Content Management Interoperability Services – Domain Model Version 0.62c

DRAFT

29 June 2009

Specification URIs:
This Version:
http://docs.oasis-open.org/[tc-short-name] / [additional path/filename] .html
http://docs.oasis-open.org/[tc-short-name] / [additional path/filename] .doc
http://docs.oasis-open.org/[tc-short-name] / [additional path/filename] .pdf

Previous Version:
http://docs.oasis-open.org/[tc-short-name] / [additional path/filename] .html
http://docs.oasis-open.org/[tc-short-name] / [additional path/filename] .doc
http://docs.oasis-open.org/[tc-short-name] / [additional path/filename] .pdf

Latest Version:
http://docs.oasis-open.org/[tc-short-name] / [additional path/filename] .html
http://docs.oasis-open.org/[tc-short-name] / [additional path/filename] .doc
http://docs.oasis-open.org/[tc-short-name] / [additional path/filename] .pdf

Technical Committee:
OASIS CMIS TC

Chair(s):
David Choy

Editor(s):
Ethan Gur-esh
Ryan McVeigh
Al Brown

Related work:

This specification replaces or supersedes:

- Not applicable

This specification is related to:

- Content Repository for Java – JSR 170/283: http://www.jcp.org/en/jsr/detail?id=283
- Atom Publishing Protocol – APP: http://www.ietf.org/internet-drafts/draft-ietf-atompub-protocol-15.txt

Declared XML Namespace(s):
[list namespaces here]
[list namespaces here]

Abstract:

The Content Management Interoperability Services (CMIS) standard defines a domain model (in this document) and set of bindings (TODO: Add links), such as Web Service and REST/Atom that can be used by applications to work with one or more Content Management repositories/systems.

The CMIS interface is designed to be layered on top of existing Content Management systems and their existing programmatic interfaces. It is not intended to prescribe how specific features should be implemented within those CM systems, nor to exhaustively expose all of the CM system's capabilities through the CMIS interfaces. Rather, it is intended to define a generic/universal set of capabilities provided by a CM system and a set of services for working with those capabilities.

Status:

Notices

Table of Contents

1 Introduction

The Content Management Interoperability Services (CMIS) standard defines a domain model (in this document) and set of bindings (TODO: Add links), such as Web Service and REST/Atom that can be used by applications to work with one or more Content Management repositories/systems.

The CMIS interface is designed to be layered on top of existing Content Management systems and their existing programmatic interfaces. It is not intended to prescribe how specific features should be implemented within those CM systems, nor to exhaustively expose all of the CM system's capabilities through the CMIS interfaces. Rather, it is intended to define a generic/universal set of capabilities provided by a CM system and a set of services for working with those capabilities.

1.1 Terminology

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MA and “OPTIONAL” in this document are to be interpreted as described in **[RFC2119]**.

1.2 Normative References

[RFC2119]	S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, http://www.ietf.org/rfc/rfc2119.txt, IETF RFC 2119, March 1997.
[ISO/IEC 9075:1992]	*Information technology - Database languages – SQL* http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt
[Reference]	[Full reference citation]

1.3 Non-Normative References

[Reference]	[Full reference citation]

NOTE: The proper format for a citation to an OASIS Technical Committee's work (whether Normative or Non-Normative) is:

OASIS

Stage (Committee Draft 01, Committee Draft 02, Committee Specification 01, etc. or Standard)

Title (italicized or in quotation marks)

Approval Date (Month YYYY)

URI of the actual Authoritative Specification (namespace is not acceptable as the content changes over time)

For example:

EDXL-HAVE	OASIS Standard, "Emergency Data Exchange Language (EDXL) Hospital AVailability Exchange (HAVE) Version 1.0", November 2008. http://docs.oasis-open.org/emergency/edxl-have/os/emergency_edxl_have-1.0-spec-os.doc

2 Data Model

CMIS provides an interface for an application to access a ***Repository***. To do so, CMIS specifies a core data model that defines the *persistent* information entities that are managed by the repository, and specifies a set of basic services that an application can use to access and manipulate these entities. In accordance with the CMIS objectives, this data model does not cover *all* the concepts that a full-function ECM repository typically supports. Specifically, transient entities (such as programming interface objects), administrative entities (such as user profiles), and extended concepts (such as compound or virtual document, work flow and business process, event and subscription) are not included.

However, when an application connects to a CMIS service endpoint, the same endpoint MAY provide access to more than one CMIS repositories. (How an application obtains a CMIS service endpoint is outside the scope of CMIS. How the application connects to the endpoint is a part of the protocol that the application uses.) An application SHALL use the CMIS “Get Repositories” service (*getRepositories*) to obtain a list of repositories that are available at that endpoint. For each available repository, the Repository MUST return a ***Repository Name***, a ***Repository Identity***, and a URI. The Repository Identity MUST uniquely identify an available repository at this service endpoint. Both the repository name and the repository identity are opaque to CMIS. Aside from the “Get Repositories” service, all other CMIS services are single-repository-scoped, and require a Repository Identity as an input parameter. In other words, except for the “Get Repositories” service, multi-repository and inter-repository operations are not supported by CMIS.

2.1 Repository

2.1.1 Optional Capabilities

Commercial ECM repositories vary in their designs. Moreover, some repositories are designed for a specific application domain and may not provide certain capabilities that are not needed for their targeted domain. Thus, a repository implementation may not necessarily be able to support all CMIS capabilities. A few CMIS capabilities are therefore “optional” for a repository to be compliant. A repository’s support for each of these optional capabilities is discoverable using the CMIS “Get Repository Information” service (*getRepositoryInfo*). The following is the list of these optional capabilities. All capabilities are “Boolean” (i.e. the Repository either supports the capability entirely or not at all) unless otherwise noted.

Navigation Capabilities:

`capabilityGetDescendants`

Ability for an application to enumerate the descendants of a folder via the getDescendants service.

See section: getDescendants

Object Capabilities:

`capabilityContentStreamUpdatability`

Indicates the support a repository has for updating a document’s content stream. Valid values are:

- **`anytime:`** The content stream may be updated any time.

- **`pwconly:`** The content stream may be updated only when updating the "Private Working Copy".

See Section: Content Stream

`capabilityChanges` (enumeration)

Indicates what level of changes (if any) the repository exposes via the "change log" service. Valid values are:

- **`none:`** The repository does not expose any information in the change log.
- **`objectidsonly:`** The change log can return only the ObjectIDs for changed objects in the repository and an indication of the type of change, not details of the actual change.
- **`all`**: The change log can return the ObjectIDs for changed objects in the repository and the details of the actual change

See Section: Change Log

`capabilityRenditions`

Indicates whether or not the repository exposes renditions of document objects.

- **`none:`** The repository does not expose renditions at all.
- **`read:`** Renditions are prodived by the repository and readable by the client.

Multi-Filing Capabilities:

`capabilityMultifiling`

Ability for an application to file a document or other file-able object in more than one folder

See Section: Folder Object

`capabilityUnfiling`

Ability for an application to leave a document or other file-able object not filed in any folder

See Section: Folder Object

`capabilityVersionSpecificFiling`

Ability for an application to file individual versions (i.e., not all versions) of a document in a folder

See Section: Versioning

Versioning Capabilities:

`capabilityPWCUpdateable`

Ability for an application to update the "Private Working Copy" of a checked-out document

See Section: Versioning

`capabilityPWCSearchable`

Ability of the Repository to include the "Private Working Copy" of checked-out documents in query search scope; otherwise PWC's are not searchable

See Section: Versioning

`capabilityAllVersionsSearchable`

Ability of the Repository to include non-latest versions of document in query search scope; otherwise only the latest version of each document is searchable

See Section: Versioning

Query Capabilities:

`capabilityQuery` (enumeration)

Indicates the types of queries that the Repository has the ability to fulfill. Query support levels are:

- **`none`:** No queries of any kind can be fulfilled.
- **`metadataonly`:** Only queries that filter based on object properties can be fulfilled.
- **`fulltextonly`:** Only queries that filter based on the full-text content of documents can be fulfilled.
- **`bothseparate`:** The repository can fulfill queries that filter EITHER on the full-text content of documents OR on their properties, but NOT if both types of filters are included in the same query.
- **`bothcombined`:** The repository can fulfill queries that filter on both the full-text content of documents and their properties in the same query.

