CSRF token in the Response Headers. All subsequent requests that authenticate with a Client Token cookie must provide the CSRF token together with the

Client Token cookie, either in the Request Header or in the Request query parameter

9.2.2.1 Generating a Client-side Token

9.2.2.1.1 Initial Authentication

In client-side CSRF token generation, the client must send a CSRF token and parameters in the initial authentication phase. Here's some code sample that shows a client sign on and server Response:

```
// Client sign on
GET /dctm-rest/repositories/REPO HTTP/1.1
Host: localhost:8080
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.3.1 (java 1.5)
Authorization: Basic ZG1hZG1pbjpwYXNzd29yZA==
DOCUMENTUM-CSRF-HEADER-NAME: {csrfHeaderName}
DOCUMENTUM-CSRF-QUERY-NAME: {csrfQueryName}
DOCUMENTUM-CSRF-TOKEN: {csrfTokenValue}

// Server Response
HTTP/1.1 200 OK
Content-Type: application/vnd.emc.documentum+json
Set-Cookie: DOCUMENTUM-CLIENT-TOKEN="ACa/Ou...W58="; ...; HttpOnly
/...
```

Note the following items in the Response Headers:

- *DOCUMENTUM-CSRF-HEADER-NAME*: This Header tells the client in which Request Header it should pass to the CSRF token.
- *DOCUMENTUM-CSRF-QUERY-NAME*: This Header tells the client which Request query parameter contains the CSRF token.
- *DOCUMENTUM-CSRF-TOKEN*: This Header specifies the CSRF token and is mandatory.

The Request must send the CSRF token together with the Client Token cookie for authentication. When both Header and query are specified, the client can choose either the Header or the query to carry the CSRF token. More information is available in the Token Validation section.

9.2.2.1.2 Runtime Configuration

Here's a code sample that shows you the token generation method:

```
# Specifies the token generation method; defaults to 'server'
rest.security.csrf.generation.method=client
```

9.2.2.1.3 Errors for Incomplete Authentication

When the CSRF token is generated by the client, the initial authentication Request must not provide CSRF token and parameters. When the authentication Request does not provide the CSRF token or one of the CSRF parameter, the server fails to authenticate with the following Response status:

- Bad Request (400)
 - E_LACK_OF_CSRF_TOKEN
 - E_LACK_OF_CSRF_KEY_DEFINITION

9.2.2.2 Generating a Server-side Token

9.2.2.2.1 Initial Authentication

Generating a server-side token is the default CSRF token generation method. The initial authentication request remains the same as before, but the Response has a CSRF token and parameters send back to the client.

```
// Client sign on
GET /dctm-rest/repositories/REPO HTTP/1.1
Host: localhost:8080
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.3.1 (java 1.5)
Authorization: Basic ZG1hZG1pbjpwYXNzd29yZA==

// Server Response
HTTP/1.1 200 OK
Content-Type: application/vnd.emc.documentum+json
Set-Cookie: DOCUMENTUM-CLIENT-TOKEN="ACa/Ou...W58="; ...; HttpOnly
DOCUMENTUM-CSRF-HEADER-NAME: {csrfHeaderName}
DOCUMENTUM-CSRF-QUERY-NAME: {csrfQueryName}
DOCUMENTUM-CSRF-TOKEN: {csrfTokenValue}

// ...
```

Note the following items in the Response Headers:

- *Set-Cookie*: This is the DOCUMENTUM-CLIENT-TOKEN cookie.
- *DOCUMENTUM-CSRF-HEADER-NAME*: This Header tells the client in which Request Header it should pass to the CSRF token.
- *DOCUMENTUM-CSRF-QUERY-NAME*: This Header tells the client which Request query parameter contains the CSRF token.
- *DOCUMENTUM-CSRF-TOKEN*: This Header specifies the CSRF token and is mandatory.

9.2.2.2.2 Token Lifetime

When the CSRF token is generated on the server side, the CSRF token and the Client Token cookie are generated at the same time, which means that the CSRF token lifetime is same as the Client Token lifetime. Each time a new Client Token cookie is sent back to the client, the client must reset the CSRF token by parsing the Response Header.

9.2.2.3 Runtime Configuration

Foundation REST API runtime properties provides configurations for server side token generation. Here's a code sample that shows the runtime properties:

```
# Specifies the token generation method; defaults to 'server'
rest.security.csrf.generation.method=

# Specifies {csrfHeaderName} for 'server' token generation; defaults to 'DOCUMENTUM-CSRF-TOKEN'
rest.security.csrf.header_name=

# Specifies {csrfQueryName} for 'server' token generation; defaults to 'csrf-token'
rest.security.csrf.parameter_name=

# Specifies the 'server' generated CSRF token length in bits; defaults to 256
rest.security.csrf.token.length=
```

The Foundation REST API server uses the same random algorithm as the Client Token to produce the CSRF token for server-side CSRF token generation. It can be customized using the following runtime property, which affect the Client Token encryption.

```
rest.security.random.algorithm=
```

9.2.3 Token Validation

Here's a code sample that shows the CSRF protection token in the Request Headers:

Example 9-1: A token Request in a request Header

```
// Client request with the CSRF protection token in the Request headers. This sample
// assumes that 'DOCUMENTUM-CSRF-TOKEN' is set as the CSRF token header name
GET /dotm-rest/repositories/REPO HTTP/1.1
Host: localhost:8080
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.3.1 (java 1.5)
Cookie: DOCUMENTUM-CLIENT-TOKEN="ACa/Ou...W58="
DOCUMENTUM-CSRF-TOKEN: {csrfTokenValue}

// Server Response
HTTP/1.1 200 OK
Content-Type: application/vnd.emc.documentum+json

/...
```

Here's a code sample that shows a client request with CSRF protection in its query parameter:

 **Example 9-2: A token request in a query parameter**

```
// Client request with the CSRF protection token in the query parameters
// this sample assumes 'csrf-token' is set as the CSRF token query parameter name
GET /dctm-rest/repositories/REPO?csrf-token={urlEncodedCsRfTokenValue} HTTP/1.1
Host: localhost:8080
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.3.1 (java 1.5)
Cookie: DOCUMENTUM-CLIENT-TOKEN="ACa/Ou...W58="

// Server Response
HTTP/1.1 200 OK
Content-Type: application/vnd.emc.documentum+json

/...
```



9.2.3.1 Errors for Validation

When the CSRF token is not validated, the Foundation REST API server sends the HTTP 403 Response status shown here:

- Forbidden (403)
 - E_CSRF_TOKEN_NOT_VALID
 - E_CSRF_TOKEN_NOT_PROVIDED

9.2.4 HTTP Inspection Method

The Foundation REST API server can determine which HTTP methods (GET, POST, PUT, DELETE) require CSRF token validation. By default, GET does not require CSRF token validation. This can be customized using runtime properties as shown in the following code sample:

```
// Used to specify which HTTP methods required CSRF protection; defaults to
POST,PUT,DELETE.
rest.security.csrf.http_methods=
```

9.3 Cross-Origin Resource Sharing (CORS) support

The Cross-Origin Resource Sharing (CORS) specification defines a mechanism to enable client-side cross-origin requests. Specifications that enable an API to make cross-origin requests to resources can use the algorithms defined by this specification.

For example, when an API is used on `http://example.org` resources, a resource on `http://hello-world.example` can be made to use the mechanism described by the CORS specification by setting `Access-Control-Allow-Origin: http://example.org` in the Response Header. This mechanism retrieves the resource cross-origin from `http://example.org`.

9.3.1 Enable Foundation REST API server CORS support

You can enable Foundation REST API server CORS support by configuring the REST API Runtime Properties file. Set the value of the `rest.cors.enabled=true` configuration property to enable CORS support.

Example 9-3: CORS configurations

```
# Whether or not cross origin resource sharing support is enabled.  
# The default value is false.  
rest.cors.enabled=false  
  
# Specifies the origins which are allowed to be shared. The default value '*' is not  
# supported from the 22.2 release.  
# When you want to enable any origin to share, set the value to *. However, setting the  
# property value to * is allowed only if the value of the rest.cors.allow.credentials  
# property is set to false.  
# If there are more than one origin, the value should be separated by ','. For example:  
# http://opentext.com,https://test.com:8443  
rest.cors.allowed.origins=  
  
# Specifies the HTTP methods allowed to be shared. The default value '*' is not  
# supported from the 22.2 release.  
# The value should be in uppercase, and separated by ',' in case multiple methods need  
# to be added.  
# When you want to enable any HTTP method to share, set the value to *. However, setting  
# the property value to * is allowed only if the value of the rest.cors.allow.credentials  
# property is set to false.  
rest.cors.allowed.methods=  
  
# Specifies the HTTP headers allowed to be shared. The default value '*' is not  
# supported from the 22.2 release.  
# The value should be separated by ',', in case multiple headers need to be added.  
# When you want to enable any HTTP headers to share, set the value to *. However,  
# setting the property value to * is allowed only if the value of the  
# rest.cors.allow.credentials property is set to false.  
rest.cors.allowed.headers=  
  
# Whether to allow user credentials  
# The default value is true  
rest.cors.allow.credentials=true  
  
# The headers that are safe to expose to the API.  
# The values are separated by a comma.  
# Simple headers are exposed by the CORS specification, and need not be set:  
# Cache-Control, Content-Language, Content-Type, Expires, Last-Modified, Pragma.  
# The Location header is added by default.  
rest.cors.exposed.headers=Location  
  
# The amount of time (in seconds) that the results of a preflight request can be cached.  
# The value is in seconds and the default is 3600 (1 hour).  
rest.cors.max.age=3600
```



9.4 Request sanitization

Stored Cross-Site Scripting allows for the permanent injection of JavaScript code. This is a security vulnerability because this JavaScript code can result in the theft of user sessions. Request sanitization cleans up user input to secure against this vulnerability.

