opentext™

OpenText™ Documentum™ Content Management

Foundation CMIS API Reference Guide

Understand capabilities, bindings, and mappings of Foundation CMIS API.

EDCPKCO250400-ARE-EN-01

OpenText™ Documentum™ Content Management
Foundation CMIS API Reference Guide

EDCPKCO250400-ARE-EN-01
Rev.: 2025-Oct-15

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

Table of Contents

Chapter 1

Overview

Foundation CMIS API is an implementation of Content Management Interoperability Services (CMIS) that works with OpenText Documentum Content Management (CM) repositories. Content Management Interoperability Services (CMIS) is an OASIS (http://www.oasis-open.org) specification that defines a domain model and bindings that enable interoperability among enterprise content management (ECM) systems and applications.

This guide is not intended to serve as a reference to Content Management Interoperability Services itself, nor is it intended to serve as a guide to Content Management Interoperability Services development. This guide is for application developers, architects, and other technical professionals who require specific information about how the Content Management Interoperability Services standard is implemented for OpenText Documentum CM.

The need for Content Management Interoperability Services arose from a recognition that many enterprises have production ECM systems from multiple vendors, and that the complexity of making these systems work together is a burden on IT departments and an impediment to developing enterprise-wide applications. The principal goal of Content Management Interoperability Services is to reduce this burden by providing a standard domain model for ECM systems, and expose this model through industry-standard web service (SOAP), RESTful AtomPub bindings, and Browser bindings. Content Management Interoperability Services enables enterprises to develop applications that are work together with various ECM systems and access the content in those system in a uniform manner. Content Management Interoperability Services also provides a way for independent software vendors to develop applications that can be marketed for use with any Content Management Interoperability Services-enabled ECM system.

The Content Management Interoperability Services domain model is generic to ECM systems provided by different vendors. Therefore, it is not a goal of Content Management Interoperability Services to provide access to all the features of any one ECM system. The Content Management Interoperability Services API is not appropriate for all types of ECM applications.

For detailed information on Content Management Interoperability Services, refer to the CMIS 1.1 specification and the CMIS 1.1 Errata 01 specification on the OASIS website.

Chapter 2

Capabilities

2.1 Repository capabilities

This section specifies what optional Content Management Interoperability Services capabilities are supported for a OpenText Documentum CM repository. These are exposed to consumers by the `getRepositoryInfo` service or resource. The capabilities may depend on the enabled features of a particular OpenText Documentum CM repository, such as full-text search.

CMIS capability	Description	Value for OpenText Documentum CM repository
capabilityGetDescendants	Ability for an application to enumerate the descendants of a folder using the getDescendants service.	true
capabilityGetFolderTree	Ability for an application to retrieve the folder tree using the getFolderTree service.	true

CMIS capability	Description	Value for OpenText Documentum CM repository
capabilityContentStream Updatability	Indicates the support a repository has for updating the content stream of a document.	enumCapability ContentStream Updates.anytime A content stream cannot be updated if the document is checked out. If document content is appended by using append content stream call, then get content stream operation returns only the content corresponding to the first append or set content operation performed. To retrieve consolidated content of all the append operations of a document then streamId has to be passed for the get content operation with value ALL. For example, to retrieve the full content of the object with id 0901e5a280022c60 and format crtext: • In AtomPub binding: <CMIS_APP>/browser/repo71/root?objectId=0901e5a280022c60&cmis selector=content&streamId=ALL;format=crtext • In Browser binding: <CMIS_APP>/browser/repo71/root?objectId=0901e5a280022c60&cmis selector=content&streamId=ALL;format=crtext
capabilityChanges	Indicates what level of changes (if any) the repository exposes using the change log service.	enumCapability Changes.none OpenText Documentum CM does not support the change log service.

CMIS capability	Description	Value for OpenText Documentum CM repository
capabilityRenditions	Indicates whether or not the repository exposes renditions of document objects.	enumCapability Rendition.read Note that folder renditions are not implemented.
capabilityMultifiling	Ability for an application to file a document or other file-able object in more than one folder.	true
capabilityUnfiling	Ability for an application to leave a document or other file-able object not filed in any folder.	false OpenText Documentum CM does not support this feature.
capabilityVersion SpecificFiling	Ability for an application to file individual versions (that is, not all versions) of a document in a folder.	false When moving a document, OpenText Documentum CM will move the CURRENT version. When getting descendants, OpenText Documentum CM will get the latest version.
capabilityPWCUpdatable	Ability for an application to update the Private Working Copy (PWC) of a checked out document.	false This capability cannot be implemented because OpenText Documentum CM does not have a PWC feature.
capabilityPWCSearchable	Ability of the repository to include the Private Working Copy (PWC) of checked out documents in query search scope; otherwise, PWCs are not searchable.	false This capability cannot be implemented because OpenText Documentum CM does not have a PWC feature.
capabilityAllVersions Searchable	Ability of the repository to include the non-latest versions of a document in query search scope; otherwise, only the latest version of each document is searchable.	true

CMIS capability	Description	Value for OpenText Documentum CM repository
capabilityQuery	Indicates the types of queries that the repository has the ability to fulfill.	With Fulltext: enumCapabilityQuery.bothcombined Without Fulltext: enumCapabilityQuery.metadataonly
capabilityJoin	Indicates the types of JOIN keywords that the repository can fulfill in queries.	enumCapability Join.inneronly
capabilityACL	Indicates the level of support for ACLs by the repository.	enumCapabilityACL.manage
principalAnonymous	A preconfigured anonymous user.	If set, this field holds the principal who is used for anonymous repository access. This principal can then be passed to the ACL services to access the services as an anonymous user. For more information, see *OpenText Documentum Content Management - Server and Server Extensions Installation Guide (EDCSY250400-IGD)*.
principalAnyone	A preconfigured user that has defined default privileges common to all users.	The `dm_world` alias. This alias represents all of the users in a repository. Unless more specific permissions exist for users, they use this alias's default permissions to access an object.