See Section: Query

`capabilityJoin` (enumeration)

Indicates the types of SQL JOIN keywords that the Repository can fulfill in queries. Support levels are:

- **`none`:** The repository cannot fulfill any queries that include any JOIN clauses.
- **`inneronly`:** The repository can fulfill queries that include an INNER JOIN clause, but cannot fulfill queries that include other types of JOIN clauses.
- **`innerandouter`:** The repository can fulfill queries that include any type of JOIN clause defined by the CMIS query grammar.

See Section: Query

ACL Capabilities:

`capabilityACL (enumCapabilityACL)`

Indicates the level of support for ACLs by the repository

- **`none`**: The repository does not support ACL services
- **`discover`**: The repository supports discovery of ACLs (getACL)
- **`manage`**: The repository supports discovery of ACLs AND applying ACLs (getACL and applyACL services)

See Section: Access Control

2.1.2 Related Repositories

In addition, the "Get Repository Information" service MAY return a list of other repositories that are *related* to the current repository. This list SHOULD be a subset of the list of repositories that are returned by the "Get Repositories" service.

For each of these repositories, a Repository Name, a Repository Identity, a URI, and a Repository Relationship Name are returned. The Relationship Name is opaque to CMIS. Suggested/example values for the “Relationship Name” include:

- “`self`”, “`peer`“, “`parent`”, “`child`”: For use by systems that support a logical hierarchy of repositories.
- “`replica`”, “`archive`”: For use by systems that support archival and replication.

2.1.3 Implementation Information

The “Get Repository Information” service MUST also return implementation information such as vendor name, product name, product version, version of CMIS that it supports, the root folder ID, and MAY include other implementation-specific information. The version of CMIS that the repository supports MUST be expressed as a Decimal that matches the specification version.

2.1.4 Repository Access Control Reporting

The repository’s access control semantics MUST be returned if the repository support either the `discover` or `manage` options reported by the `capabilityACL` optional capability. See the 2.8.5.2 AllowableActions Mapping section for more information.

2.2 Object

The entities managed by CMIS are modeled as typed ***Objects***. There are four base types of objects: ***Document Objects***, ***Folder Objects***, ***Relationship Objects***, and ***Policy Objects***.

- A *document object* represents a standalone information asset. Document objects are the elementary entities managed by a CMIS repository.
- A *folder object* represents a logical container for a collection of “file-able” objects, which include folder objects and document objects. Folder objects are used to organize file-able objects. Whether or not an object is file-able is specified in its object-type definition.
- A *relationship object* represents an instance of directional relationship between two objects.
- A *policy* object represents an administrative policy, which may be “applied” to one or more “controllablePolicy” objects. Whether or not an object is controllable is specified in its object-type definition. The support for policy objects is optional, and may be discovered via the “Get Types” service.

Document objects, folder objects, and policy objects are *independent* objects, in the sense that each object can persist independently as a standalone object in the repository. A relationship object, on the other hand, represents an explicit, application-maintained, instance of relationship between two independent objects. Therefore, relationship objects are semantically *dependent* objects. Additional object-types MAY be defined in a repository as subtypes of these base types. CMIS services are provided for the discovery of object types that are defined in a repository. However, object-type management services, such as the creation, modification, and deletion of an object type, are outside the scope of CMIS.

Every CMIS object has an opaque and immutable ***Object Identity*** (ID), which is assigned by the repository when the object is created. An ID uniquely identifies an object within a repository regardless of the type of the object. Repositories SHOULD assign IDs that are “permanent” – that is, they remain unchanged during the lifespan of the identified objects, and they are never reused or reassigned after the objects are deleted from the repository.

Every CMIS object has a set of named, but not explicitly ordered, ***Properties***. (However, a Repository SHOULD always return object properties in a consistent order.) Within an object, each property is uniquely identified by its name.

In addition, a document object MAY have a ***Content-Stream***, which may be used to hold a raw digital asset such as an image or a word-processing document. A repository MUST specify, in each object-type definition, whether document objects of that type MAY, SHALL, or SHALL NOT have a content-stream.

Document or folder objects MAY have one ***Access Control List*** (ACL), which – in addition to policies – controls access to the document or folder. An ACL represents a list of ***Access Control Entries*** (ACEs). An ACE in turn represents a permission being granted to a ***principal*** (a user, group, or something similar). Within the ACL of an object, each ACE is uniquely identified by the tuple *principal ID* and *permission name*

Properties and content-streams MAY NOT be shared between objects.

2.2.1 Property

A property MAY hold zero, one, or more typed data value(s). Each property MAY be *single-valued* or *multi-valued*. A single-valued property contains a single data value, whereas a multi-valued property contains an ordered list of data values of the same type. The ordering of values in a multi-valued property SHOULD be preserved by the repository.

If a value is not provided for a property, the property is in a "*value not set*" state. There is no "null" value for a property. Through protocol binding, a property is either not set, or is set to a particular value or a list of values.

A multi-valued property is either set or not set in its entirety. An individual value of a multi-valued property SHALL NOT be in an individual "value not set" state and hold a position in the list of values. An empty list of values SHALL NOT be allowed.

Every property is typed. The Property-type defines the data type of the data value(s) held by the property. CMIS specifies the following ***Property-Types***. They include the following data types defined by "XML Schema Part 2: Datatypes Second Edition" (W3C Recommendation, 28 October 2004, http://www.w3.org/TR/xmlschema-2/):

- `string` (xsd:string)
- `boolean` (xsd:boolean)
- `decimal` (xsd:decimal)
- `integer` (xsd:integer)
- `datetime` (xsd:dateTime)
- `uri` (xsd:anyURI)

The Precision for Decimal property-types is specified via its Property Definition.

In addition, the following Property-Types are also specified by CMIS:

- `id` (xsd:string)
- `xml` (xs:any)
- `html` (xs:any)
- `xhtml` (xs:any)

2.2.1.1 ID Property

An ID property holds a system-generated, read-only identitfier, such as an Object ID, an Object Type ID, a Repository ID, or a Version Series ID. (The ID Property-Type is NOT defined by xsd:id.) The lexical representation of an ID is an opaque string. As such, an ID cannot be assumed to be interpretable syntactically or assumed to be to be collate-able with other IDs, and can only be used in its entirety as a single atomic value. When used in a query predicate, an ID can only participate in an “equal” or a “not equal” comparison with a string literal or with another ID.

While all CMIS identities share the same Property-Type, they do not necessarily share the same address space. (So a particular repository implementation MAY use separate address spaces for different kinds of identity, and may perform different validity checks in a way that is outside the scope of the CMIS interfaces.)

Unless explicitly specified, ID properties NEED NOT maintain a referential integrity constraint. Therefore, storing the ID of one object in another object NEED NOT constrain the behavior of either object. A repository MAY, however, support referential constraint underneath CMIS if the effect on CMIS services remains consistent with an allowable behavior of the CMIS model. For example, a repository MAY return an exception when a CMIS service call violates an underlying referential constraint maintained by the repository. In that case, an error message SHOULD be returned to the application to describe the cause of exception and suggest a remedial action. The content of such messages is outside the scope of CMIS.

IDs in CMIS for documents usually refer to the specific document version. They can also refer to the Version Series if obtained from the `versionSeriesId` property.

2.2.1.2 XML Property

An XML property holds a well-formed Extensible Markup Language (XML) document.

2.2.1.3 HTML Property

An HTML property holds a valid document or fragment of Hypertext Markup Language (HTML) content.

2.2.1.4 XHTML Property

An XHTML property holds a valid document or fragment of Extensible Hypertext Markup Language (XHTML) content.

2.2.1.5 Property Attributes

Besides name, type, and whether single- or multi-valued, a property has other defining attributes. They are described in the 2.9 Object-Type section.

2.3 Document Object

Document objects are the elementary information entities managed by the repository.

Depending on its Object-type definition, a Document Object may be:

- **Version-able:** Can be acted upon via the Versioning Services (for example: *checkOut, checkIn)*.
- **File-able:** Can be filed in zero, one, or more than one folder via the Multi-filing services.
- **Query-able:** Can be located via the Discovery Services (*query*).
- **Control-able:** Can have Policies applied to it (see the “1.1 Policy Object” section.)
- **ACLControl-able:** Can have an ACL applied to it (see the “Access Control Lists” section).

Additionally, whether a Document object SHALL, MAY or SHALL NOT have a content-stream is specified in its object-type definition.

Note: When a document is versioned, each version of the document is a separate document object. Thus, for document objects, an object ID actually identifies a specific version of a document.