Request sanitization looks at two parts of the input:

- The input object metadata
- The input HTML content

9.4.1 Third-party Libraries

We used a third-party library called <OWASP AntiSamy>. The <OWASP AntiSamy> library provides the *java api*, which is used to sanitize an input string. When found, a suspicious script is removed from the input string. The library also provides policies that can be used to configure sanitization.

9.4.2 Sanitize the Input Object Metadata

Here's a sample that shows you how a malicious script might be added while creating a document:

Example 9-4: XML Script Injection Sample

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
  <properties>
    <object_name>
      The<script>alert("I shouldn't be here!")</script>Document
    </object_name>
    <r_object_type>dm_document</r_object_type>
  </properties>
</document>
```



Example 9-5: JSON Script Injection Sample

```
{
  "name" : "document",
  "type" : "dm_document",
  "properties" : {
    "object_name" : "The<script>alert('I shouldn't be here!')</script>Document",
  }
}
```



After sanitization, the metadata becomes:

▶ **Example 9-6: XML Sanitized Metadata**

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
  <properties>
    <object_name>TheDocument</object_name>
    <r_object_type>dm_document</r_object_type>
  </properties>
</document>
```



▶ **Example 9-7: JSON Sanitized Metadata**

```
{
  "name" : "document",
  "type" : "dm_document",
  "properties" : {
    "object_name" : "TheDocument",
  }
}
```



The metadata sanitization is combined with the Foundation REST API marshalling framework by Java annotations. When the input metadata is custom processed, the sanitization should be called manually.

You can use the `rest.sanitize.type.metadata` configuration setting, which is found in the runtime properties file, to control whether the metadata for all objects is sanitized.

```
# Determines whether to sanitize the metadata of an object.
# The sanitize will modify the metadata (remove the suspicious part).
# The default value is false.
#
rest.sanitize.type.metadata=false
```

When model field have special behavior against the global configuration, the `@SanitizeConfig` annotation can be used in the following two use cases:

- When metadata is supposed to be saved with XML, or a legitimate script, and so on. After the metadata has been sanitized the script and other XML elements may be inadvertently removed. To avoid the removal of legitimate content, set the annotation `@SanitizeConfig` with `<skipSanitize=true>`. This value causes the annotated field to be skipped during the sanitization process while deserializing.

Here's an example that shows you how to use the `@SanitizeConfig` annotation:

```
@SerializableType(value...)
public class MyModel {
  @SerializableField
  @SanitizeConfig(skipSanitize=true)
  private String notSanitized;
  ...
}
```

- When a specific field always requires sanitizing in spite of the global configuration, that field must be annotated with the `@SanitizeConfig` annotation

with the value `<enforceSanitize=true>`. Regardless of the configuration, that field is always sanitized.

Here's an example that shows you how to use this annotation:

```
@SerializableType(value...)
public class MyModel {
    @SerializableField
    @SanitizeConfig(enforceSanitize=true)
    private String mustBeSanitized;
    ...
}
```

9.4.3 Sanitize the Input HTML content

When performing the following operations, the content may be sanitized according the configurations:

- While importing the content (folder child objects/documents resources)
- While creating the primary content/renditions (contents resource)
- While checking in the content (versions resource)

You can configure whether to sanitize content, and for what kind of content format (`<dm_format>`) . By default, only HTML and `<put_html>` are sanitized.

To sanitize content, it must be loaded into memory, and converted into a readable string. When loaded into memory, there is one configuration to set the maximum content size to be loaded. When the content is larger than the set size, the content is not sanitized.



Note: When converting the content into a readable string, the charset of the content is required.

There is one new query parameter called `content-charset` that is added to the resources mentioned. This parameter tells the server how to parse the content. When the content-charset is not provided, the Foundation REST API server tries to get metadata information from the content itself.

When the metadata has a valid charset, is used to sanitize the content. When neither content-charset parameter, nor metadata information can be found, the configuration default charset is used.



Note: When the charset and the content do not match, then the uploaded content may have incorrect encoded values.

9.4.4 Configurations

```
# Determines whether to sanitize the metadata of an object.  
# The sanitize will modify the metadata (remove the suspicious part).  
# The default value is false.  
rest.sanitize.type.metadata=false  
  
# Determines whether to sanitize the content of an html.  
# The sanitize will modify the content (remove the suspicious part).  
# The default value is false.  
rest.sanitize.type.content=false  
  
# Set the max size of the content which will be sanitized.  
# If the content size is larger than the max size, it will not be sanitized.  
# The value is in bytes.  
# The default value is 500K bytes.  
rest.sanitize.content.max.size=500000  
  
# Set the default html content charset/encoding.  
# When sanitizing the input html content, the charset is determined by the following  
sequence:  
# 1. the Request parameter content-charset  
# 2. try to get the charset from html meta info  
# 3. both 1 and 2 failed, then use this default charset to sanitize the html content  
rest.sanitize.content.default.charset=UTF-8  
  
# Specifies the content formats which will be sanitized.  
# Only the formats in this list will be sanitized.  
# The formats in the list should be separated by ','.  
# The format names refer to dm_format.  
rest.sanitize.content.format=html,pub_html
```

9.4.5 Customize the AntiSamy Policy Files

When you want to use customized *AntiSamy* policy files instead of the default policy files, you can create policy files under the WEB-INF\classes directory. This is the same directory where the rest-api-runtime.properties file is located.

You can create a rest-antisamy-html.xml file to define the policy for sanitization of file content, and rest-antisamy-string.xml to customize the object metadata sanitization policy.

There are two sample policy files provided:

- rest-antisamy-html.sample.xml
- rest-antisamy-string.sample.xml

You can create one or both of these files to customize your sanitization policy. When a custom policy file is provided, the system default policy file that would normally be used is ignored.

Chapter 10

Explore Foundation REST API

This section provides a sample that guides you through some of the basic concepts and tasks that you can do in Foundation REST API. The sample focuses on common tasks involving folders, documents, and contents. By exploring the sample, you will become familiar with the following tasks:

- Navigating to a repository from the service node
- Navigating to a cabinet in the repository
- Using the pagination feature in a folder
- Creating a document under a folder
- Adding content to a document
- Deleting a document
- Creating a document with content with a single Request

Notes

- This sample uses the JSON representation.
- Foundation REST API is assumed to be deployed on `localhost:8080`.

10.1 Prerequisites

- The web browser must be able to render the JSON representation automatically.
- The web browser must have a Foundation REST API client plug-in installed.

10.2 Common tasks on folders, documents, and contents

Follow these steps after you deploy Foundation REST API:

1. Type the following URL in your web browser to navigate to the service node (Home Document).

```
http://localhost:8080/dctm-rest/services.json
```

The service node contains a list of the available services.

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8

{
  "resources":
  {
    "http://identifiers.emc.com/linkrel/repositories": {
```

```

        "href": "http://localhost:8080/dctm-rest/repositories",
        "hints": {
            "allow": [
                "GET"
            ],
            "representations": [
                "application/xml",
                "application/json",
                "application/atom+xml",
                "application/vnd.emc.documentum+json"
            ]
        }
    },
    "about": {
        "href": "http://localhost:8080/dctm-rest/product-information",
        "hints": {
            "allow": ["GET"],
            "representations": [
                "application/xml",
                "application/json",
                "application/vnd.emc.documentum+xml",
                "application/vnd.emc.documentum+json"
            ]
        }
    }
}
}

```

Typically, you will see the resources service as the first node of services. In the link relation `http://identifiers.emc.com/linkrel/repositories`, note the URI to the repositories resource that resembles the following:

`http://identifiers.emc.com/linkrel/repositories`

2. Click the repositories link you got from step 1 to navigate to the list of all available repositories. Explore the output, and note the information in the `entries` element.