Chapter 3

Bindings

3.1 Restful AtomPub binding

3.1.1 Authentication

The Restful AtomPub binding as implemented by Foundation CMIS API supports the following authentication:

"HTTP basic authentication" on page 17

3.1.2 HTTP

Foundation CMIS API implements optimistic locking using the ETag HTTP header, the value of which maps to the OpenText Documentum CM i_vstamp property.

Cache control is implemented using the HTTP ETag header for the Object services.

For non-content stream resources, HTTP PATCH is supported instead of HTTP PUT.

3.1.3 Content ranges

The Restful AtomPub binding implements content ranges as specified in RFC 2616 documentation.

3.1.4 URI templates

Foundation CMIS API supports all of the URI templates defined by the CMIS 1.1 specification.

3.1.5 Atom feed links

Foundation CMIS API serves the first, last, next, and previous links of an atom feed when appropriate.

3.1.6 Updated time of feeds

The updated time on a feed will be computed based on the current time to prevent excessive performance cost in attempting to calculate the latest update time in all pages of a feed.

3.1.7 Feed titles

Feeds will use the following titles (where <<*TypeName*>> is the display name of a type definition):

- Type Children Collection
- Type Descendants Feed
- <<*TypeName*>> Type Descendants Feed
- <<*TypeName*>> Type Children Collection
- Query Collection
- Checked Out Collection
- Folder Children Collection
- Object Parents Collection
- Relationships Collection
- Folder Descendants
- Folder Tree
- All Versions Feed

3.1.8 Overlapping elements and properties

Certain AtomPub elements duplicate object properties. For example, `atom:title` and `cmis:objectName`. When this is the case in a request, the following rules apply:

Atom element	Object property	Resolution
not specified	specified	Use property
specified	not specified	Use element
specified	specified	Use element

3.1.9 Enclosures

AtomPub enclosures are populated for a contentful document entry.

3.1.10 Invalid boolean resource parameters

When an invalid value is specified in a Boolean parameter such as `includeAllowableActions` or `includeACL`, the value is treated as false.

3.2 Web Services binding

3.2.1 Authentication

The Foundation CMIS API Web Services binding supports the following authentication:

"WS-Security UsernameToken Profile 1.1 authentication in the Web Services binding" on page 16

3.2.2 Optimistic locking

Foundation CMIS API uses the `changeToken` input and output to implement optimistic locking using the `Documentum i_vstamp` property.

3.2.3 Content ranges

Foundation CMIS API supports content ranges as described in CMIS 1.1 specification documentation.

3.3 Browser binding

Content Management Interoperability Services Browser Binding is based upon JSON (Java Script Object Notation in RFC 4627) and patterns used in Web applications. This binding is specifically designed to support applications running in a web browser but is not restricted to them.

3.3.1 Authentication

The Browser binding as implemented by Foundation CMIS API supports the following authentication:

- Non-browser clients: "HTTP basic authentication" on page 17
- Browser clients: "HTTP basic authentication with tokens in browser binding for browser clients" on page 18

3.3.2 Protocol

HTTP is used as the protocol for service requests. HTTP GET is used for reading content and HTTP POST is used for creating, updating, and deleting content. Using just these two HTTP verbs makes it possible to develop applications that rely on built-in browser capabilities such as HTML Forms and typical server configurations. The use of HTTPS is recommended for repositories that contain non-public data.

3.3.3 Data representation

Browser applications are typically written in JavaScript. A lightweight data representation format is JavaScript Object Notation (JSON). JSON is used to represent the various Content Management Interoperability Services objects described by the data model.

3.3.4 URL patterns

The URLs used by the Browser Binding must be predictable to simplify client development. The URL patterns allow objects to be referenced by either object ID or path.

3.3.5 Multipart forms

Browser applications typically use HTTP multipart forms as described in RFC2388 to create and update content. This is useful to update file content with the addition of the FILE attribute. In this binding, HTTP POST of multipart/form-data is used to update content streams.

3.3.6 Callback

Modern browsers restrict a JavaScript function from making HTTP calls to servers other than the one on which the function originated. This set of restrictions is called the “same-origin policy”. A Content Management Interoperability Services client web application and the Content Management Interoperability Services repositories it uses may often be deployed to different servers. This specification allows JavaScript clients to access repositories on other servers, within the constraints of the same-origin policy, by using the “JSON with Padding”(JSONP) pattern. This binding introduces a parameter called callback to allow Content Management Interoperability Services clients to use this pattern.

A Content Management Interoperability Services client web application and the Content Management Interoperability Services repositories it uses may often be deployed to different servers. This specification allows JavaScript clients to access repositories on other servers, within the constraints of the same-origin policy, by using the “JSON with Padding”(JSONP) pattern. This binding introduces a parameter called callback to allow Content Management Interoperability Services clients to use this pattern. The callback may be included by clients on read operations defined by this protocol that answer a JSON object. The server must respond to valid read requests containing this token by answering the token,

followed by an open parenthesis, followed by the JSON object returned, followed by a close parenthesis. If the parameter is included in a request, the server must validate that its value is not empty but the server must not do any additional validation of the token; for example, assuring it conforms to JavaScript function naming conventions. If the parameter value is empty, or if the parameter is used on a service for which it is not allowed, then the `invalidArgument` exception must be used to signal the error.