2.3.1 Content Stream

A content-stream is a binary string. Its maximum length is repository-specific. Each content-stream has a ***MIME Media Type***, as defined by RFC2045 and RFC2046 and registered with IANA (http://www.iana.org/assignments/media-types/). A content-stream's attributes are represented as properties of the content-stream's containing document object. There is no MIME-type-specific attribute or name directly associated with the content-stream outside of the document object.

A content-stream is unnamed. CMIS provides basic CRUD services for content-stream, using the ID of a content-stream's containing document object for identification. The "Set Content-Stream" service (*setContentStream*) either creates a new content-stream for a document object or replaces an existing content-stream. The "Get Content-Stream" service (*getContentStream*) retrieves a content-stream. The "Delete Content-Stream" service (*deleteContentStream*) deletes a content-stream from a document object. In addition, the "CreateDocument" and "Check-in" services MAY also take a content-stream as an optional input. A content stream MUST be specified if required by the type definition. These are the only services that operate on content-stream. The "Get Properties" and "Query" services, for example, do not return a content-stream.

"Set Content-Stream" and "Delete Content-Stream" services are considered modifications to a content-stream's containing document object, and will therefore change the object's *LastModificationDate* property upon successful completion. The ability to set or delete a content stream is controlled by the `capabilityContentStreamUpdateability` capability.

2.4 Folder Object

A folder object serves as the anchor for a collection of *file-able* objects. The folder object has an *implicit* hierarchical relationship with each object in its collection, with the anchor folder object being the ***Parent*** object and each object in the collection being a ***Child*** object. This implicit relationship has a specific containment semantics which SHALL be maintained by the repository with implicit referential integrity. (That is, there will never be a dangling parent-relationship or a dangling child-relationship. Furthermore, object A is a parent of object B if and only if object B is a child of object A.) This system-maintained implicit relationship is distinct from an *explicit* relationship which is instantiated by an application-maintained Relationship Object. (See the "2.5 Relationship Object" section.)

A folder object does not have a content-stream and is not version-able, although it is query-able and MAY be controllable and/or ACL controllable.

2.4.1 File-able Objects

A *file-able* object is one that MAY be "filed" into a folder. That is, it MAY be a child object of a folder object. The following list defines whether the base CMIS Object Types are file-able:

cmis:Folder

SHALL be file-able

cmis:Document

- If the "un-filing" optional capability is not supported by the Repository: SHALL be file-able
- If the "un-filing" optional capability is supported by the Repository: SHOULD be file-able

cmis:Relationship

SHALL NOT be file-able

cmis:Policy

MAY be file-able

2.4.1.1 Multi-filing/un-filing of Document and Policy Objects

The following list summarizes the number of parent folders that a Document or Policy Object should have.

If Multi-filing and un-filing capabilities are not supported, there MUST be 1 parent folder.

If Multi-filing is unsupported and un-filiing is supported, there MUST be 0 or 1 parent folders.

If Multi-filing is supported and un-filing is not supported, there MUST be 1 or more parent folders.

If Multi-filing is supported and un-filing is supported, there MUST be 0 or more parent folders.

A fileable object that has multiple parent folders is said to be ***multi-filed***. A fileable object that has no parent folder is said to be ***unfiled***.

2.4.1.2 Document Version Series and Filing

Since document objects are versionable, a document object's membership in a folder collection MAY be version-specific or version-independent. That is, the folder membership MAY be restricted to that particular version of the document or MAY apply to all versions of the document. Whether or not a repository supports version-specific filing is discoverable via the "Get Repository Information" service (*getRepositoryInfo*). When the child objects of a folder are retrieved, a specific version of a document is returned if the repository supports version-specific filing, and the latest version is returned if the repository does not support version-specific filing. Likewise, this version sensitivity in child-binding also affects the behavior of parent retrieval for a document object, as well as the scope of the IN_FOLDER() and IN_TREE() function calls in a query. For non-versionable fileable objects, their membership in a folder does not have version sensitivity.

2.4.1.3 Filing Restrictions by Object-Type

A folder collection's membership MAY be restricted by object type. Each folder object has a multi-valued *AllowedChildObjectTypeIDs* property, which specifies that only objects of these types are allowed to be its children. (These allowed object types SHOULD all be fileable.) If this property is "not set", then objects of any file-able type are allowed to be filed in the Folder. It is repository specific if subtypes of the types listed in the *AllowedChildObjectTypeIDs* property may be children of the folder.

If a repository does not support type constraint on folder membership, it MAY define this folder property as read-only, and keep the property value "not set" for all folder objects.

Because of these filing constraints, when a new folder object is created, an existing folder object SHALL be specified as its parent. When a file-able non-folder object is created, its parent folder SHALL be specified if the repository does not support the "un-filing" optional capability, and MAY be specified if the "un-filing" optional capability is supported. The Repository SHALL only allow creation of a file-able object in a folder if the object's Object-Type is specified as allowed in that folder as described above.

When a non-file-able object is created, a parent folder SHALL NOT be specified.

When a file-able object is deleted, it is removed from any folder collection in which the object is a member. In other words, when an object is deleted, all implicit parent-child relationships with the deleted object as a child cease to exist.

2.4.2 Folder Hierarchy

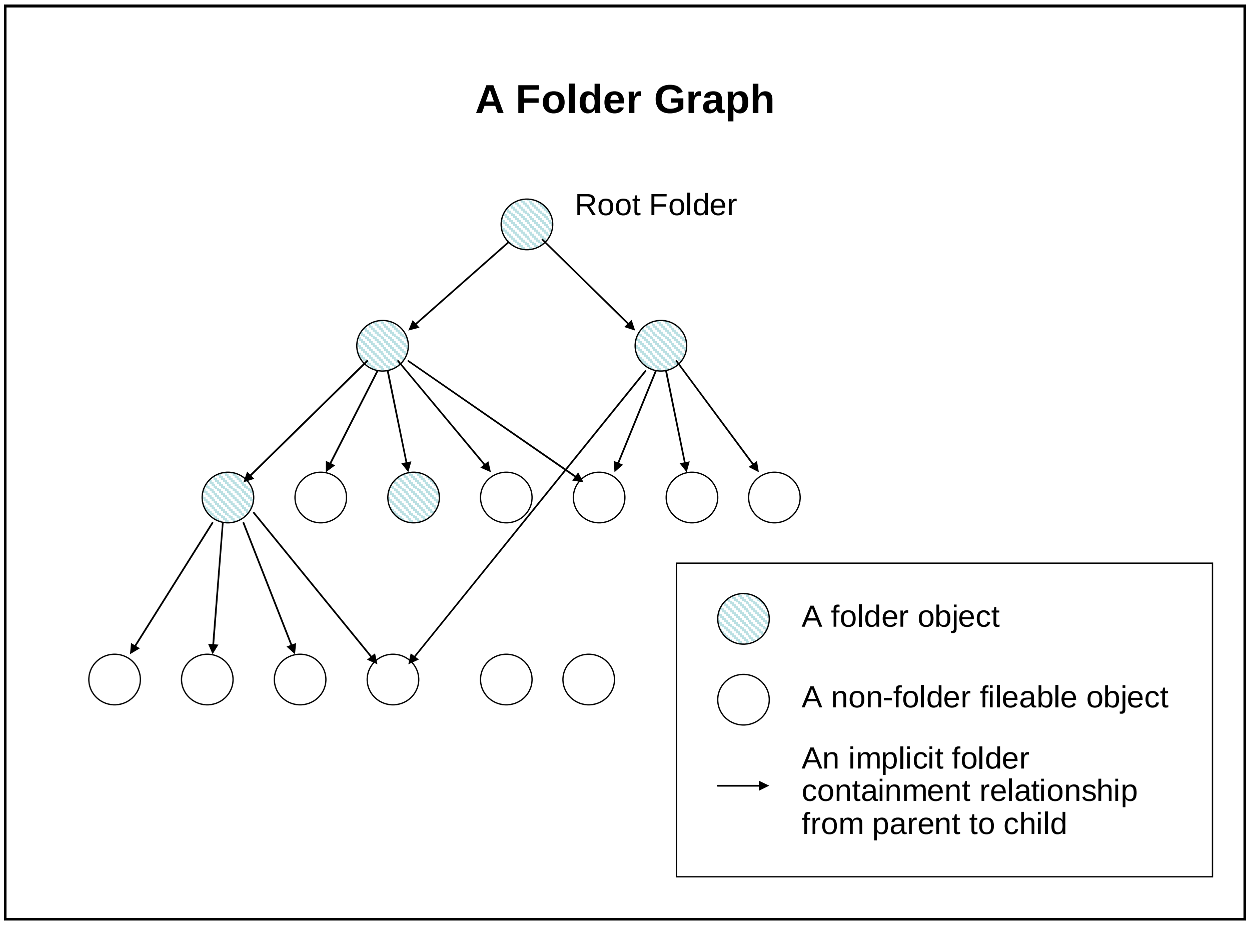
CMIS imposes the following constraints on folder objects:

- Every folder object, except for one which is called the ***Root Folder***, MUST have one and only one parent folder. The Root Folder does not have a parent.
- A cycle in folder containment relationships is not allowed. That is, a folder object cannot have itself as one of its descendant objects.
- A child object that is a folder object can itself be the parent object of other file-able objects.