```

HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8

{
    "id": "http://localhost:8080/dctm-rest/repositories",
    "title": "Repositories",
    "updated": "2013-05-22T14:41:29.672+08:00",
    "author": [
        {
            "name": "OpenText Documentum"
        }
    ],
    "total": 2,
    "links": [
        {
            "rel": "self",
            "href": "http://localhost:8080/dctm-rest/repositories"
        }
    ],
    "entries": [
        {
            "id": "http://localhost:8080/dctm-rest/repositories/acme01",
            "title": "acme01",
            "content": {
                "content-type": "application/json",
                "src": "http://localhost:8080/dctm-rest/repositories/acme01"
            }
        }
    ]
}

```

```

        "links":
        [
            {
                "rel": "edit",
                "href": "http://localhost:8080/dctm-rest/repositories/acme01"
            }
        ]
    },
    {
        "id": "http://localhost:8080/dctm-rest/repositories/acme02",
        "title": "acme02",
        "content":
        {
            "content-type": "application/json",
            "src": "http://localhost:8080/dctm-rest/repositories/acme02"
        },
        "links":
        [
            {
                "rel": "edit",
                "href": "http://localhost:8080/dctm-rest/repositories/acme02"
            }
        ]
    }
]
}

```

- Click the href link of the edit link relation of a repository in the entries element to retrieve the details of the repository. Enter your credentials if you are prompted for authentication.

```
GET http://localhost:8080/dctm-rest/repositories/acme01
Authorization: Basic ZG1hZG1pbjpwYXNzd29yZ
```

You will get an output that resembles the following:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8

{
    "id": 1234,
    "name": "acme01",
    "servers": [
        {
            "name": "acme01",
            "host": "CS70_Main",
            "version": "7.2.0000.0000Win64.SQLServer",
            "docbroker": "CS70_Main"
        }
    ],
    "links": [
        {
            "rel": "self",
            "href": "http://localhost:8080/dctm-rest/repositories/acme01"
        },
        {
            "rel": "http://identifiers.emc.com/linkrel/users",
            "href": "http://localhost:8080/dctm-rest/repositories/acme01/users"
        },
        {
            "rel": "http://identifiers.emc.com/linkrel/current-user",
            "href": "http://localhost:8080/dctm-rest/repositories/acme01/currentuser"
        },
        {
            "rel": "http://identifiers.emc.com/linkrel/groups",
            "href": "http://localhost:8080/dctm-rest/repositories/acme01/groups"
        },
        {
            "rel": "http://identifiers.emc.com/linkrel/cabinets",
            "href": "http://localhost:8080/dctm-rest/repositories/acme01/cabinets"
        }
    ]
}
```

```

"http://localhost:8080/dctm-rest/repositories/acme01/cabinets",
{
  "rel": "http://identifiers.emc.com/linkrel/formats",
  "href":
    "http://localhost:8080/dctm-rest/repositories/acme01/formats",
  {
    "rel": "http://identifiers.emc.com/linkrel/network-locations",
    "href":
      "http://localhost:8080/dctm-rest/repositories/acme01/
network-locations",
    {
      "rel": "http://identifiers.emc.com/linkrel/relations",
      "href":
        "http://localhost:8080/dctm-rest/repositories/acme01/relations",
      {
        "rel": "http://identifiers.emc.com/linkrel/relation-types",
        "href":
          "http://localhost:8080/dctm-rest/repositories/acme01/relation-types",
        {
          "rel": "http://identifiers.emc.com/linkrel/checked-out-objects",
          "href":
            "http://localhost:8080/dctm-rest/repositories/acme01/
checked-out-objects",
          {
            "rel": "http://identifiers.emc.com/linkrel/types",
            "href":
              "http://localhost:8080/dctm-rest/repositories/acme01/types",
            {
              "rel": "http://identifiers.emc.com/linkrel/dql",
              "hreftemplate":
                "http://localhost:8080/dctm-rest/repositories/acme01
{?dql,page,items-per-page}"
            }
          }
        }
      }
    }
}

```

By clicking the link relations in the `links` element, you can drill down various resources in the repository.

4. By clicking the link relation `http://identifiers.emc.com/linkrel/cabinets` in the `links` element, you can navigate to the list of all available cabinets. Explore the output, and note the information in the `entries` element.



Note: You can set the `inline` parameter to `true`, and this retrieves the entire content of each entry in the collection. In the following output, the `inline` parameter uses the default value `false` so that content of an entry only contains `content-type` and `src`.

```

HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8

{
  "id": "http://localhost:8080/dctm-rest/repositories/acme01/cabinets",
  "title": "Cabinets",
  "updated": "2013-05-22T14:55:24.594+08:00",
  "author": [
    {
      "name": "OpenText Documentum"
    },
    "links": [
      {
        "rel": "self",
        "href": "http://localhost:8080/dctm-rest/repositories/acme01/cabinets"
      },
      "entries": [
        {

```

```
"id": "http://localhost:8080/dctm-rest/repositories/acme01/objects/0c0004d280000d1f",
"title": "dmadmin",
"updated": "2012-10-15T23:31:27.000+08:00",
"author": [
{
"name": "dmadmin",
"uri": "http://localhost:8080/dctm-rest/repositories/acme01/users/dmadmin"
},
{
"content": {
"content-type": "application/json",
"src": "http://localhost:8080/dctm-rest/repositories/acme01/cabinets/0c0004d280000d1f"
},
"links": [
{
"rel": "edit",
"href": "http://localhost:8080/dctm-rest/repositories/acme01/cabinets/0c0004d280000d1f"
}
],
{
"id": "http://localhost:8080/dctm-rest/repositories/acme01/objects/0c0004d280000107",
"title": "Temp",
"updated": "2012-10-15T15:27:31.000+08:00",
"author": [
{
"name": "acme01",
"uri": "http://localhost:8080/dctm-rest/repositories/acme01/users/acme01"
},
{
"content": {
"content-type": "application/json",
"src": "http://localhost:8080/dctm-rest/repositories/acme01/cabinets/0c0004d280000107"
},
"links": [
{
"rel": "edit",
"href": "http://localhost:8080/dctm-rest/repositories/acme01/cabinets/0c0004d280000107"
}
]
}
]
}
]
```

- Click the `src` link in the `content` element of a cabinet in the `entries` element to retrieve the details of the cabinet. You will get an output that resembles the following:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8

{
    "name": "object",
    "type": "dm_cabinet",
    "definition": "http://localhost:8080/dctm-rest/repositories/acme01/types/dm_cabinet",
    "properties": {
        "r_object_id": "0c0004d280000107",
        "object_name": "Temp",
        "r_object_type": "dm_cabinet",
        "title": "Temporary Object Cabinet",
        ...
    }
}
```

```

},
  "links":
[
  {
    "rel": "self",
    "href": "http://localhost:8080/dctm-rest/repositories/acme01/objects/0c0004d280000107"
  },
  {
    "rel": "edit",
    "href": "http://localhost:8080/dctm-rest/repositories/acme01/objects/0c0004d280000107"
  },
  {
    "rel": "http://identifiers.emc.com/linkrel/delete",
    "href": "http://localhost:8080/dctm-rest/repositories/acme01/objects/0c0004d280000107"
  },
  {
    "rel": "canonical",
    "href": "http://localhost:8080/dctm-rest/repositories/acme01/cabinets/0c0004d280000107"
  },
  {
    "rel": "http://identifiers.emc.com/linkrel/folders",
    "href": "http://localhost:8080/dctm-rest/repositories/acme01/folders/0c0004d280000107/folders"
  },
  {
    "rel": "http://identifiers.emc.com/linkrel/documents",
    "href": "http://localhost:8080/dctm-rest/repositories/acme01/folders/0c0004d280000107/documents"
  },
  {
    "rel": "http://identifiers.emc.com/linkrel/objects",
    "href": "http://localhost:8080/dctm-rest/repositories/acme01/folders/0c0004d280000107/objects"
  },
  {
    "rel": "http://identifiers.emc.com/linkrel/child-links",
    "href": "http://localhost:8080/dctm-rest/repositories/acme01/folders/0c0004d280000107/child-links"
  },
  {
    "rel": "http://identifiers.emc.com/linkrel/relations",
    "href": "http://localhost:8080/dctm-rest/repositories/acme01/relations?related-object-id=0c0004d280000107&related-object-role=any"
  }
]
}

```

By clicking the link relations in the `links` element, you can drill down various resources in the cabinet.

- Click the `href` link of link relation `http://identifiers.emc.com/linkrel/folders` of the cabinet to retrieve the details of the child folders under the cabinet. You will get an output that resembles the following:

```

HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8

{
  "id": "http://localhost:8080/dctm-rest/repositories/acme01/folders/0c0004d280000107/folders",
  "title": "Folders under folder 0c0004d280000107",
  "updated": "2013-05-22T15:07:54.156+08:00",
  "author": [{"name": "OpenText Documentum"}],
  "page": 1,
  "items-per-page": 100,

```