3.3.7 JSONP and form requests

Cross-domain requests should use JSONP and callbacks for GET requests and should use HTML forms for POST requests. A token should be added to each request to prevent Cross-Site Request Forgery (CSRF) attacks. A parameter token must be added to the parameters of a GET request and a control token must be added to the controls of a HTML form. The repository should return a `permissionDenied` error if the client sends an invalid token. If the client sends any other form of authentication, such as, Basic Authentication, OAuth, and so on, then the token may be omitted. It is recommended for web applications to always provide a token, even when another form of authentication is in place.

3.4 Authentication in the Foundation CMIS API implementation

All Content Management Interoperability Services services implemented in a OpenText Documentum CM repository require authentication. Authentication is also required to list available repositories that are advertised on a connection broker (historically called docbroker). If the user cannot authenticate across all the repositories, only the authenticated repositories are returned in the repository list. Credentials may be in the form of a token.

The following are the support authentication for the various bindings:

Binding	Authentication
SOAP Binding	"WS-Security UsernameToken Profile 1.1 authentication in the Web Services binding" on page 16. This is the default authentication.
AtomPub binding	"HTTP basic authentication" on page 17. This is the default authentication.
Browser Binding	• "HTTP basic authentication" on page 17. This is the default authentication. • "HTTP basic authentication with tokens in browser binding for browser clients" on page 18

3.4.1 WS-Security UsernameToken Profile 1.1 authentication in the Web Services binding

The WS-Securtiy UsernameToken Profile 1.1 authentication protocol is supported for the web service binding. For a detailed description of the Web Services Security UsernameToken Profile 1.1, see OASIS website.

Client SOAP messages should be secured using a WS-Security UsernameToken Profile header. If the authentication fails, the Content Management Interoperability Services implementation will respond with a WS-Security SOAP fault. The following examples show a WS authenticated request and response.

Note: By default, `<wsu:Timestamp>` is returned in response. You can disable `<wsu:Timestamp>` from being returned by configuring `cmis-security.xml` in `cmis-ws-binding.jar`. For more information, see the comments in `cmis-security.xml`.

Example 3-1: WS authenticated request

```
Content-type: text/xml;charset="utf-8"
Soapaction: ""
Accept: text/xml, multipart/related, text/html, image/gif, image/jpeg, *;
        q=.2, */*; q=.2
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
               oasis-200401-wss-wssecurity-secext-1.0.xsd"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
              oasis-200401-wss-wssecurity-utility-1.0.xsd"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <S:Header>
        <wsse:Security S:mustUnderstand="1">
            <wsse:UsernameToken
                xmlns:ns15="http://schemas.xmlsoap.org/ws/2006/02/
                      addressingidentity"
                xmlns:ns14="http://docs.oasis-open.org/ws-sx/
                      ws-secureconversation/200512"
                xmlns:ns13="http://www.w3.org/2003/05/soap-envelope"
                      wsu:Id="XWSSGID-129644497810287092214">
                <wsse:Username>my-user</wsse:Username>
                <wsse:Password
                    Type="http://docs.oasis-open.org/wss/2004/01/
                   oasis-200401-wss-username-token-profile-1.0#PasswordText">
                   my-password</wsse:Password>
            </wsse:UsernameToken>
        </wsse:Security>
    </S:Header>
    <S:Body>
        <getRepositoryInfo
            xmlns="http://docs.oasis-open.org/ns/cmis/messaging/200908/"
            xmlns:ns2="http://docs.oasis-open.org/ns/cmis/core/200908/">
            <repositoryId>my-repository</repositoryId>
        </getRepositoryInfo>
    </S:Body>
</S:Envelope>
```

Example 3-2: WS authenticated response

```
Transfer-encoding: chunked
null: HTTP/1.1 200 OK
Content-type: text/xml;charset=utf-8
Server: Apache-Coyote/1.1
Date: Mon, 31 Jan 2011 03:36:17 GMT
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
               oasis-200401-wss-wssecurity-secext-1.0.xsd"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
              oasis-200401-wss-wssecurity-utility-1.0.xsd"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <S:Header>
        <wsse:Security S:mustUnderstand="1">
            <wsu:Timestamp
                xmlns:ns15="http://schemas.xmlsoap.org/ws/2006/02/addressingidentity"
                xmlns:ns14="http://docs.oasis-open.org/ws-sx/ws-secureconversation/
200512"
                xmlns:ns13="http://www.w3.org/2003/05/soap-envelope"
                      wsu:Id="XWSSGID-1296444877084-1727713687">
                <wsu:Created>2011-01-31T03:36:18Z</wsu:Created>
                <wsu:Expires>2011-01-31T03:41:18Z</wsu:Expires>
            </wsu:Timestamp>
        </wsse:Security>
    </S:Header>
    <S:Body>
        <getRepositoryInfoResponse
            xmlns="http://docs.oasis-open.org/ns/cmis/messaging/200908/"
            xmlns:ns2="http://docs.oasis-open.org/ns/cmis/core/200908/">
            <repositoryInfo>
                <ns2:repositoryId>my-repository</ns2:repositoryId>
                <ns2:repositoryName>my-repository</ns2:repositoryName>
                <ns2:repositoryDescription>my-repository</ns2:repositoryDescription>
                <ns2:vendorName>OpenText</ns2:vendorName>
                <ns2:productName>Documentum</ns2:productName>
                <ns2:productVersion>6.6.0.030</ns2:productVersion>
                <ns2:rootFolderId>root</ns2:rootFolderId>
                .....
                <ns2:cmisVersionSupported>1.1</ns2:cmisVersionSupported>
            </repositoryInfo>
        </getRepositoryInfoResponse>
    </S:Body>
</S:Envelope>
```

3.4.2 HTTP basic authentication

HTTP basic authentication is supported for the Restful AtomPub binding and Browser binding for non-browser clients in Foundation CMIS API. For a detailed description of the HTTP basic authentication protocol, refer to RFC 2617.