With these constraints, the folder objects in a CMIS repository necessarily form a strict hierarchy, with the Root Folder being the root of the hierarchy.

The child objects of a given folder object, their child objects, and grandchild objects, etc., are called ***Descendant*** objects of the given folder object. Similarly, the parent object of a file-able object, and recursively the parent of the parent, etc., are called ***Ancestor*** objects of that file-able object. A folder object together with all its descendant objects are collectively called a ***Tree*** rooted at that folder object.

A non-folder object does not have any descendant object. Thus, a ***Folder Graph*** that consists of all fileable objects as nodes, and all the implicit folder containment relationships as directed edges from parent to child, is a directed acyclic graph, possibly with some disconnected (orphan) nodes. It follows that the tree rooted at any given folder object is also a directed acyclic graph, although a non-folder object in the tree MAY have ancestors that are not ancestors of the rooted folder.


A Folder Graph
Root Folder
A folder object
A non-folder fileable object
An implicit folder containment relationship from parent to child

Folder objects are handled using the basic CRUD services for objects, and the folder graph is traversed using the Navigation Services.

The ***Root Folder*** is a special folder such that it cannot be created, deleted, or moved using CMIS services. Otherwise, it behaves like any other folder object.

2.4.3 Paths

A folder hierarchy MAY be represented in a canonical notation such as path. For CMIS, a folder path is represented by:

- '/' for the root folder
- all paths start with the root folder.
- a set of the folder names separated by '/' in order of closest to the root.
    - Folder names are specified by the folders cmis: PathName property MUST not include the '/' separator character.

E.g., if folder A is under the root, and folder B is under A, then the path would be /A/B.

Paths do not apply to any other objects in CMIS except folders.

2.5 Relationship Object

A relationship object is semantically a *dependent* object. A relationship object SHALL NOT have a content-stream, and SHALL NOT be versionable, SHALL NOT be queryable, and SHALL NOT be fileable, although it MAY be controllable.

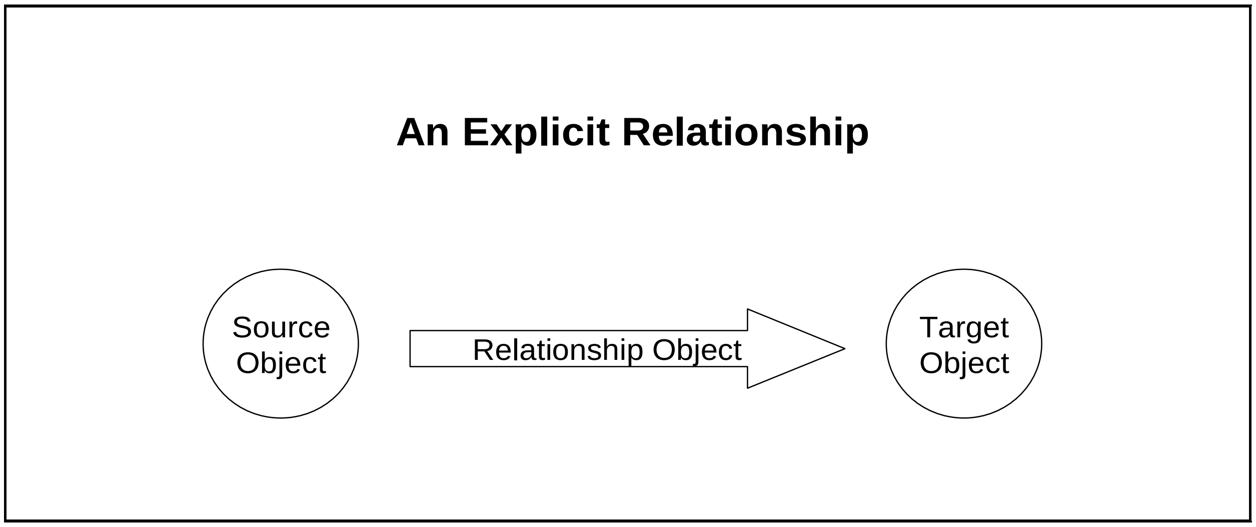
A ***Relationship Object*** instantiates an explicit, binary, directional, non-invasive, and typed relationship between a ***Source Object*** and a ***Target Object***. The source object and the target object MUST both be an independent objects, such as a document object, a folder object, or a policy object. Whether a policy object is allowed to be the source or target object of a relationship object is repository-specific.

The relationship instantiated by a relationship object is *explicit* since it is explicitly represented by an object and is explicitly managed by application.

This relationship is *non-invasive* in the sense that creating or removing this relationship SHALL NOT modify either the source or the target object. That is, it SHALL NOT require an update capability (or permission) on either object; SHALL NOT affect the versioning state of either object; and SHALL NOT change their "Last Modification Date".

The source object and the target object of a relationship MAY be the same object.

Explicit relationships can be used to create an arbitrary relationship graph among independent objects. Such a relationship graph is only structural in nature. No inheritance or transitive properties are attached to a relationship graph.



The notion of a source object and a target object of a relationship is used solely to indicate the direction of the relationship. No semantics or implementation bias is implied by this terminology.

The binding of a relationship object to a source document object or to a target document object MAY be either version-specific or version-independent. This version sensitivity is repository-specific, and is largely transparent to CMIS. An independent object MAY participate in any number of explicit relationships, as the source object for some and as the target object for others. Multiple relationships MAY exist between the same pair of source and target objects.

Referential integrity, either between the source object and the target object, or between the relationship object and the source or target object, is not specified. Therefore, creating an explicit relationship between two objects NEED NOT impose a constraint on any of the three objects, and removing a relationship or deleting either the source or the target object NEED NOT be restricted by such a constraint. If the source or the target object of a relationship is deleted, the repository MAY automatically delete the relationship object.

A repository MAY support referential integrity or other constraints underneath CMIS, either between the source object and the target object of a relationship, or between a relationship object and its source or target object, provided that the resulting effect on CMIS services is consistent with an allowable behavior of CMIS. For example, a repository MAY throw a generic exception when an underlying referential constraint is violated, and SHOULD return an error message of the cause of the exception. However the content of such a message is outside the scope of CMIS.

Like all CMIS objects, relationship objects are typed. Typing relationship allows them to be grouped, identified, and traversed by type id, and for properties to be defined for individual relationship types.

Additionally, a relationship object-type MAY specify that only Objects of a specific Object-Type (or one of a set of specific Object-Tyeps) can participate as the source object or target object for relationship objects of that type. If no such constraints are specified , then an independent object of any type MAY be the source or the target of a relationship object of that type.

Relationship objects are created/retrieved/updated/deleted using the CMIS Object Services (*createRelationship, getProperties, updateProperties, deleteObject).*

When a relationship object is created, the source object and the target object MUST already exist.

Relationship Objects are retrieved using the “Relationships” service (*getRelationships),*which can be used to return a set of relationship objects in which a given independent object is identified as the source or target object, according to the binding semantics maintained by the repository (i.e., either a version restricted to relationships of a given type and/or in a particular direction (i.e., where the given object is the source, the target, or either).

When a relationship object is retrieved, its source object or target object MAY no longer exist, since referential integrity NEED NOT be maintained by a repository.

When a relationship object is deleted, the source and the target objects are left untouched.

In addition to object CRUD services, a "Get Relationships" service (*getRelationships*) may be used to return a set of relationship objects in which a given independent object is identified as the source or the target object, according to the binding semantics maintained by the repository (i.e., either a version-specific or a version-independent binding as described above).

2.6 Policy Object

A policy object represents an administrative policy that can be enforced by a repository, such as an Access Control List (ACL) or a retention management policy. CMIS 1.0 does not specify what kinds of administrative policies that are specifically supported, nor attempts to model administrative policy of any particular kind. Only a base object type is specified for policy objects. Each policy object holds the text of an administrative policy as a repository-specific string, which is opaque to CMIS and which may be used to support policies of various kinds. (For CMIS 1.0, the use case is primarily access control.) A repository may create subtypes of this base type to support different kinds of administrative policies more specifically. The support for policy objects is optional. If a repository does not support policy objects, the policy base object type SHOULD NOT be returned by a "Get Types" service call.