```

"links":
[
  {
    "rel": "self",
    "href": "http://localhost:8080/dctm-rest/repositories/acme01/folders/
0c0004d28000107/folders"
  },
  {
    "rel": "next",
    "href": "http://localhost:8080/dctm-rest/repositories/acme01/folders/
0c0004d28000107/folders?items-per-page=100&page=2"
  },
  {
    "rel": "first",
    "href": "http://localhost:8080/dctm-rest/repositories/acme01/folders/
0c0004d28000107/folders?items-per-page=100&page=1"
  }
],
"entries":
[
  {
    "id": "http://localhost:8080/dctm-rest/repositories/acme01/objects/
0b0004d280009646",
    "title": "REST-API-TEST-FOLDER43fd1f60-fc9a-4402-9aca-30d86ab9f4b4",
    "updated": "2013-05-22T15:07:39.000+08:00",
    "author": [
      {
        "name": "dave", "uri": "http://localhost:8080/dctm-rest/repositories/
acme01/users/dave"
      }
    ],
    "content": {
      "content-type": "application/json",
      "src": "http://localhost:8080/dctm-rest/repositories/acme01/folders/
0b0004d280009646"
    },
    "links": [
      {
        "rel": "edit", "href": "http://localhost:8080/dctm-rest/
repositories/acme01/folders/0b0004d280009646"
      }
    ]
  },
  {
    "id": "http://localhost:8080/dctm-rest/repositories/acme01/objects/
0b0004d280009647",
    "title": "REST-API-TEST-FOLDERRa0cb9000-2b85-47ea-a427-9108a43c5097",
    "updated": "2013-05-22T15:07:39.000+08:00",
    "author": [
      {
        "name": "dave", "uri": "http://localhost:8080/dctm-rest/
repositories/acme01/users/dave"
      }
    ],
    "content": {
      "content-type": "application/json",
      "src": "http://localhost:8080/dctm-rest/repositories/acme01/folders/
0b0004d280009647"
    },
    "links": [
      {
        "rel": "edit", "href": "http://localhost:8080/dctm-rest/
repositories/acme01/folders/0b0004d280009647"
      }
    ]
  }
]

```

```

        },
        ...
    ]
}
```

Click the `href` link in the next link relation to navigate to the next page.

7. Click the `href` link of the `edit` link relation of a folder in the `entries` element to retrieve the details of the folder.

```

HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8

{
    "name": "folder",
    "type": "dm_folder",
    "definition": "http://localhost:8080/dctm-rest/repositories/acme01/types/
                  dm_folder",
    "properties": {
        ...
        "r_object_id": "0b0004d280009646",
        "object_name": "REST-API-TEST-FOLDER43fd1f60-fc9a-4402-9aca-30d86ab9f4b4",
        "r_object_type": "dm_folder",
        ...
    },
    "links": [
        {
            "rel": "self",
            "href": "http://localhost:8080/dctm-rest/repositories/acme01/folders/
                    0b0004d280009646"
        },
        {
            "rel": "edit",
            "href": "http://localhost:8080/dctm-rest/repositories/acme01/folders/
                    0b0004d280009646"
        },
        {
            "rel": "http://identifiers.emc.com/linkrel/delete",
            "href": "http://localhost:8080/dctm-rest/repositories/acme01/folders/
                    0b0004d280009646"
        },
        {
            "rel": "http://identifiers.emc.com/linkrel/parent-links",
            "href": "http://localhost:8080/dctm-rest/repositories/acme01/objects/
                    0b0004d280009646/parent-links"
        },
        {
            "rel": "parent",
            "title": "0c0004d280000107",
            "href": "http://localhost:8080/dctm-rest/repositories/acme01/folders/
                    0c0004d280000107"
        },
        {
            "rel": "http://identifiers.emc.com/linkrel/folders",
            "href": "http://localhost:8080/dctm-rest/repositories/acme01/folders/
                    0b0004d280009646/folders"
        },
        {
            "rel": "http://identifiers.emc.com/linkrel/documents",
            "href": "http://localhost:8080/dctm-rest/repositories/acme01/folders/
                    0b0004d280009646/documents"
        },
        {
            "rel": "http://identifiers.emc.com/linkrel/objects",
            "href": "http://localhost:8080/dctm-rest/repositories/acme01/folders/
                    0b0004d280009646/objects"
        },
        {

```

```

        "rel": "http://identifiers.emc.com/linkrel/child-links",
        "href": "http://localhost:8080/dctm-rest/repositories/acme01/folders/
Ob0004d280009646/child-links"
    },
    {
        "rel": "http://identifiers.emc.com/linkrel/cabinet",
        "href": "http://localhost:8080/dctm-rest/repositories/acme01/cabinets/
0c0004d28000107"
    },
    {
        "rel": "http://identifiers.emc.com/linkrel/relations",
        "href": "http://localhost:8080/dctm-rest/repositories/acme01/
relations?related-object-id=Ob0004d280009646&
related-object-role=any"
    }
]
}

```

In the output you received from [step 7](#), click the `href` link in the `http://identifiers.emc.com/linkrel/documents` link relation to navigate to the list of available documents in this folder. You will notice that the structure of the Documents resource is similar to that of the Folders resource.

```

HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8

{
    "id": "http://localhost:8080/dctm-rest/repositories/acme01/folders/
Ob0004d280009646/documents",
    "title": "Documents under folder Ob0004d280009646",
    "updated": "2013-05-22T15:18:41.844+08:00",
    "author": [
        {
            "name": "OpenText Documentum"
        }
    ],
    "links": [
        {
            "rel": "self",
            "href": "http://localhost:8080/dctm-rest/repositories/acme01/folders/
Ob0004d280009646/documents"
        }
    ],
    "entries": [
        {
            "id": "http://localhost:8080/dctm-rest/repositories/acme01/objects/
090004d280009651",
            "title": "REST-API-TEST-DOC",
            "updated": "2013-05-22T15:07:39.000+08:00",
            "author": [
                {
                    "name": "dave",
                    "uri": "http://localhost:8080/dctm-rest/repositories/acme01/
users/dave"
                }
            ],
            "content": {
                "content-type": "application/json",
                "src": "http://localhost:8080/dctm-rest/repositories/acme01/documents/
090004d280009651"
            },
            "links": [
                {
                    "rel": "edit",
                    "href": "http://localhost:8080/dctm-rest/repositories/acme01/
documents/
090004d280009651"
                }
            ]
        }
    ]
}

```

8. Send a POST Request to create a document under the folder with the following configuration:

- Set the URL to the href link in the documents link relation you got in [step 7](#).
- Set the content type to application/vnd.emc.documentum+json
- Enter the following data in the Request body:

```
{
  "properties":{
    "object_name":"Vienna",
    "r_object_type":"dm_document"
  }
}
```

- Set the method to POST, and then send the Request.



Note: The detailed steps may vary depending on the tool you use to send the Request.

You will receive an HTTP 201 Created status upon a successful document creation. Also, you will receive the URI pointing to the document in the Location Header of the Response body. Enter this URI in the web browser to navigate to the newly-created document. The output resembles the following:

```
HTTP/1.1 201 OK
Content-Type: application/json; charset=UTF-8
Location: http://localhost:8080/dctm-rest/repositories/acme01/documents/090004d280009894

{
  "name": "document",
  "type": "dm_document",
  "definition": "http://localhost:8080/dctm-rest/repositories/acme01/types/dm_document",
  "properties": {
    "r_object_id": "090004d280009894",
    "object_name": "Vienna",
    "r_object_type": "dm_document",
    ...
  },
  "links": [
    {"rel": "self",
      "href": "http://localhost:8080/dctm-rest/repositories/acme01/documents/090004d280009894"},
    {"rel": "edit",
      "href": "http://localhost:8080/dctm-rest/repositories/acme01/documents/090004d280009894"},
    {"rel": "http://identifiers.emc.com/linkrel/delete",
      "href": "http://localhost:8080/dctm-rest/repositories/acme01/documents/090004d280009894"},
    {"rel": "http://identifiers.emc.com/linkrel/parent-links",
      "href": "http://localhost:8080/dctm-rest/repositories/acme01/objects/090004d280009894/parent-links"},
    {"rel": "parent",
      "title": "0b0004d280009646",
      "href": "http://localhost:8080/dctm-rest/repositories/acme01/folders/0b0004d280009646"},
    {"rel": "http://identifiers.emc.com/linkrel/cabinet",
      "href": "http://localhost:8080/dctm-rest/repositories/acme01/cabinets/0c0004d280000107"},
    {"rel": "contents",
      "href": "http://localhost:8080/dctm-rest/repositories/acme01/objects/090004d280009894/contents"},
    {"rel": "http://identifiers.emc.com/linkrel/primary-content",
```

```

        "href": "http://localhost:8080/dctm-rest/repositories/acme01/objects/
090004d280009894/contents/content"},

        {"rel": "http://identifiers.emc.com/linkrel/checkout",
         "href": "http://localhost:8080/dctm-rest/repositories/acme01/objects/
090004d280009894/lock"},

        {"rel": "version-history",
         "href": "http://localhost:8080/dctm-rest/repositories/acme01/objects/
090004d280009894/versions"},

        {"rel": "http://identifiers.emc.com/linkrel/current-version",
         "href": "http://localhost:8080/dctm-rest/repositories/acme01/objects/
090004d280009894/versions/current"},

        {"rel": "http://identifiers.emc.com/linkrel/relations",
         "href": "http://localhost:8080/dctm-rest/repositories/acme01/
relations?related-object-id=090004d280009894&related-object-role=any"}
    ]
}

```

- Send a POST request to create a content for this document with the following configuration:
 - Set the URL to the href link in the contents link relation you got in step 8.
 - If the plug-in allows you to set the Request body from a local file, select the local Content Media that you want to import, and then set the corresponding content type.

If the plug-in does not allow you to set the Request body from a local file, input the content file binary to the Request body, and then set the corresponding value for the content type.

If the plug-in does not allow you to set the Request body from a local file, input the content file binary to the Request body, and then set the corresponding value for the content type. For example:

```

POST http://localhost:8080/dctm-rest/repositories/acme01/objects/
090004d280009894/contents
Authorization: Basic ZG1hZG1pbjpwYXNzd29yZ
Content-Type: text/plain

a test string content: hello, rester!

```

- Set the method to POST, and then send the Request.



Note: The detailed steps may vary depending on the tool you use to send the Request.

You will receive an HTTP 201 Created status upon a successful content creation.