As credentials passed in clear text in HTTP can be captured and misused, HTTP basic authentication should be used with HTTPS to provide encrypted communication. Foundation CMIS API supports this option.

3.4.3 HTTP basic authentication with tokens in browser binding for browser clients

The authentication mechanism described in this section addresses the following scenario: A web application is hosted on one domain; the Content Management Interoperability Services browser binding interface is served from another domain. There is no proxy process on the server that hosts the web application. That is, all communication between the application and the repository has to happen in the web browser using JavaScript. The “sameoriginpolicy”enforced by the web browser prohibits a direct and secure two-way communication between the application and the repository. To access the repository, a user has to authenticate and has to authorize the application and only this application, not all scripts in the web browser to make Content Management Interoperability Services calls.

JSONP and form requests: Cross-domain requests should use JSONP and callbacks for GET requests and should use HTML forms for POST requests. “Browser binding” on page 13 provides more details.

Chapter 4

Content Management Interoperability Services to OpenText Documentum CM mapping

4.1 Repository elements mapping

CMIS	OpenText Documentum CM	Description
repositoryId	Unique repository name as obtained from connection broker	
repositoryName	Identical to repositoryId	
repositoryDescription	Description of repository in connection broker	
vendorName	"OpenText"	
productName	"Documentum"	
productVersion	dm_server_config.r_server_version	
rootFolderId	"root"	A fictitious root folder for the repository.
latestChangeLogToken	Not set	Change log not supported.
cmisVersionSupported	1.1	
thinClientURI	Not set	An optional repository-specific URI pointing to the web interface of the repository.
capabilities		The collection of Content Management Interoperability Services capabilities supported by the repository. See "Repository capabilities" on page 7.
aclCapability		The collection of ACL-related capabilities supported by the repository. See "ACL mapping" on page 31.
changesIncomplete	Not set	Change log not supported.
changesOnType	Not set	Change log not supported.

4.2 Folder mapping

4.2.1 Root folder

Content Management Interoperability Services has a single root folder per repository and OpenText Documentum CM has multiple root folders (cabinets) per repository. To address this difference, the implementation exposes a fictitious root folder that contains all root cabinets. The `cmis:objectId` of this root folder is `root`.

Because the root folder is a fictitious object, it is not queryable, and no ACL can be applied to it. Beneath the root folder, only objects of type `dm_cabinet` can be created.

To create a `dm_cabinet` (or subtype) object as a child of the root folder, simply create a `cmis:folder` object under the root folder, Foundation CMIS API automatically translates the `cmis:folder` object to a `dm_cabinet` object. Specifying a `cmis:folder` subtype object other than `dm_cabinet` under the root folder does not create a `dm_cabinet` (or subtype) and causes an error.

Note: A consumer can determine the object types that can exist under the root folder by examining its *`cmis:allowedChildObjectTypeIds`* property.

4.2.2 Folder multi filing

Content Management Interoperability Services specifies that folders only have one parent. OpenText Documentum CM allows folders to be multi filed in multiple folders, creating multiple possible paths to a folder. In cases where the parent or path to a folder is in a response, the implementation returns only the primary parent folder or path that is, the first of all the possible paths as determined by OpenText™ Documentum™ Content Management Server. When accessing an object by path the implementation locates the object using any folder path.

In determining the parent of an object the parent may be reassigned if the primary parent of the folder is not accessible; that is, if permission = none.

If a folder has multiple parents and the client navigates from root to leaf, the parent link in the child entry will be directed to the parent that traverses to the child entry. Thus, if a folder has multiple parents, its parent link (`cmis:parentId`) may be different in different branches. This behavior applies to the navigation service's getChildren, getDescendants, and getFolderTree operations with AtomPub binding only.

Similarly, if a folder has multiple parents and the client navigates from root to leaf, the path for the folder (the cmis:path property) will be the traversal path from its dynamic parent. Thus, if a folder has multiple parents, its path may be different in different branches.

4.2.3 Path segments

Content Management Interoperability Services allows documents to be addressed by path. In some ECM systems the path to a document is unique, but in OpenText Documentum CM a document cannot be uniquely identified by what would normally be considered a path. Because identically named objects can reside in the same folder.

The Foundation CMIS API implementation therefore exposes unique document path segments when requested using a combination of the object_name and r_object_id. This is done for every document, not just those that cannot be addressed by path. The path segments take this form:

`with name: 0900000b80001234_docnamewithout name: 0900000b80001234_`

4.2.4 getFolderTree default depth

The default depth of the getFolderTree resource or service is 2.

4.3 Type mapping

4.3.1 Base types

Content Management Interoperability Services includes a standardization of base types. The Content Management Interoperability Services base types are mapped to the local OpenText Documentum CM types as shown in the following table:

CMIS type	OpenText Documentum CM (local) type	Notes
cmis:document	dm_document (and subtypes)	
cmis:folder	dm_folder (and subtypes)	
cmis:item	dm_item (and subtypes)	
cmis:secondary	dmc_aspect_type	
cmis:relationship	dm_relation (and subtypes)	An instance of dm_relation, or of a subtype of dm_relation, where its definition (that is, its relation name) is an instance of dm_relation_type.
cmis:policy		Not supported.