Aside from allowing an application to create and maintain policy objects, CMIS allows an application to "apply" a policy to an object, and to remove an applied policy from an object. An object to which a policy may be applied is called a *controllable* object. A policy MAY be applied to multiple controllable objects. Conversely, a repository MAY allow multiple policies applied to a controllable object. (A repository may, for example, impose constraints such as only one policy of each kind can be applied to an object.) Whether or not an object is controllable is specified by the object's type definition. Applying a policy to an object is to place the object under the control of that policy (while the object may also be under the control of other policies at the same time), and removing an applied policy from one of its controlled objects is to remove the corresponding control from that object. This control may change the state of the object, may impose certain constraints on service calls operating on this object, or may cause certain management actions to take place. The effect of this control, when this effect takes place, and how this control interacts with other controls, are repository-specific. Only directly/explicitly applied policies are covered by CMIS 1.0. Indirectly applying policy to an object, e.g. through inheritance, is outside the scope of CMIS 1.0.

A policy object does not have a content-stream and is not versionable. It may be fileable, queryable, or controllable. Policy objects are handled using the basic CRUD services for objects. If a policy is updated, the change may alter the corresponding control on objects that the policy is currently applied to. If a controlled object is deleted, all the policies applied to that object, if there is any, are removed from that object. A policy object that is currently applied to one or more controllable objects can not be deleted. That is, there is an implicit referential constraint from a controlled object to its controlling policy object(s). Besides the basic CRUD services, the "Apply Policy" (*applyPolicy*) and the "Remove Policy" (*removePolicy*) services may be used to apply a policy object to a controllable object and respectively to remove an applied policy from one of its controlled objects. In addition, the "Get Applied Policies" (*getAppliedPolicies*) service may be used to obtain the policy objects that are currently applied to a controllable object.

2.7 Renditions

Some ECM repositories provide a facility to retrieve alternative versions of a document. These alternative versions are known as renditions. This could apply to a preview case which would enable the client to preview the content of a document without needing to download the full content. Previews are generally reduced fidelity representations such as thumbnails. Renditions can take on any general form, such as a PDF version of a word document.

A document or folder may support zero or more renditions. Renditions are not CMIS object types. They are bound to a document or folder object like properties.

The server is responsible for determining the number and types of renditions present for a given document / folder. The server is responsible for the availability of a document / folder rendition. A rendition may not be immediately available after checkin or document creation. Renditions are specific to the version of the document and may differ between document versions.

Each rendition consists of a content stream of a given mime-type containing the alternative representation. Each rendition should have an advisory label to assist identifying the kind of rendition. Additionally each rendition may provide additional metadata.

2.7.1 Rendition Attributes

A rendition consists of the following attributes:

`streamId` ID

Identifies the content stream of the rendition.

`mimeType` String

The mimetype of the rendition content stream.

`length` Integer

The length of the content stream in bytes.

`title` String

Human readable information about the rendition.

`kind` String

A categorization String associated with the rendition.

`metadata` <`RenditionMetadata` list>

A list of repository generated metadata about the rendition.

`renditionDocumentId` ID

The ObjectID of the rendition document if this rendition is represented as a document by the repository.

2.7.2 Rendition Kind

A Rendition may be categorized via its `kind`. The repository is responsible for assigning kinds to Renditions, including custom kinds. A repository kind does not necessarily identify a single Rendition for a given Object.

CMIS defined kinds are distinguished from custom kinds by the CMIS-specific prefix **CMIS**.

CMIS defines the following kind:

- **`CMIS.Thumbnail`**: A rendition whose purpose is to a provide a preview of the document without requiring the client to download the full document content stream. Thumbnails are generally reduced fidelity representations.

2.7.3 Rendition Metadata

Rendition meta-data describes extra characteristics of the Rendition primarily to allow a client to decide which Rendition to choose in a particular use-case.

CMIS defines the following Rendition (optional) meta-data.

- `height` : typically used for 'image' renditions (expressed as pixels)
- `width` : typically used for 'image' renditions (expressed as pixels)

A repository may introduce custom meta-data for each Rendition. Examples include image resolution and video time length.

2.7.4 Rendition as a Document

A repository may choose to provide a separate Document (conceptually known as a Rendition Document) which represents and complements the Rendition. The rendition `renditionDocumentId` contains the Object Id of the Rendition Document (if one exists), otherwise `renditionDocumentId` is null.

The referential integrity of `renditionDocumentId` should be maintained. If provided, the Object Id should refer to a Document that exists.

Rendition Documents act like any other Document, therefore may be filed in a Folder, support access control, related etc. It is repository specific as to whether the Rendition Document can be updated or deleted. Repositories that support deletion of Rendition Documents should maintain the referential integrity of `renditionDocumentId` (e.g. setting its value to null, or removing the Rendition all together), but how this is achieved is repository specific.

2.8 Access Control

As outlined above for "Policy Object", policy objects represent administrative policies, usually expressing access control constraints like "*only users with security clearance 2 are allowed to view the documents with security level HIGH*". Policies are basic CMIS object types.

Access Control Lists (ACLs) are for user scenarios, like "*I want to allow user A from my team to work with the documents within this folder*". Unlike Policies, ACLs are not independent CMIS objects, they are bound to a document or folder object.

Documents and folders MAY be *controllablePolicy* (by Policies) or *controllableACL* (by ACLs) or both. This implies that a client MUST NOT assume that a document's or folder's ACL can be used to exactly compute the allowable actions. A client can use *getAllowableActions* to check if a given user is allowed to perform a specific operation at a given time.

2.8.1 Discovering and Managing ACLs

ACLs are discoverable for a client via getACL. ACLs can be changed by a client via applyACL. The enum capabilityACL returned by getRepositoryInfo indicates the level of support: either getACL ("**discover"**) only, or both getACL and applyACL ("**manage**").

2.8.2 ACL Supported Permissions

A repository can support either or both of a set of CMIS defined permissions or its own set of repository specific permissions.

The getACL service allows the requestor to specify that the result be expressed using only the CMIS defined permissions. Without this restriction, the response may include, or be solely expressed in repository specific permissions. The applyACL service permits either CMIS permissions or repository permissions, or a combination of both, to be used.

For the CMIS permissions a repository's service-permissions mappings are discoverable through getRepositoryInfo, Service-permissions mappings are not discoverable for the repository permissions.

For full details of each of these features refer to the ACL Capabilities, getACL and applyACL sections.

2.8.3 ACL, ACE, Principal, and Permission

An ***ACL*** is a list of ***Access Control Entries*** (ACEs) and MAY hold zero or more ACEs.

An ***ACE*** holds:

- one ***Principal:*** A principal represents a user management object, e.g. a user, group, or role. It holds:
    - one **String** with the ***principalid***.
    - Repository-specific information MAY be returned as well.
- one **String** with the name of the ***permission****.*
- a **Boolean** flag ***direct***, which indicates if the ACE is directly assigned to the object itself. FALSE, if the ACE is somehow derived from some other ACE or Policy applied to another object.

The ACEs for an ACL MAY be “shared“, in that sense that adding or removing an ACE to or from an ACL for an object MAY have side-effects for the ACLs of other objects.

2.8.4 Basic CMIS Permissions

There are four basic permissions predefined by CMIS:

- **cmis:BasicPermission.Read**: to be used to express “permission to read”. A Repository SHOULD express the permission for reading properties AND reading content with this permission.
- **cmis:BasicPermission.Write**: to be used to express “permission to write”. SHOULD be used to express permission to write properties and content of an object. MAY include other basic CMIS permissions.
- **cmis:BasicPermission.Delete**: to be used to express “permission to delete”. SHOULD be used to express permission to delete, deleteTree, and deleteContent of an object, MAY include other basic CMIS permissions.
- **cmis:BasicPermission.All**: SHOULD be used to express all the permissions of a repository. SHOULD include all other basic CMIS permissions.

How these basic permissions can be mapped to the allowable actions is repository specific. However, the actual repository semantics for the basic permissions with regard to allowable actions can be discovered by the *mappings* parameter returned by *getRepositoryInfo* (see below).

Repositories MAY extend this set with repository-specific permissions

2.8.5 ACL Capabilities

Whether a repository supports ACLs at all, may be discovered via *capabilityACL* returned by *getRepositoryInfo* (see Optional Capabilities). If *capabilityACL* is `none`, ACLs are not supported by the repository.