```

HTTP/1.1 201 OK
Content-Type: application/json; charset=UTF-8
Location: http://localhost:8080/dctm-rest/repositories/acme01/objects/
090004d280009894/contents/content?format=atd&modifier=&page=0

{
  "name": "content",
  "properties": {
    "r_object_id": "060004d280004a5f",
    "rendition": 0,
    ...
  },
  "links": [
    {
      "rel": "self",
      "href": "http://localhost:8080/dctm-rest/repositories/acme01/objects/

```

```

090004d280009894/contents/content?format=atd&modifier=&page=0"
},
{
"rel": "http://identifiers.emc.com/linkrel/content-media",
"title": "ACS",
"href": "http://localhost:9080/ACS/servlet/ACS?command=read&version=2.3
&docbaseid=0004d2&basepath=%3A%5Cdocumentum%5Cdata%5Cacme01%5Content_
storage_01%5C000004d2&filepath=80%5C00%5C26%5Coa.atd&objectid=090004d280
009894&cacheid=dAAEAgA%3D%3DCiYAgA%3D%3D&format=atd&pagenum=0&signature=
QQj3oFudClohPno49lwoVFnPQihxQGNRiv0W7U%2BrMqzCD%2FngiDKM7sBKpsk4S6a%2B%2F
nBPcR6cW1qmXW2SiuP%2FeIbtI4upEs3%2B4aMZVeac9njIJ6zRosgm8yBIYAgm038KhDVOLF
1Bxrb8Wsx%2BvQMSZyZpcHmuOpovpjZHtwCiuj8%3D&servername=CS7ORC2_MAINACS1&no
de=1&timestamp=1369207791&length=38&mime_type=text%2Fplain&parallel_stream
ing=true&expire_delta=360"
},
{
"rel": "parent",
"href": "http://localhost:8080/dctm-rest/repositories/acme01/objects/
090004d280009894"
}
]
}

```

Click the href link of the link relation `http://identifiers.emc.com/linkrel/content-media` to open the imported content.



Note: The Accelerated Content Services link may have been URL encoded.

```

HTTP/1.1 201 OK
Content-Type: text/plain
Date: Wed, 22 May 2013 07:38:52 GMT
ETag: W/"38-1369207790346"
Expires: 0
Last-Modified: Wed, 22 May 2013 07:29:50 GMT

a test string content: hello, rester!

```

- Send a DELETE request to delete the document with the following configuration:

- Set the URL to the URI pointing to the document you got in step 8.
- Set the method to DELETE, and then send the Request.

```

DELETE http://localhost:8080/dctm-rest/repositories/acme01/objects/
090004d280009894
Authorization: Basic ZG1hZG1pbjpwYXNzd29yZ

```



Note: The detailed steps may vary depending on the tool you use to send the Request.

You will receive an HTTP 204 No Content status upon a successful deletion.

10.3 Create a document and content with two Requests



Tip: All of our code samples assume that you are using your own Foundation REST API client to formulate the REST Requests and consume the Responses from the Foundation REST API server.

If your Foundation REST API client does not support the use of a POST multipart HTTP Request, then you can use a two HTTP Requests to create a document and then to add content to it. Let's examine these two steps. First, you must create a contentless document under the folder:

Example 10-1: Create a contentless document

```
POST http://localhost:8080/dctm-rest/repositories/repo/folders/
0b0000018000416b/documents
Accept: application/vnd.emc.documentum+json
Content-Type: application/vnd.emc.documentum+json
{
  "properties": {
    "r_object_type": "dm_document",
    "object_name": "readmeNoContent.txt"
  }
}
```



Second, you must create the new primary content under the document contents feed. To do this, follow the contents link relation for the document resource. Here is a code sample to illustrate this point:

Example 10-2: Create primary content

```
POST http://localhost:8080/dctm-rest/repositories/repo/objects/
090000018000a091/contents
Content-Type: text/plain
Hello REST Services!!
```

The resulting Response from the Foundation REST API server returns a content metadata resource for the newly create content. Here is a code sample that shows you the Response content metadata:

```
{
  "name": "content",
  "properties": {
    "r_object_id": "0600000180015710",
    "rendition": 0,
    "parent_count": 1,
    "content_size": 15,
    "full_format": "atd",
    "format": "2700000180000215",
    "encoding": "",
    "set_time": "2018-10-22T07:57:41.000+00:00",
    "full_content_size": 15,
    ...
  },
  "links": [
    {
      "rel": "self",
      "href": "http://localhost:8080/dctm-rest/repositories/repo/objects/090000018000a091/contents/0600000180015710"
    }
  ]
}
```

```

        "rel": "self",
        "href": "http://localhost:8080/dctm-rest/repositories/repo/
            objects/090000018000a091/contents/content?
            format=atd&modifier=&page=0"
    },
    {
        "rel": "enclosure",
        "title": "ACS",
        "href": "http://localhost:9080/ACS/servlet/ACS?
            command=read&version=2.3&docbaseid=000001&
            basepath=%3A%5CDocumentum%5Cdata%5Crepo%5C
            Content_storage_01%5C00000001&filepath"
    },
    {
        "rel": "http://identifiers.emc.com/linkrel/content-media",
        "title": "ACS",
        "href": "http://localhost:9080/ACS/servlet/ACS?command=read&
            version=2.3&docbaseid=000001&
            basepath=%3A%5CDocumentum%5Cdata%5Crepo%5C
            content_storage_01%5C00000001&filepath"
    },
    {
        "rel": "parent",
        "href": "http://localhost:8080/dctm-rest/repositories/repo/
            objects/090000018000a091"
    }
]
}

```



The content metadata resource contains both metadata and the media links of the content.

In the preceding code sample, note that the format of the content is set to atd according to its mime type, which is text/plain. This is the case because of one of the mappings between mime types and formats in the Documentum CM Server format types.

When the format is not explicitly set by the Foundation REST API client, the Foundation REST API server picks up the default format for the mime type by name (in ascending order). You can customize the preferred format name in two ways:

- Append a URI query parameter, such as format=<YOUR_PREFERRED_FORMAT_NAME>, to the contents feed URI of the POST Request.
- Update the Foundation REST API server runtime properties file to specify the default mime-to-format mapping in the `dctm-rest.war/WEB-INF/classes/rest-api-runtime.properties` file. Here is a code sample that shows you how to do that:

```

rest.mime.format.mapping.enabled=true

# [repositoryName].[mime_type]=[format], support wildcard repository name

*.text/plain=text
my_repository.application/msword=msw12

```

10.3.1 Creating a rendition

In addition to primary content, you can also create renditions within the same contents feed. Here is a code sample that shows you the POST Request to do this.

► Example 10-3: POST Request to create a rendition

To create a rendition, you must explicitly specify the format name, page number, or page modifier. In addition to this, when you create a rendition, it is mandatory to set `primary=false`. Here is a code sample to illustrate the preceding items, the `primary` property is shown in bold:

```
POST http://localhost:8080/dctm-rest/repositories/repo/objects/
    090000018000a091/contents?format=crtext&page=0&primary=false
Accept: application/vnd.emc.documentum+json
Content-Type: text/plain

Hello REST rendition!
```

The Foundation REST API server returns the following rendition. Note that the property `"rendition": 2` means that this content is in fact a rendition. It is shown in bold in the following properties:

```
{
    "name": "content",
    "properties": {
        "r_object_id": "0600000180015711",
        "rendition": 2",
        "parent_count": 1,
        "content_size": 29,
        "full_format": "crtext",
        "format": "270000018000011b",
        ...
        "parent_id": [
            "090000018000a091"
        ],
        ...
    },
    "links": [
        {
            "rel": "self",
            "href": "http://localhost:8080/dctm-rest/repositories/repo/
                objects/090000018000a091/contents/content?
                format=crtext&modifier=&page=0"
        },
        {
            "rel": "enclosure",
            "title": "ACS",
            "href": "http://localhost:9080/ACS/servlet/ACS?
                command=read&version=2.3&docbaseid=000001&
                basepath=C%3A%5CDocumentum%5Cdata%5Crepo%5C
                content_storage_01%5C00000001..."
        },
        {
            "rel": "http://identifiers.emc.com/linkrel/content-media",
            "title": "ACS",
            "href": "http://localhost:9080/ACS/servlet/ACS?
                command=read&version=2.3&docbaseid=000001&basepath=C%3A%5C
                Documentum%5Cdata%5Crepo%5C
                content_storage_01%5C00000001..."
        },
        {
            "rel": "parent",
            "href": "http://localhost:8080/dctm-rest/repositories/repo/
                objects/090000018000a091"
```

}



10.4 Creating a document and content with a single Request

When your Foundation REST API client supports the use of POST multipart HTTP Requests, then use a single multipart HTTP Request to create a document and add content to it in Foundation REST API. Proceeding with this in mind, here are the steps to create a contentful document, that includes properties and primary content:

Create a contentful document

- Using your Foundation REST API client, **POST** a multipart HTTP Request to the feed to create a contentful document directly with both properties and primary content. Here is a code sample to illustrate the **POST** Request. The important parts are shown in bold.



Example 10-4: JSON Multipart HTTP POST Request