Only these OpenText Documentum CM types and their corresponding subtypes are supported.

When a base type is used in a service input, the Content Management Interoperability Services type must be specified. The equivalent local types (`dm_`

`document`, `dm_folder`, `dm_item`, `dmc_aspect_type`, `dm_relation`) must not be used as service inputs.

Object types are cached. You can configure object type caching characteristics. For more information, see *OpenText Documentum Content Management - Server and Server Extensions Installation Guide (EDCSY250400-IGD)*.

4.4 Property type mapping

CMIS property type	OpenText Documentum CM property type	Notes
CmisPropertyBoolean	DM_BOOLEAN	
CmisPropertyDateTime	DM_TIME	OpenText Documentum CM literals, such as `NOW`, `YESTERDAY` and `TOMORROW`, that are default values in `datetime` properties are set to `NULL`.
CmisPropertyDecimal	DM_DOUBLE	
CmisPropertyHtml	DM_STRING	
CmisPropertyId	DM_ID	
CmisPropertyInteger	DM_INTEGER	
CmisPropertyString	DM_STRING	
CmisPropertyUri	DM_STRING	

4.5 Property mapping

4.5.1 General information

All property names and query names are case-insensitive in service requests. Content Management Interoperability Services services respond with lowercase local (or custom) properties, and lower camel case Content Management Interoperability Services properties.

4.5.2 OpenText Documentum CM properties

CMIS property	OpenText Documentum CM property	Description
cmis:name	object_name	
cmis:objectId	r_object_id	
cmis:baseTypeId	cmis:document	See "Type mapping" on page 21.
cmis:objectTypeId	r_object_type	
cmis:createdBy	r_creator_name	
cmis:creationDate	r_creation_date	
cmis:lastModifiedBy	r_modifier	
cmis:lastModificationDate	r_modification_date	
cmis:changeToken	i_vstamp	See "Change tokens" on page 26.
cmis:isImmutable	r_immutable_flag	
cmis:isLatestVersion	i_has_folder	
cmis:isMajorVersion	none	Determined from r_version_label
cmis:isLatestMajorVersion	none	Computed
cmis:versionLabel	r_version_label	Numeric label
cmis:versionSeriesId	i_chronicle_id	
cmis:isVersionSeries CheckedOut	none	Determined from r_lock_owner
cmis:versionSeries CheckedOutBy	r_lock_owner	
cmis:versionSeries CheckedOutId	none	Always Not set
cmis:checkinComment	log_entry	
cmis:contentStream Length	r_full_content_size	
cmis:contentStreamMimeTyp e	a_content_type	
cmis:contentStreamFileName	object_name	Read-only
cmis:contentStreamId	i_contents_id	
cmis:SecondaryObjectTypeId s	r_aspect_name	All the secondary type associated

CMIS property	OpenText Documentum CM property	Description
<SecondaryTypeName>.<propertyName>		For example : *<SecondaryTypeName1>.<property-A>* , *<SecondaryTypeName2>.<property-B>*

4.5.3 Behavior when property filter is not set

When the property filter is Not set in a request, it will return all Content Management Interoperability Services properties (that is, all properties in the Content Management Interoperability Services namespace), with the exception of cmis:isLatestMajorVersion. This property is excluded for performance reasons (that is, it is a computed value that is expensive to calculate). To get the cmis:isLatestMajorVersion property, the consumer can set `filter=cmis:isLatestMajorVersion` or `filter=*`.

4.5.4 Folder properties

CMIS property	OpenText Documentum CM property	Description
cmis:name	object_name	
cmis:objectId	r_object_id	
cmis:baseTypeId	cmis:folder	See "Type mapping" on page 21.
cmis:objectTypeId	r_object_type	
cmis:createdBy	r_creator_name	
cmis:creationDate	r_creation_date	
cmis:lastModifiedBy	r_modifier	
cmis:lastModificationDate	r_modification_date	
cmis:changeToken	i_vstamp	See "Change tokens" on page 26.
cmis:parentId	i_folder_id	
cmis:path	r_folder_path	
cmis:allowedChildObjectTypeIds		See "Type mapping" on page 21.
cmis:SecondaryObjectTypeIds	r_aspect_name	All the secondary type associated

CMIS property	OpenText Documentum CM property	Description
<SecondaryTypeName>.<propertyName>		For example : *<SecondaryTypeName1>.<property-A>* , *<SecondaryTypeName2>.<property-B>*

4.5.5 Item properties

CMIS property	OpenText Documentum CM property	Description
cmis:name	object_name	
cmis:objectId	r_object_id	
cmis:baseTypeId	cmis:item	See "Base types" on page 21.
cmis:objectTypeId	r_object_type	
cmis:createdBy	r_creator_name	
cmis:creationDate	r_creation_date	
cmis:lastModifiedBy	r_modifier	
cmis:lastModificationDate	r_modification_date	
cmis:changeToken	i_vstamp	See "Change tokens" on page 26.
cmis:isImmutable	r_immutable_flag	
cmis:item	dm_item (and subtypes)	

4.5.6 Secondary properties

CMIS property	OpenText Documentum CM property	Description
cmis:name	object_name	
cmis:objectId	r_object_id	
cmis:baseTypeId	cmis:secondary	See "Base types" on page 21.
cmis:objectTypeId	r_object_type	
cmis:createdBy	r_creator_name	
cmis:creationDate	r_creation_date	
cmis:lastModifiedBy	r_modifier	
cmis:lastModificationDate	r_modification_date	
cmis:changeToken	i_vstamp	See "Change tokens" on page 26.