If *capabilityACL* is `discover` or `manage`, additional information about the repositories permission model and how changes to ACL are handled, can be discovered via the *getRepositoryInfo* service:

- **Enum supportedPermissions**: specifies which types of permissions are supported.
  - basic: indicates that the CMIS Basic permissions are supported.
  - repository: Indicates that repository specific permissions are supported.
  - both: indicates that both CMIS basic permissions and repository specific permissions are supported.
- **<Array> Enum ACLPropagation:** specifies, how non-direct ACEs can be handled by the repository using the following values (see *applyACL*):
  - **object-only** indicates, that the repository is able to apply ACEs to a document or folder, without changing the ACLs of other objects. I.E. the repository is able to "break" the dependency for non-direct ACEs when requested by the client.
  - **propagate**: indicates that the ACEs is to be applied to the given object and all "inheriting" objects – i.e. with the intended side effect that all objects which somehow share the provided security constraints should be changed accordingly.
  - **repository-determined** indicates, that the repository has its own mechanism of computing how changing an ACL for an object influences the non-direct ACEs of other objects.
- **<Array> PermissionDefinition repositoryPermissions:** is a list with names and descriptions of the supported repository specific permissions, if allowedPermissions is repository or both.

- **<Array> PermissionMapping mappings:** contains a list with mappings for the basic CMIS permissions to allowed actions (see *getAllowableActions*) .

2.8.5.1 Repository-Specific Permissions

The list of permission definitions returned by *getRepositoryInfo* lists the repo*sitory-specific per*missions a repository supports.

A PermissionDefinition holds:

- **String permission:** the (technical) name of the permission (unique within the list of permission definitions).
- **(Optional) String description:** an optional description of the permission, that should be used as the permission's name to be presented to the user.

The list of permission definitions MUST contain all the repository-specific permissions the repository supports

2.8.5.2 AllowableActions Mapping

The permission mapping returned by *getRepositoryInfo* provides a map of basic CMIS permissions to CMIS objects in order to grant or enable a specific CMIS action on *getAllowableActions*. The set of allowable actions for an object is represented as a set of Boolean values. This section defines both the AllowableActions and PermissionsMapping together.

A PermissionMapping holds only these two values. The operation and operand are implicitly defined by the action and are not part of the PermissionMapping.).

- **String key:** a unique name representing the allowable action plus one of the operands for this action that the mapping applies to. The “key” is built by using the name of the allowable action (see *getAllowableActions*) and the name of the operand (without the ID or prefix, see *Services* section).
- **String permission:** the name of the permission that needs to be applied to the parameter as indicated by the “key”.

Repositories SHOULD use the following mapping. Repositories MAY require stronger permissions. An AllowableAction may be listed more than once if there are multiple operands it applies to. For each Allowable Action the following information is given:

Base Object:	The base object types for which the allowable action MAY be TRUE.
Action:	The description and name of action the AllowableAction enables.
Operand:	The object the permission applies to.
Key:	The permission mapping key.
Permission:	The permission value.

Navigation Services:

canGetDescendants

Base Object:	cmis:Folder
Action:	Can get the descendants of the folder (`getDescendants`)
Operand:	cmis:Folder
Key:	canGetDescendants.Folder
Permission:	Read

canGetChildren

Base Object:	cmis:Folder
Action:	Can get the children of the folder (`getChildren`)
Operand:	cmis:Folder
Key:	canGetChildren.Folder
Permission:	Read

canGetFolderParent

Base Object:	cmis:Folder
Action:	Can get the parent/ancestor folder(s) of the folder (`getFolderParent`)
Operand:	cmis:Folder
Key:	canGetFolderParent.Folder
Permission:	Read

canGetObjectParents Object

Base Object:	cmis:Document, cmisFolder, cmisPolicy
Action:	Can get the parent folders of the object. (`getObjectParents`)
Operand	Object
Key:	canGetObjectParents.Object
Permission:	Read

Object Services:

canCreateDocument

Base Object:	cmis:Folder
Action:	Can create a cmis:Document Object in the folder (`createDocument`)
Operand:	Type
Key:	canCreateDocument.Type
Permission:	Write

canCreateDocument

Base Object:	cmis:Folder
Action:	Can create a cmis:Document Object in the folder (`createDocument`)
Operand:	Folder
Key:	canCreateDocument.Folder
Permission:	Read

canCreateFolder

Base Object:	Type
Action:	Can create a cmis:Folder Object in the folder (`createFolder`)
Operand:	createFolder
Key:	canCreateFolder.Type
Permission:	Write

canCreateFolder

Base Object:	cmis:Folder
Action:	Can create a cmis:Folder Object in the folder (`createFolder`)
Operand:	createFolder
Key:	canCreateFolder.Folder
Permission:	Read

canCreateRelationship

Base Object:	cmis:Document, cmis:Folder
Action:	Can create a Relationship in which this Object is a source/target (`createRelationship`)
Operand:	Type
Key:	canCreateRelationship.Type
Permission:	Write

canCreateRelationship

Base Object:	cmis:Document, cmis:Folder
Action:	Can create a Relationship in which this Object is a source/target (`createRelationship`)
Operand:	Source

Key:	canCreateRelationship.Source
Permission:	Read

canCreateRelationship

Base Object:	cmis:Document, cmis:Folder
Action:	Can create a Relationship in which this Object is a source/target (`createRelationship`)
Operand:	Target
Key:	canCreateRelationship.Target
Permission:	Read

canCreatePolicy

Base Object:	cmis:Folder
Action:	Can create a Policy Object in the folder (`createPolicy`)
Operand:	Type
Key:	canCreatePolicy.Type
Permission:	Repository Specific

canGetProperties

Base Object:	cmis:Document, cmis:Folder, cmis:Relationship, cmisPolicy
Action:	Can read the properties of this object (`getProperties`)
Operand:	Object
Key:	canGetProperties.Object
Permission:	Read

canGetRenditions

Base Object:	cmis:Document
Action:	Can retrieve the renditions of this object (`getRenditions`)
Operand:	Object
Key:	canGetRenditions.Object
Permission:	Read

canGetContentStream

Base Object:	cmis:Document
Action:	Can get the content stream for the Document object (`getContentStream`)
Operand:	Object
Key:	canGetContentStream.Object
Permission:	Read

canUpdateProperties

Base Object:	cmis:Document, cmis:Folder, cmis:Relationship, cmisPolicy

Action:	Can update the properties of this object (`updateProperties`)
Operand:	Object
Key:	canUpdateProperties.Object
Permission:	Write

canMoveObject

Base Object:	cmis:Document, cmis:Folder, cmisPolicy
Action:	Can move the object (`moveObject`)
Operand:	Object
Key:	canMoveObject.Object
Permission:	Write

canMoveObject

Base Object:	cmis:Document, cmis:Folder, cmisPolicy
Action:	Can move the object (`moveObject`)
Operand:	Target
Key:	canMoveObject.Target
Permission:	Read

canMoveObject

Base Object:	cmis:Document, cmis:Folder, cmisPolicy
Action:	Can move the object (`moveObject`)
Operand:	Source
Key:	canMoveObject.Source
Permission:	Read

canDeleteObject

Base Object:	cmis:Document, cmis:Folder, cmis:Relationship, cmisPolicy
Action:	Can delete this object (`deleteObject`)
Operand:	Object
Key:	canDelete.Object
Permission:	Delete

canDeleteObject

Base Object:	cmis:Folder
Action:	Can delete this object (`deleteTree`)
Operand:	Folder
Key:	canDelete.Folder
Permission:	Delete

canDeleteObject

Base Object:	cmis:Document

Action:	Can delete this object (`deleteAllVersions`)
Operand:	VersionSeries
Key:	canDelete.VersionSeries
Permission:	Delete

canSetContentStream

Base Object:	cmis:Document
Action:	Can set the content stream for the Document object (`setContentStream`)
Operand:	cmis:Document
Key:	canSetContentStream.Document
Permission:	Write

canDeleteContentStream

Base Object:	cmis:Document
Action:	Can delete the content stream for the Document object (`deleteContentStream`)
Operand:	cmis:Document
Key:	canDeleteContentStream.Document
Permission:	Delete