```
POST http://localhost:8080/dctm-rest/repositories/repo/folders/0b0000018000416b/documents
Accept: application/vnd.emc.documentum+json
Content-Type: multipart/form-data;
--WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="object"

{
    "properties": {
        "r_object_type": "dm_document",
        "object_name": "readme.txt"
    }
}

--WebKitFormBoundary7MA4YWxkTrZu0gW-
Content-Disposition: form-data; name="content"; filename=""
Content-Type: text/plain

the content
--WebKitFormBoundary7MA4YWxkTrZu0gW--
```

The preceding Request creates a new document resource with content. Within the new document resource you will find two important link relations:

- contents
 - <http://identifiers.emc.com/linkrel/primary-content>

The preceding two link relations, shown in bold in the following code sample, are critical because they point to the contents feed and the primary content of the newly created document.

```

HTTP 201 Created
Location: http://localhost:8080/dctm-rest/repositories/repo/documents/
090000018000a08f
Content-Type: application/vnd.emc.documentum+json; charset=UTF-8
{
  "name": "document",
  "type": "dm_document",
  ...
  "links": [
    ...
    {
      "rel": "contents",
      "href": "http://localhost:8080/dctm-rest/repositories/repo/
objects/090000018000a08f/contents"
    },
    {
      "rel": "http://identifiers.emc.com/linkrel/primary-content",
      "href": "http://localhost:8080/dctm-rest/repositories/repo/
objects/090000018000a08f/contents/content"
    },
    ...
  ]
}

```



The contentful document has now been fully created with a single multipart POST HTTP Request.

10.5 Updating content or renditions

To update an existing content or rendition, create a POST Request in the content feed with the binary of the new content or rendition. In the URI of the contents feed, you must specify the format, page number, and page modifier to update. Then you must set the `overwrite=true` parameter. Here is a code to illustrate this:

► Example 10-5: Update a piece of content or a rendition

```

POST http://localhost:8080/dctm-rest/repositories/repo/objects/
090000018000a091/contents?
  format=crtext&page=0&primary=false&overwrite=true
Accept: application/vnd.emc.documentum+json
Content-Type: text/plain

Hello REST, this is a rendition update!!

```



10.6 Deleting content or renditions

To delete an existing piece of content, such as a file, or a rendition, use the `DELETE` HTTP Request on the `self` link relation of the content or rendition resource.

Here is a code sample that shows you how to use the `DELETE` Request on the `self` link relation:

 **Example 10-6: A DELETE Request on the self link relation**

```
DELETE http://localhost:8080/dctm-rest/repositories/repo/objects/  
090000018000a091/contents/content?format=crtext&page=0
```



Chapter 11

Tutorial: Consume Foundation REST API programmatically

This tutorial provides you with information on how to navigate collection hierarchies, read documents, and add documents in Python, using the Requests library for HTTP Requests and a JSON representation of resources.



Note: Foundation REST API is programming language independent. You can use other languages to consume Foundation REST API as well. In this tutorial we happen to use JSON and Python.

11.1 Basic navigation

Foundation REST API has a rich hierarchy as follows:

- Foundation REST API starts with the service node as the root.
- The service node contains a set of repositories.
- Each repository contains a set of cabinets or folders.
- Either folders or cabinets can contain documents or sysobjects.
- A document contains metadata, and can contain a primary content and multiple renditions.

The Foundation REST API allows clients to traverse these structures using navigation or queries. The following code enables you to navigate from the service entry point to a repository named tagsalad. From the tagsalad repository, you can access the cabinet San Francisco.

```
import requests
import json
homeResource = "http://example.com/documentum-rest/services"
drel="http://identifiers.emc.com/linkrel/"
repository = "tagsalad"
cabinet = "San Francisco"
credentials = ('tagsalad', 'password')
home = requests.get(homeResource)
home.raise_for_status()
repositories_uri = home()['resources'][drel+'repositories']['href']
repository = get_repository(repositories_uri, repository)
cabinets_uri = get_link(repository, drel+'cabinets')
sanfran = get_atom_entries(cabinets_uri, 'title="San Francisco"')[0]
documents = get_link(sanfran, drel+'documents')
```

In this code sample, The function `get_link` returns a link based on a link relation. The function `get_repository` gets a repository with a given name. The function `get_atom_entries` returns Atom entries from a collection based on their properties. The rest of this section explains the code in more details, shows the JSON representation of resources, and discusses some REST design principles.

11.1.1 Service entry point

Foundation REST API is a hypermedia-driven API, with a single entry point, which is associated with the Home Document resource. You can use the Requests library to read the Home Document resource:

```
home = requests.get(homeResource)
```

By running the `home.json()` method, you get the JSON representation of the Home Document resource:

```
{
  "resources": {
    "http://identifiers.emc.com/linkrel/repositories": {
      "hints": {
        "allow": [
          "GET"
        ],
        "representations": [
          "application/xml",
          "application/atom+xml",
          "application/vnd.emc.documentum+json"
        ]
      },
      "href": "http://example.com/documentum-rest/repositories"
    },
    "about": {
      "hints": {
        "allow": [
          "GET"
        ],
        "representations": [
          "application/xml",
          "application/vnd.emc.documentum+xml",
          "application/vnd.emc.documentum+json"
        ]
      },
      "href": "http://localhost:8080/dctm-rest/product-information"
    }
  }
}
```

11.1.2 Link relations

In a hypermedia-driven Foundation REST API, the entry point for a REST service must contain links, identified by link relations that allow a client to navigate to all resources exposed by the service. The link relation is not a physical location. Instead, it is a name encoded as a URI, which identifies the purpose of a link. A link relation is associated with the `href` entry that contains the physical address of the link. For example, in the preceding JSON representation of the Home Document resource, the `http://identifiers.emc.com/linkrel/repositories` link relation identifies a URI for an Atom feed containing repository entries, and the `href` entry indicates the physical address of this Atom feed, which is `http://example.com/documentum-rest/repositories`.

In Python, if `home` contains the result of a GET request that retrieved the Home Document resource, the following expression returns the physical address of the repositories feed:

```
home()['resources']['http://identifiers.emc.com/linkrel/repositories']['href']
```

Alternatively, you can use the following code to get the physical address of the repositories feed.

```
#Retrieving a link based on a link relation
def get_link(e, linkrel, default=None):
    return [ l['href'] for l in e['links'] if l['rel'] == linkrel ][0]
repositories_uri = get_link(home, 'http://identifiers.emc.com/linkrel/repositories')
```

Code explanation:

The `get_link()` function returns the link (physical address) associated with a link relation in a resource. If the link relation is not present, an error is raised. In Python, this can be done in a single line, which uses a list comprehension to create a list that contains all link relations that match the property, and then returns the first entry (there will never be more than one entry matching a given link relation).

11.1.3 Feeds and entries

You may have followed the instructions in the Chapter “[Explore Foundation REST API](#)” on page 363 and have familiarized yourself with the JSON representation of the Repositories feed. In that sample, each entry represents one repository and only contains a small subset of the content of a repository resource. If you need to get a specified repository from the feed and return its entire content programmatically, refer to the following code as an example:

```
#Retrieving the entire content of a repository specified by title
def get_repository(repositories_uri, name):
    for repository in get_atom_entries(repositories_uri):
        if repository['name'] == name:
            return repository
    return None
def get_atom_entries(feed_uri, filter=None, default=None):
    if filter:
        params = {'inline': 'true', 'filter': filter }
    else:
        params = {'inline': 'true' }
    Response = requests.get(feed_uri, params=params, auth=credentials)
    Response.raise_for_status()
    return [ e['content'] for e in Response()['entries'] ]
repository = get_repository(repositories_uri, repository)
```

Code explanation:

The `get_repository` function calls `get_atom_entries` to return a list of the repositories contained in the feed, and then searches for a repository with the specified name and returns it.

The `get_atom_entries` function returns a list of entries contained in a feed. This function sets the `inline` parameter to `true` so that the resulting entries contain the entire resource, instead of a subset of the content.

The return expression contains the following list comprehension:

```
[ e['content'] for e in Response()['entries'] ]
```

For collections within a given repository, you can specify conditions that are used to select results by using “[Filter expression](#)” on page 37. For example, a repository contains cabinets, so we can find the San Francisco cabinet using the following code:

```
cabinet = get_atom_entries(cabinets_uri, 'title="San Francisco"')[0]
```

11.1.4 From the Home document resource to Documents

By now, you should be able to understand the code at the beginning of the “[Basic navigation](#)” on page 381 section.

Three kinds of resources are used.

- Home Document resource: Here is the code that retrieves this resource and finds the repositories URI in it:

```
homeResource = "http://example.com/documentum-rest/services"
drel="http://identifiers.emc.com/linkrel/"
home = requests.get(homeResource)
home.raise_for_status()
repositories_uri = home()['resources'][drel+'repositories']['href']
```

- EDAA feed (the JSON representation of an Atom Feed): The following code retrieves a repository from the JSON representation using the title of the corresponding entry, and then retrieves the URI of the Cabinets resource:

```
repository = get_repository(repositories_uri, repository)
cabinets_uri = get_link(repository, drel+'cabinets')
```

- Single resources (in our tutorial, the Repository resource and the Cabinet resources). The following code retrieves a cabinet from the cabinets feed, and then retrieves the URI of the documents feed from it:

```
sanfran = get_atom_entries(cabinets_uri, 'title="San Francisco"')[0]
documents = get_link(sanfran, drel+'documents')
```

11.2 Read entries

After you get a collection of documents, you can read each entry and print its properties.

If all returned documents can fit on one page (by default, 100 entries), and you only print properties that are present in the feed when `inline` is set to `false`, refer to the following code:

```
r = requests.get(documents, auth=credentials)
for e in r()['entries']:
print (e['title'])
```



Note: The code sample prints the title of the entry instead of the document title under the properties element, as the properties element is not returned when `inline` is set to `false`.

The following code sample prints two properties that are returned only when `inline` is set to `true` (`object_name` and `title`) for each document in the collection. Additionally, the sample supports pagination by using the next link relation so that the results can span multiple pages when the number of results is large.

```
documents = get_link(sanfran, drel+'documents')
while True:
```

```

Response = requests.get(documents, params='inline=true', auth=credentials)
Response.raise_for_status()
for e in Response()['entries']:
    p = e['content']['properties']
    print ( p['object_name'], ' ', p['title'])
try:
    documents = get_link(Response(), 'next')
except:
    break

```

The first part of the while loop is similar to our previous sample. After printing the items on the given page, the `get_link` function looks for the `next` link relation that contains the URI for the next page. If it does not find a `next` link relation, it knows that it has read all pages in the collection.

11.3 Filter, sort, and pagination

When searching for the San Francisco cabinet, we used a simple filter expression. Here is an example of a slightly more complex filter:

```
contains(object_name, "COFFEE") and r_modify_date >= date("2012-12-03")
```

With this expression, the Request returns resources whose `object_name` contains COFFEE. Additionally, resources that were modified earlier than 2012-12-03 are filtered out. For more examples, see “[Examples of the filter expression](#)” on page 48.

A filter always returns an entire resource. However, you can use “[Property view](#)” on page 49 to specify a set of properties that should be returned. The sort order can also be specified, as can the number of items on a page. For more information, see “[Common query parameters](#)” on page 10.

The following code sample shows how these URI parameters can be combined in the parameter list.

```

params = {
    'inline' : True,
    'sort' : 'object_name',
    'view' : 'object_name,title',
    'filter' : 'contains(object_name, "COFFEE") and r_modify_date >= date("2012-12-03")',
    'items-per-page' : 50
}
Response = requests.get(documents, params=params, auth=credentials)
Response.raise_for_status()
for e in Response()['entries']:
    print (prettyprint(e))

```

11.4 Create entries

You can create entries in a feed by using POST and setting the corresponding content type.

At least the object name and the object type must be specified. In a real application, you may need to create an object type that represents the properties of a given kind of document. To keep it simple, the following sample only sets the title property.

```
body = json.dumps(
{
    "properties" : {
        "object_name" : "Earthquake McGoons",
        "r_object_type" : "dm_document",
        "title" : "50 California Street, 94111"
    }
})
headers = { 'content-type' : 'application/vnd.emc.documentum+json' }
Response = requests.post( documents, data=body, headers=headers, auth=credentials)
```

If the POST operation succeeds, the status code is 201 CREATED, and the Response contains the newly-created document resource.

```
>>> Response = requests.post( documents, data=body, headers=headers, auth=credentials)
>>> Response.status_code
201
>>> Response.raise_for_status()
>>> Response.reason
'Created'
>>> Response()
{
    "properties" : {
        "object_name" : "Earthquake McGoons",
        "r_object_type" : "dm_document",
        "title" : "50 California Street, 94111"
    }
}
```

11.5 Update entries

You can update a resource by using POST with the URI of the resource and setting the corresponding content type.

Suppose Response.json contains a document. The following code updates the object_name property in the document.

```
document = Response()
document['properties']['object_name'] = 'Kilroy was Here!'
headers = { 'content-type' : 'application/vnd.emc.documentum+json' }
uri = get_link(document, 'edit')
Response = requests.post(uri, json.dumps(document), headers=headers, auth=credentials)
```



Note: The preceding code uses the edit link relation, which is an IANA standard link relation that allows a resource to be read, updated, or deleted.

If the POST succeeds, the status code is 200 OK, and the Response contains the updated document resource.

```
>>> Response.status_code  
200  
>>> Response.reason  
'OK'  
>>> Response.raise_for_status()  
>>> responded()  
{  
    "properties": {  
        "object_name": "Earthquake McGoon's",  
        "r_object_type": "dm_document",  
        "title": "Kilroy was Here!"  
    }  
}
```

11.6 Delete entries

You can delete a resource by using `DELETE` with the URI of the resource.

```
Response = requests.delete(get_link(document, 'edit'), auth=credentials)
```

The Response contains the HTTP status code 204 with no content.