CMIS property	OpenText Documentum CM property	Description
cmis:isImmutable	r_immutable_flag	
cmis:secondary	dmc_aspect_type	
cmis:parentId	cmis:secondary	

4.5.7 Relationship properties

CMIS property	OpenText Documentum CM property	Description
cmis:name	object_name	
cmis:objectId	r_object_id	
cmis:baseTypeId	cmis:relationship	See "Type mapping" on page 21.
cmis:objectTypeId	none	
cmis:createdBy	none	
cmis:creationDate	none	
cmis:lastModifiedBy	none	
cmis:lastModificationDate	none	
cmis:changeToken	i_vstamp	See "Change tokens" on page 26.
cmis:sourceId	parent_id	
cmis:targetId	child_id	

4.5.8 Change tokens

The changeToken property is used for optimistic locking and any concurrency checking to ensure that user updates do not conflict.

For an invocation of any update method on an object (updateProperties, setContentStream, deleteContentStream), OpenText suggests that the client application provide the value of the changeToken property as an input parameter. If the client application sets the changeToken property, the value must match the value that the repository provides for the changeToken property of that object. Otherwise, the repository throws an updateConflictException.

Note that setting the changeToken property is not mandatory. If the client application does not set this property for an update method, the repository does not reject the request.

4.6 Query mapping

4.6.1 Queryable types

The types cmis:document and cmis:folder are queryable; the type cmis:relationship is not queryable.

Type	Queryable
cmis:document	true
cmis:folder	true
cmis:item	true
cmis:secondary	true
cmis:relationship	false

Note: All properties of an object of type cmis:relationship are non-queryable because cmis:relationship is a non-queryable type.

4.6.2 Type names in queries

Only type query names can be used in Content Management Interoperability Services SQL queries. For the base types, the query name is equivalent to the Content Management Interoperability Services name. Local base type names (such as dm_document, dm_folder, dm_item, dmc_aspect_type, dm_relation) cannot be used in queries. For information on OpenText Documentum CM types, see *OpenText Documentum Content Management - Server System Object Reference Guide (EDCCS250400-ORD)*.

All type names and query names are case-insensitive for service inputs (as are column names, table names, and reserved keywords). When responding, the Content Management Interoperability Services services responds with lowercase type names.

4.6.3 Properties not allowed in WHERE clause

Some Content Management Interoperability Services properties are computed, rather than mapped to a single persistent OpenText Documentum CM property. In some cases, the Foundation CMIS API implementation may not support including these computed properties in the WHERE clause of a query. For example, because cmis:isMajorVersion is computed from multiple OpenText Documentum CM properties, the following query is not supported:

```
SELECT * FROM cmis:document WHERE cmis:isMajorVersion=true
```

These cmis:folder and cmis:document properties cannot be used in the WHERE clause:

- cmis:baseTypeId

- cmis:objectTypeId
- cmis:isMajorVersion
- cmis:isLatestMajorVersion
- cmis:isVersionSeriesCheckedOut
- cmis:versionSeriesCheckedOutId
- cmis:contentStreamMimeType
- cmis:contentStreamFileName
- cmis:allowedChildObjectTypeIds

4.6.4 Search queries

Foundation CMIS API supports search queries for both full-text content and metadata of documents. Additionally, these can be combined into a single query as shown in the following example:

```
SELECT * FROM cmis:document WHERE CONTAINS('documentum')
```

4.6.4.1 xPlore search

Foundation CMIS API supports search using xPlore. For example, you can execute a search query using the following AtomPub web address:

```
<context path>/dctm-cmis/resources/xplore/query?q=SELECT cmis:name FROM cmis:document
WHERE CONTAINS ('<document name>')
```

4.6.4.2 Secondary type queries

Foundation CMIS API supports secondary types to assign additional metadata and behaviors to documents, enabling flexible and dynamic classification beyond the primary object type. For example:

```
SELECT * FROM aspect1 WHERE aspect1.school = 'podar'
```

In the preceding example, `aspect1` defines the reusable properties that can be attached to multiple document types. `aspect1` is a secondary type that extends the primary document type by adding additional attributes such as school. This query filters documents where the `school` property in `aspect1` is set to `podar`.

The following are examples of adding additional conditions using AND or OR:

```
SELECT * FROM aspect1 WHERE aspect1.school = 'podar' AND aspect1.grade = '10th'
```

```
SELECT * FROM aspect1 WHERE aspect1.school = 'podar' OR aspect1.grade = '10th'
```

4.6.5 Limitations

Note the following limitations on Foundation CMIS API queries:

- Duplicate column aliases are not supported in Content Management Interoperability Services queries, because during the SQL to DQL translation and conversion process, ambiguity that limits alias use is introduced. For example, the following query would not be supported because two different columns use the same alias (“id_or_name”):

```
SELECT cmis:objectId AS id_or_name, cmis:name AS id_or_name FROM cmis:document
```

- International characters in table names or column names are not supported.

4.7 Format mapping

4.7.1 Format and MIME type

Content Management Interoperability Services content is categorized by MIME type, whereas OpenText Documentum CM content is mapped to a format configured in repository metadata. In OpenText Documentum CM, the `dm_format` table stores the mapping between formats and the corresponding MIME types. Typically, when an object is accessed through Content Management Interoperability Services, its MIME type is retrieved from the `dm_format` table according to the object’s format. Similarly, when an object is stored into OpenText Documentum Content Management (CM) Server through Content Management Interoperability Services, its format is retrieved from the table according to the MIME type.