Mutli-filing Services:

canAddObjectToFolder

Base Object:	cmis:Document, cmis:Policy
Action:	Can file the document in a folder (`addObjectToFolder`)
Operand:	Object
Key:	canAddObjectToFolder.Object
Permission:	Write

canAddObjectToFolder

Base Object:	cmis:Document, cmis:Policy
Action:	Can file the document in a folder (`addObjectToFolder`)
Operand:	cmis:Folder
Key:	canAddObjectToFolder.Folder
Permission:	Read

canRemoveObjectFromFolder

Base Object:	cmis:Document, cmis:Policy
Action:	Can unfile the document in a folder (`removeObjectFromFolder`)
Operand:	Object
Key:	canRemoveObjectFromFolder.Object
Permission:	Write

canRemoveObjectFromFolder

Base Object:	cmis:Document, cmis:Policy
Action:	Can unfile the document in a folder (`removeObjectFromFolder`)
Operand:	cmis:Folder
Key:	canRemoveObjectFromFolder.Folder
Permission:	Read

Versioning Services:

canCheckOut

Base Object:	cmis:Document
Action:	Can check out the Document object (`checkOut`)
Operand:	cmis:Document
Key:	canCheckOut.Document
Permission:	Write

canCancelCheckOut

Base Object:	cmis:Document
Action:	Can cancel the check out the Document object (`cancelCheckOut`)
Operand:	cmis:Document
Key:	canCancelCheckOut.Document
Permission:	Write

canCheckIn

Base Object:	cmis:Document
Action:	Can check in the Document object (`checkIn`)
Operand:	cmis:Document
Key:	canCheckIn.Document
Permission:	Write

canGetAllVersions

Base Object:	cmis:Document
Action:	Can get the version series for the Document object (`getAllVersions`)
Operand:	VersionSeries
Key:	canGetAllVersions.VersionSeries
Permission:	Read

Relationship Services:

canGetRelationships

Base Object:	cmis:Document, cmis:Folder, cmisPolicy
Action:	Can get the relationship in which this object is a source/target (`getRelationships`)
Operand:	Object
Key:	canGetRelationships.Object
Permission:	Read

Policy Services:

canApplyPolicy

Base Object:	cmis:Document, cmis:Folder
Action:	Can apply a policy to the Object (`applyPolicy`)
Operand:	Object
Key:	canApplyPolicy.Object
Permission:	Repository Specific

canAddPolicy

Base Object:	cmis:Document, cmis:Folder
Action:	Can apply a policy to the Object (`applyPolicy`)
Operand:	cmis:Policy
Key:	canAddPolicy.Policy
Permission:	Read

canRemovePolicy

Base Object:	cmis:Document, cmis:Folder
Action:	Can remove a policy from the Object (`removePolicy`)
Operand:	Object
Key:	canRemovePolicy.Object
Permission:	Repository Specific

canRemovePolicy

Base Object:	cmis:Document, cmis:Folder
Action:	Can remove a policy from the Object (`removePolicy`)
Operand:	cmis:Policy
Key:	canRemovePolicy.Policy
Permission:	Read

canGetAppliedPolicies

Base Object:	cmis:Document, cmis:Folder
Action:	Can get the list of Policies applied to the Object (`getAppliedPolicies`)
Operand:	Object

Key:	canGetAppliedPolicies.Object
Permission:	Read

ACL Services:

canGetACL

Base Object:	cmis:Document, cmis:Folder, cmis:Relationship, cmis:Policy
Action:	Can get ACL for object (`getACL`)
Operand:	Object
Key:	canGetACL.Object
Permission:	Read

canApplyACL

Base Object:	cmis:Document, cmis:Folder, cmis:Relationship, cmis:Policy
Action:	Can apply ACL to this object (`applyACL`)
Operand:	Object
Key:	canApplyACL.Object
Permission:	All

2.9 Object-Type

An ***Object-Type*** defines a fixed and non-hierarchical set of properties ("schema") that all objects of that type have. This schema is used by a repository to validate objects and enforce constraints, and is also used by a user to compose object-type-based (structured) queries. All CMIS objects are strongly typed. Incidental properties that are not prescribed in an object's type definition (such as "residual properties" in JCR, or "dead properties" in WebDAV) are not modeled in CMIS. If a property not specified in an object's object-type definition is supplied by an application, an exception SHOULD be thrown.

Each object-type is uniquely identified within a repository by a system-assigned and immutable ***Object-Type Identity***, which is of type ID.

A CMIS repository SHALL expose exactly one collection of Object-Types via the "Repository" services *(getTypes, getTypeDefinition).*

While a repository MAY define additional object types beyond the CMIS Base Object-Types, these Object Types SHALL NOT extend or alter the behavior or semantics of a CMIS service (for example, by adding new services). A repository MAY attach additional constraints to an object-type underneath CMIS, provided that the effect visible through the CMIS interface is consistent with the allowable behavior of CMIS.

2.9.1 Object-Type Hierarchy and Inheritance

Hierarchy and ***Inheritance*** for Object-Types are supported by CMIS in the following manner:

- A CMIS repository SHALL have exactly four base types:
    - *cmis:Document* object type
    - *cmis:Folder* object type
    - *cmis:Relationship* object type

    - *cmis:Policy* object type
- Additional root types SHALL NOT exist. Additional object types MAY be defined as sub-types or descendant types of these four root types.
- A ***Base Type*** does not have a parent type.
- A non-base type has one and only one parent type. An object-type's ***Parent Type*** is a part of the object-type definition.
- An object-type definition includes a set of object-type attributes (e.g. Fileable, Queryable, etc.) and a property schema that will apply to Objects of that type.
    - There is no inheritance of object-type attributes from a parent object-type to its sub-types.
- The properties of a CMIS base type SHALL be inherited by its descendant types.
- A ***Child-type*** whose immediate parent is NOT its base type SHOULD inherit all the property definitions that are specified for its parent type. In addition, it MAY have its own property definitions.
    - If a property is NOT inherited by a subtype, the exhibited behavior for query SHALL be as if the value of this property is "not set" for all objects of this sub-type.
- The scope of a query on a given object-type is automatically expanded to include all the ***Descendant Types*** of the given object type with the attribute `includedInSuperTypeQuery` equals TRUE. This was added for synthetic types as well as to support different type hierarchies that are not necessarily the same as CMIS. Only the properties of the given object type, including inherited ones, SHALL be used in the query. Properties defined for its descendant types MAY NOT be used in the query, and CAN NOT be returned by the query.
    - If a property of its parent type is not inherited by this type, the property SHALL still appear as a column in the corresponding virtual table in the relational view, but this column SHALL contain a SQL NULL value for all objects of this type. (See the "2.11 Query" section.)

A repository that does not support Object-type inheritance natively MAY define all additional object types as immediate sub-types of these Base Object-types. In that case, the type hierarchy is essentially flat.

2.9.2 Object-Type Attributes

2.9.2.1 Attributes common to ALL Object-Type Definitions

All ***Object-Type Definitions*** SHALL contain the following ***attributes***:

`id` ID

This opaque attribute uniquely identifies this object type in the repository.

`localName` String

This attribute represents the underlying repository's name for the object type. This field is opaque and has no uniqueness constraint imposed by this specification.

`localNamespace` String

This optional attribute allows repositories to represent an internal namespace for represents the underlying repository's name for the object type.

`queryName` String

Used for query operations on object types. This is an opaque String with limitations. The string should not contain whitespace, the period (.) character or the open "(" or close ")" parenthesis characters.

`displayName` String

Used for presentation by application

`baseTypeId` Enum

A value that indicates whether the base type for this Object-Type is the Document, Folder, Relationship, or Policy base type.

`parentId` ID

The ID of the Object-Type's immediate parent type.

It SHALL be "not set" for a base type.

`description` String

Description of this object type, such as the nature of content, or its intended use. Used for presentation by application.

`creatable` Boolean

Indicates whether new objects of this type MAY be created. If the value of this attribute is FALSE, the repository MAY contain objects of this type already, but SHALL NOT allow new objects of this type to be created.

`fileable` Boolean

Indicates whether or not objects of this type are file-able.

`queryable` Boolean

Indicates whether or not this object type is queryable. A non-queryable object type is not visible through the relational view that is used for query, and can not appear in the FROM clause of a query statement.

`controllablePolicy` Boolean

Indicates whether or not objects of this type are controllable via policies. Policy objects can only be applied to controllablePolicy objects.

`fulltextIndexed` Boolean

Indicates whether objects of this type are full-text indexed for querying via the CONTAINS() query predicate.

`includedInSupertypeQuery` Boolean

Indicates whether this type and its subtypes appear in a query of this type's ancestor types.

For example: if Invoice is a sub-type of Document, if this is TRUE on Invoice then for a query on Document type, instances of Invoice will be returned if they match.