```
>>> Response.status_code  
204  
>>> Response.reason  
'No Content'  
>>> Response.raise_for_status()  
>>>
```


Appendix A. Link relations

Foundation REST API uses the following link relations:

Table A-1: Public link relations

Link relation	Description	Specification
about	Returns product information	RFC 6903
canonical	Designates an Internationalized Resource Identifier (IRI) as preferred over resources with duplicate content.	RFC 6596
child	Points to a hierarchical child, or subobject, of the current object.	http://www.w3.org/MarkUp/draft-ietf-html-relrev-00.txt
contents	Points to the contents metadata feed for a contentful object.	REC-html401-19991224 (http://www.w3.org/TR/1999/REC-html401-19991224)
edit	Points to a resource that can be used to edit the link's context.	RFC 5023
enclosure	Identifies a related resource that is potentially large and might require special handling.	new-link-relation (http://bitworking.org/projects/atom/rfc5023.html#new-link-relation)
first	An Internationalized Resource Identifier (IRI) that refers to the furthest preceding resource in a series of resources.	RFC 5005
icon	Points to the icon for an entry, a page or a site.	rel-icon (http://www.w3.org/TR/html5/links.html#rel-icon)
last	An Internationalized Resource Identifier (IRI) that refers to the furthest following resource in a series of resources.	RFC 5005
next	Indicates that the link's context is a part of a series, and that the next in the series is the link target.	RFC 5005
parent	Refers to one of the following: <ul style="list-style-type: none">• parent type of a type• parent folder of a sysobject• parent document of a document• parent object of a relation• parent group of a group, a role, or a user	http://www.w3.org/MarkUp/draft-ietf-html-relrev-00.txt

Link relation	Description	Specification
predecessor-version	Points to a resource containing the predecessor version in the version history.	RFC 5829
previous	Points to the previous resource in an ordered series of resources.	RFC 5005
related	Points to the feed for a sysobject related with specified relation type(s).	RFC 4287
self	Conveys an identifier for the link's context.	RFC 4287
version-history	Points to a resource containing the version history for the context.	RFC 5829

Table A-2: Link relations introduced in Foundation REST API

Link relation	Description
acl	Points to the ACL resource for a specific permission set object.
acls	Points to the ACLs feed under a repository.
as-search-template	Points to the save as search template behavior for a saved search.
associations	Points to the associated Sysobjects feed for a specific ACL object.
assist-values	Points to the type assist values resource to obtain the value assistance of a type.
aspect-types	Points to the aspect types resource to show all aspect types of a repository.
aspect-type	Points to the single aspect type resource.
batches	Points to the batches resource under a repository.
batch-capabilities	Points to the batch capabilities resource under a repository.
cabinets	Points to the cabinets feed under a repository.
cancel-checkout	Points to the cancel-checkout behavior for a versionable object.
checkin-next-major	Points to the checkin as next major version behavior for an object that is checked out.
checkin-next-minor	Points to the checkin as next minor version behavior for an object that is checked out.
checkin-branch	Points to the checkin as branch version behavior for an object that is checked out.
child-links	Points to the child links feed resource of a folder.
checkout	Points to the checkout behavior for a versionable object.
comments	Points to top level comments for a sysobject.

Link relation	Description
content-media	Indicates that the content can be downloaded.
current-user	Points to the current login user resource under a repository.
current-user-preferences	Points to the current user preferences resource to manipulate the user preference.
current-version	Points to the current version resource for a versionable object.
default-folder	Points to the default folder resource for a user.
delete	Points to the delete behavior of an object.
dematerialize	Points to the link to dematerialize a lightweight object.
documents	Points to the documents feed under a repository.
folders	Points to the folders feed under a repository.
format	Points to the format resource for a content resource.
formats	Points to the formats feed under a repository.
freeze	Points to the freeze behavior for a snapshot.
groups	Points to the groups feed under a repository or direct member groups under a group.
lightweight-types	Points to the types resource which contains the collection of children lightweight types of a shareable type.
lightweight-objects	Points to the collection of lightweight objects shares one shareable object.
logoff	Points to the log out action for single sign on user.
materialize	Points to the link to materialize a lightweight object.
objects	Points to the Lists folder contents.
object-aspects	Points to the object aspects resource to get attached aspects of an object, or attache an aspect to the object.
parent-links	Points to the parent links resource of a non-cabinet sysobject.
parent-shareable-type	Points to the parent shareable type of a lightweight type.
permissions	Points to the permissions resource for a Sysobject.
permission-set	Points to the permission set resource for a Sysobject or a user.
primary-content	Points to the primary content resource for a content sysobject type.
relate	Indicates that the object can be related to other objects.
relation-type	Points to the relation type resource for a relation instance.
relation-types	Points to the relation types feed under a repository.
relations	Points to the relations feed under a repository, relations feed for a specific relation type, or relations feed related to a specified object.
replies	Points to the log out action for single sign on user.

Link relation	Description
repositories	Points to the repositories feed from the docbrokers.
saved-searches	Points to the saved searches resource under a repository.
search-templates	Points to the search templates resource under a repository.
search-execution	Points to the execution behavior for a saved search or search template.
saved-search-results	Points to the results resource of a saved search.
shared-parent	Points the parent shareable object of a lightweight object.
snapshots	Points to snapshots feed resource under a repository.
snapshot	Points to snapshot resource for a virtual document snapshot.
snapshot-nodes	Points to the snapshot nodes resource to fetch all nodes of a snapshot.
types	Points to the data dictionary types feed under a repository.
unfreeze	Points to the unfreeze behavior for a snapshot.
users	Points to the users feed under a repository, or direct member users under a group.
virtual-document-nodes	Points to the virtual document nodes resource to fetch all nodes of an virtual document.
virtual-document	Points to the sysobject resource, which is a virtual document.
virtual-document-conversion	Points to the conversion behavior for a document.
virtual-document-component	Points to a virtual document node's component object.
The fully qualified OpenText Documentum CM link relation path is prefixed with the following string:	
<code>http://identifiers.emc.com/linkrel/</code>	

Table A-3: Link relations for a user subscription resource

Link relation	Description
subscribe	Indicates that the current object is unsubscribed and points to the subscription link for the current object.
subscriptions	Points to the subscriptions collection of the current user.
unsubscribe	Indicates that the current object has been subscribed to and points to the link to unsubscribe from the current object.

Table A-4: Link relations for an audit management resource

Link relation	Description
audit-trails	Points to the audit trails feed under a repository or a custom scope.
available-audit-events	Points to available audit events for the current object.
registered-audit-events	Points to registered audit events for the current object.
audit-policies	Points to audit policies feed under a repository.
audit-trail	Points to the audit trail resource under a repository.
audit-event	Points to the audit event resource for the current object.
audit-policy	Points to the audit policy resource under a repository.
unregister-all-audit-events	Points to the link within the current scope to unregister from all audit events.
audited-object	Points to the audited object resource of the current audit trail instance.
recent-trails	Points to the audit trails of the current logged in user in a newest-first order.

Table A-5: Link relations for a distributed upload resource

Link relation	Description
distributed-upload	Points to the distributed upload link for the current object according to the setting of the Documentum CM Server.

Table A-6: Link relations for a Lifecycle resource

Link relation	Description
lifecycles	Points to the Lifecycle feed resource under a repository.
attachable-lifecycles	Points to the attachable Lifecycle feed resource under an object.
lifecycle	Points to the Lifecycle resource for a specific Lifecycle.
object-lifecycle	Points to the current Lifecycle resource for a specific object.
promotion	Indicates that a specific Lifecycle object can be promoted.
demotion	Indicates that a specific Lifecycle object can be demoted.
suspension	Indicates that a specific Lifecycle object can be suspended.
resumption	Indicates that a specific Lifecycle object can be resumed.
cancel-promotion	Indicates that a specific scheduled promotion can be canceled.
cancel-demotion	Indicates that a specific scheduled demotion can be canceled.
cancel-suspension	Indicates that a specific scheduled suspension can be canceled.
cancel-resumption	Indicates that a specific scheduled resumption can be canceled.

Appendix B. Resource coding index

This table lists the elements that you may need to reference when developing custom resources.

Resource	Code name	Model class	View class
All versions	versions	AtomFeed	VersionsFeedView
ACLs	acls	AtomFeed	AclsFeedView
ACL	acl	AclObject	AclView
ACL Associations	acl-associations	AtomFeed	SysObjectsFeedView
Aspect types	aspect-types	AtomFeed	AspectTypesFeedView
Aspect type	aspect-type	AspectTypeObject	AspectTypeView
Batch capabilities	batch-capabilities	BatchCapabilities	NA
Batches	batches	Batch	NA
Cabinet	cabinet	CabinetObject	CabinetView
Cabinets	cabinets	AtomFeed	CabinetView
Checked out objects	checked-out-objects	AtomFeed	SysObjectView
Child folder link	child-folder-link	FolderLink	FolderLinkView
Child folder links	child-folder-links	AtomFeed	FolderLinksFeedView
Comment	comment	Comment	CommentView
Comments	comments	AtomFeed	CommentsFeedView
Comment Replies	comment-replies	AtomFeed	CommentsFeedView
Content media resource	content-media	NA	NA
Content	content	ContentMeta	ContentView
Contents	contents	AtomFeed	ContentsFeedView
Current user	current-user	UserObject	UserView
Current user preferences	current-user-preferences	AtomFeed	PreferencesFeedView
Current user preference	current-user-preference	PreferenceObject	PreferenceView
Current version	current-version	SysObject	SysObjectView
Document	document	DocumentObject	DocumentView
Folder child documents	folder-child-document	AtomFeed	DocumentsFeedView

Resource	Code name	Model class	View class
Folder child folders	folder-child-folders	AtomFeed	FoldersFeedView
Folder child objects	folder-child-objects	AtomFeed	SysObjectsFeedView
Folder	folder	FolderObject	FolderView
Format	format	Format	FormatView
Formats	formats	AtomFeed	FormatsFeedView
Group	group	GroupObject	GroupView
Group users	group-member-users	AtomFeed	UsersFeedView
Groups	groups	AtomFeed	GroupsFeedView
Home document	home-document	NA	NA
Lightweight objects	object-lightweight-objects	AtomFeed	SysObjectsFeedView
Lock	lock	SysObject	SysObjectView
Materialization	materialization	SysObject	SysObjectView
Network locations	network-locations	AtomFeed	NetworkLocationsFeedView
Network location	network-location	NetworkLocation	NetworkLocationView
Object aspects	object-aspects	ObjectAspects	ObjectAspectsView
Object parent	object-parent	SysObject	SysObjectView
Parent folder link	parent-folder-link	FolderLink	FolderLinkView
Parent folder links	parent-folder-links	AtomFeed	FolderLinksFeedView
Permissions	permissions	AccessorPermission	AccessorPermissionView
Permission Set	permission-set	PermissionSetObject	PermissionSetView
Product information	product-information	ProductInfo	ProductInfoView
Query	dql-query	AtomFeed	QueryResultFeedView
Relation	relation	RelationObject	RelationView
Relation type	relation-type	RelationTypeObject	RelationTypeView
Relation types	relation-types	AtomFeed	RelationTypesFeedView
Relations	relations	AtomFeed	RelationsFeedView
Repositories	repositories	AtomFeed	RepositoriesFeedView
Repository	repository	Repository	RepositoryView

Resource	Code name	Model class	View class
Search	search	SearchAtomFeed	SearchFeedView
Saved searches	saved-searches	AtomFeed	SavedSearchFeedView
Saved search	saved-search	SavedSearch	SavedSearchView
Search templates	search-templates	AtomFeed	SearchTemplateFeedView
Search template	search-template	SearchTemplate	SearchTemplateView
Saved search results	saved-search-results	SearchAtomFeed	SearchFeedView
Saved search execution	saved-search-execution	SearchAtomFeed	SearchFeedView
Sub groups	group-member-groups	AtomFeed	GroupsFeedView
SysObject	object	SysObject	SysObjectView
Type	type	TypeObject	TypeView
Types	types	AtomFeed	TypesFeedView
Type assistance values	type-assist-values	ValueAssistances	ValueAssistancesView
User default folder	default-folder	FolderObject	FolderView
User	user	UserObject	UserView
Users	users	AtomFeed	UsersFeedView
User Permission Set	user-permission-set	PermissionSetObject	PermissionSetView
Virtual document nodes	virtual-document-nodes	AtomFeed	VirtualDocumentNodesFeedView

1. Model Classes for Core resources are in the package com.emc.documentum.rest.model
 2. View Classes for Core resources are in the package com.emc.documentum.rest.view.impl