Multiple formats can be mapped to a single MIME type. For example, the `excel8book` and `excel`formats are mapped to the same MIME type `application/vnd.ms-excel` in the `dm_format` table when you use the default configuration. When an object is read from Documentum CM Server through Content Management Interoperability Services, the process is straightforward. The format's corresponding MIME type will be used. When an object is stored into Documentum CM Server through Content Management Interoperability Services, and client provides a MIME type that can be mapped to multiple formats, the Content Management Interoperability Services server must determine which format to use. In this case, the Content Management Interoperability Services server uses the MIME type/format mapping table in the `mapping.xml` file (located at `WEB-INF/classes`, or in `APP-INF` in EAR file deployments) to determine the format. For example, if an entry maps the MIME type `application/vnd.ms-excel`to the `excel8book` format in `mapping.xml`, and the client provides `application/vnd.ms-excel` as an object's MIME type, its format will be `excel8book`.

If the MIME type of the object is not specified in `mapping.xml`, the mapping in the `dm_format` table will be used to determine the format. One of the formats matching the MIME type will be selected as the format of the object. The result may vary depending on the environment, such as the database where the `dm_format` resides, and the collation. To get a consistent result, it is highly recommended that you specify the MIME type and the corresponding format in `mapping.xml`.

When you access an object from Documentum CM Server through Content Management Interoperability Services, if the format of the object does not have a MIME type specified in the `dm_format` table, the `cmis:contentStreamMIMEType` property of the Content Management Interoperability Services object will be populated with the "value not set"state. When you store an object into Documentum CM Server through Content Management Interoperability Services, if the MIME type of the object does not have a format specified in the dm_format table, the object is treated as binary. The MIME type for binaries is `application/octet-stream`.

Notes

- The format/MIME type mapping in `mapping.xml` takes precedence over the mapping in the `dm_format` table.
- MIME types are cached. You can configure MIME type caching characteristics. For more information, see *OpenText Documentum Content Management - Server and Server Extensions Installation Guide (EDCSY250400-IGD)*.

4.8 Allowable actions mapping

4.8.1 Mapping Content Management Interoperability Services allowable actions to permissions

Content Management Interoperability Services allowable actions enable an application to determine which service operations can be performed on a specified object. The following table shows mappings between Content Management Interoperability Services allowable actions and permissions. Other factors may affect the ability to apply a service operation to an object.

For more information on Content Management Interoperability Services and OpenText Documentum CM permissions, see "ACL mapping" on page 31.

CMIS allowable action	Required permission(s)
canGetDescendents.Folder	browse
canGetChildren.Folder	browse
canGetParents.Folder	browse
canGetFolderParent.Object	browse
canCreateDocument.Folder	cmis:write
canCreateFolder.Folder	cmis:write
canCreateRelationship.Source	relate
canCreateRelationship.Target	relate
canGetProperties.Object	browse
canViewContent.Object	cmis:read

CMIS allowable action	Required permission(s)
canUpdateProperties.Object	cmis:write
canMove.Object	cmis:write, change_location
canMove.Target	cmis:write
canMove.Source	cmis:write
canDelete.Object	delete
canSetContent.Document	cmis:write
canDeleteContent.Document	cmis:write
canAddToFolder.Object	cmis:write, change_location
canRemoveFromFolder.Folder	cmis:write
canCheckout.Document	version
canCancelCheckout.Document	version
canCheckin.Document	version
canGetAllVersions.VersionSeries	cmis:read
canGetObjectRelationships.Object	browse
canGetACL.Object	browse
canApplyACL.Object	change_permit

4.9 ACL mapping

4.9.1 About ACLs

The CMIS 1.1 specification states that an implementation can support a set of Content Management Interoperability Services basic permissions and/or a set of repository-specific permissions. The OpenText Documentum CM implementation supports the Content Management Interoperability Services basic permissions and extends the Content Management Interoperability Services permissions with a set of repository-specific permissions that map directly to OpenText Documentum CM permissions.

In the OpenText Documentum CM repository, when an object is created an ACL is either explicitly assigned to it by the user or a default ACL, which is configurable is assigned to it by OpenText Documentum CM. This repository behavior is not changed in by the Content Management Interoperability Services implementation.

The root folder which is fictitious in a OpenText Documentum CM repository has no ACL.

4.9.2 Content Management Interoperability Services basic and OpenText Documentum CM permissions

A Content Management Interoperability Services client has the option of getting the ACL of an object with just the CMIS-defined basic permissions, or with the OpenText Documentum CM permission set. The default behavior of OpenText Documentum CM implementation for the getAcl service, as well as for services that have an includeACL parameter, is to return the Content Management Interoperability Services basic permissions.

4.9.3 ACL Type Mapping

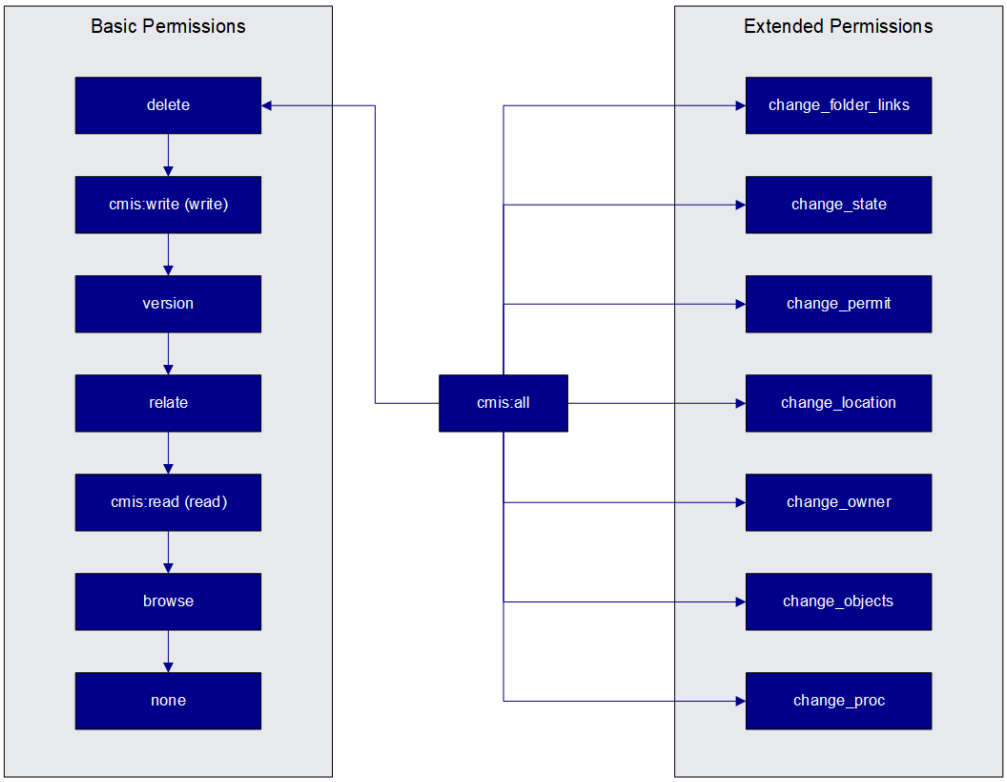
The following table shows the mapping between Content Management Interoperability Services ACL-related types and properties to OpenText Documentum CM types and properties.

CMIS type/property	OpenText Documentum CM type/property	Notes
CmisAccessControlListType	dm_acl	
CmisACLType.exact		A value of false indicates that there are security constraints applied to this object other than those designated by the CMIS ACL object.
CmisAccessControl EntryType		An ACL entry. There is no direct correspondence to the OpenText Documentum CM type.
CmisAccessControl EntryType.permission	r_accessor_permit	A OpenText Documentum CM basic or extended permission.
CmisAccessControl EntryType.direct		A value of true indicates that the ACE is directly assigned to the object; that is, it is not derived.
CmisAccessControl PrincipalType	r_accessor_name	A user, group, role, or OpenText Documentum CM-specific accessor.

4.9.4 Permissions mapping

OpenText Documentum CM permissions are of two types: basic permissions, which are hierarchical, and extended permissions, which are composite. This distinction is applicable for CMIS as well. The `cmis:read` and `cmis:write` permissions map to the OpenText Documentum CM read and write permissions, and each of them includes all permissions below them in the basic permission hierarchy. The `cmis:all` permission includes all basic and extended permissions.

Foundation CMIS API Permissions Hierarchy



CMIS permission	OpenText Documentum CM permission
none	none
browse	browse
cmis:read	read
relate	relate
version	version
cmis:write	write
delete	delete

CMIS permission	OpenText Documentum CM extended permission
change_location	change_location
change_owner	change_owner
change_permit	change_permit
change_state	change_state
change_folder_links	change_folder_links
delete_object	delete_object
execute_proc	execute_proc

Example 4-1: ACL representation with only CMIS basic permissions

```
<cmis:acl>
<cmis:permission>
    <cmis:principal>
        <cmis:principalId>dm_world</cmis:principalId>
    </cmis:principal>
    <cmis:permission>cmis:read</cmis:permission>
    <cmis:direct>true</cmis:direct>
</cmis:permission>
<cmis:permission>
    <cmis:principal>
        <cmis:principalId>dm_owner</cmis:principalId>
    </cmis:principal>
    <cmis:permission>cmis:write</cmis:permission>
    <cmis:direct>true</cmis:direct>
</cmis:permission>
<cmis:permission>
    <cmis:principal>
        <cmis:principalId>docu</cmis:principalId>
    </cmis:principal>
    <cmis:permission>cmis:read</cmis:permission>
    <cmis:direct>true</cmis:direct>
</cmis:permission>
</cmis:acl>
```

Example 4-2: ACL representation with OpenText Documentum CM permissions

```
<cmis:acl>
<cmis:permission>
    <cmis:principal>
        <cmis:principalId>dm_world</cmis:principalId>
    </cmis:principal>
    <cmis:permission>cmis:read</cmis:permission>
    <cmis:permission>execute_proc</cmis:permission>
    <cmis:permission>change_location</cmis:permission>
    <cmis:direct>true</cmis:direct>
</cmis:permission>
<cmis:permission>
    <cmis:principal>
        <cmis:principalId>dm_owner</cmis:principalId>
    </cmis:principal>
    <cmis:permission>delete</cmis:permission>
    <cmis:permission>execute_proc</cmis:permission>
    <cmis:permission>change_location</cmis:permission>
    <cmis:direct>true</cmis:direct>
```

```
</cmis:permission>
<cmis:permission>
    <cmis:principal>
        <cmis:principalId>docu</cmis:principalId>
    </cmis:principal>
    <cmis:permission>version</cmis:permission>
    <cmis:direct>true</cmis:direct>
</cmis:permission>
</cmis:acl>
```