If this attribute is FALSE, no instances of Invoice will be returned even if they match the query.

`controllableACL`	Boolean

This attribute indicates whether or not objects of this type are controllable by ACL's. Only objects that are controllableACL can have an ACL (see Access Control Lists).

2.9.2.2 Attributes specific to Document Object-Types

The following Object ***attributes*** SHALL only apply to Object-Type definitions whose baseType is the cmis:Document Object-Type, in addition to the common attributes specified above:

`versionable`	Boolean

Indicates whether or not objects of this type are version-able. (See 2.10 Versioning section.)

`contentStreamAllowed`	Enum

A value that indicates whether a content-stream MAY, SHALL, or SHALL NOT be included in objects of this type. Values:

- **`notallowed`***:* A content-stream SHALL NOT be included
- **`allowed`***:* A content-stream MAY be included
- **`required`***:* A content-stream SHALL be included (i.e. SHALL be included when the object is created, and SHALL NOT be deleted.)

2.9.2.3 Attributes specific to Relationship Object-Types

The following Object ***attributes*** SHALL only apply to Object-Type definitions whose baseType is the cmis:Relationship Object-Type, in addition to the common attributes specified above:

`allowedSourceTypes`	String (mulit-valued)

A list of object type IDs, indicating that the source object of a relationship object of this type SHALL only be one of the types listed.

If this attribute is "not set", then the source object MAY be of any type.

`allowedTargetTypes`	String (mulit-valued)

A list of object type IDs, indicating that the target object of a relationship object of this type SHALL only be one of the types listed.

If this attribute is "not set", then the target object MAY be of any type.

2.9.3 Object-Type Property Definitions

Besides these object type attributes, an object type definition SHOULD contain inherited property definitions and zero or more additional property definitions. All the properties of an object, including inherited properties, SHALL be retrievable through the "get" services, and MAY appear in the SELECT clause of a query.

2.9.3.1 Property Types

Property types are defined in section 2.2.1 Property.

2.9.3.2 Attributes common to ALL Object-Type Property Definitions

All ***Object-Type Property Definitions*** SHALL contain the following ***attributes***:

`id` ID

This opaque attribute uniquely identifies the property in the repository.

`localName` String

This attribute represents the underlying repository's name for the object type. This field is opaque and has no uniqueness constraint imposed by this specification.

`localNamespace` String

This optional attribute allows repositories to represent an internal namespace for represents the underlying repository's name for the object type.

`queryName` String

Used for query operations on properties. This is an opaque String with limitations. The string should not contain whitespace, the period (.) character or the open "(" or close ")" parenthesis characters.

`displayName` String (optional)

Used for presentation by application.

`description` String

This is an optional attribute containing a description of the property

`propertyType` Enum

This attribute indicates the type of this property. It MUST be one of the allowed property types. (See the "2.2.1 Property" section.)

`cardinality` Enum

Indicates whether the property can have "zero or one" or "zero or more" values.

Values:

- **`single`**: Property can have zero or one values (if property is not required), or exactly one value (if property is required)
- **`multi`**: Property can have zero or more values (if property is not required), or one or more values (if property is required).

Repositories SHOULD preserve the ordering of values in a multi-valued property. That is, the order in which the values of a multi-valued property are returned in "get" operations SHOULD be the same as the order in which they were supplied during previous "create/update" operation.

`updatability` Enum

Indicates under what circumstances the value of this property MAY be updated.

Values:

- **`readonly`:** The value of this property SHALL NOT ever be set directly by an application. It is a system property that is either maintained or computed by the repository.
    - The value of a readOnly property MAY be indirectly modified by other repository interactions (for example, calling “updateProperties” on an object will change the object’s last modified date, even though that property cannot be directly set via an updateProperties() service call.)
- **`readwrite`:** The property value can be modified using the *updateProperties* service.
- **`whencheckedout`:** The property value SHALL only be update-able using a “private working copy” Object.
    - I.e. the update is either made on a “private working copy” object or made using a “check in” service.

`inherited` Boolean

Indicates whether the property is inherited from the parent-type or it is explicitly defined for this object-type.

`required` Boolean

If TRUE, then the value of this property SHALL never be set to the “not set” state when an object of this type is created/updated.

If a value is not provided, then the default value defined for the property SHALL be set. If no default value is provided and no default value is defined, the repository SHALL throw an exception.

A property definition SHOULD never state that a property has a “required” value of TRUE and an updatability value of “readonly”.

`queryable` Boolean

Indicates whether or not the property MAY appear in a CMIS query statement.

This attribute SHALL have a value of FALSE if the Object-type’s attribute for “Queryable” is set to FALSE.

(Note: “Queryable” has a different meaning for object type and for property. The former pertains e FROM clause and the latter pertains to the WHERE clause.)

`orderable` Boolean

Indicates whether the property can appear in the ORDER BY clause of a SQL SELECT statement.

This property SHALL be FALSE for any property whose cardinality is “multi”.

`choices` <PropertyChoiceType list> (multi-valued)

Indicates an explicit set of values allowed for this property.

If this attribute is “not set”, then any valid value for this property based on its type may be used.

Each choice is includes a displaName, an optional value and index. The displayName is used for presentation purpose. The value will be stored in the property when selected. The index provides guidance for ordering of displayNames when presented.

Choices MAY be hierarchically presented. In this case, the index orders siblings in the choice hierarchy; if a choice has no siblings then the index may be omitted; if a choice has siblings then the index is required on all siblings.

`openChoice` Boolean

This attribute is only applicable to properties that provide a value for the “Choices” attribute.

If FALSE, then the data value for the property SHALL only be one of the values specified in the “Choices” attribute. If TRUE, then values other than those included in the “Choices” attribute may be set for the property.

`defaultValue` <PropertyType>

The value that the repository SHALL set for the property if a value is not provided by an application when the object is created.

If no default value is specified and an application creates an object of this type without setting a value for the property, the repository SHALL attempt to store a “value not set” state for the property value. If this occurs for a property that is defined to be required, then the creation attempt SHALL throw an exception.

2.9.3.3 Attributes specific to Integer Object-Type Property Definitions

The following Object ***attributes*** SHALL only apply to Property-Type definitions whose *propertyType* “Integer”, in addition to the common attributes specified above. A repository MAY provide additional guidance on what values can be accepted. If the following attributes are not present the repository behavior is undefined and it MAY throw an exception if a runtime constraint is encountered.

`minValue` Integer

The minimum value allowed for this property.

If an application tries to set the value of this property to a value lower than minValue, the repository SHALL throw a ConstraintViolation exception.

`maxValue` Integer

The maximum value allowed for this property.

If an application tries to set the value of this property to a value higher than maxValue, the repository SHALL throw a ConstraintViolation exception.

2.9.3.4 Attributes specific to Decimal Object-Type Property Definitions

The following Object ***attributes*** SHALL only apply to Property-Type definitions whose *propertyType* “Decimal”, in addition to the common attributes specified above. A repository MAY provide additional guidance on what values can be accepted. If the following attributes are not present the repository behavior is undefined and it MAY throw an exception if a runtime constraint is encountered.

`precision` Integer

This is the precision in bits supported for values of this property. Valid values for this attribute are:

- 32: 32-bit precision (“single” as specified in IEEE-754-1985).
- 64: 64-bit precision (“double” as specified in IEEE-754-1985.)

`minValue` Decimal

The minimum value allowed for this property.

If an application tries to set the value of this property to a value lower than minValue, the repository SHALL throw a ConstraintViolation exception.

`maxValue`	Decimal

The maximum value allowed for this property.

If an application tries to set the value of this property to a value higher than maxValue, the repository SHALL throw a ConstraintViolation exception.

2.9.3.5 Attributes specific to String Object-Type Property Definitions

The following Object ***attributes*** SHALL only apply to Property-Type definitions whose *propertyType* “String”, in addition to the common attributes specified above. A repository MAY provide additional guidance on what values can be accepted. If the following attributes are not present the repository behavior is undefined and it MAY throw an exception if a runtime constraint is encountered.

`maxLength`	Integer

The maximum length (in characters) allowed for a value of this property.

If an application attempts to set the value of this property to a string larger than the specified maximum length, the repository SHALL throw a ConstraintViolation exception.

2.9.3.6 Attributes specific to XML Object-Type Property Definitions

The following Object ***attributes*** SHALL only apply to Property-Type definitions whose *propertyType* “Xml”, in addition to the common attributes specified above. A repository MAY provide additional guidance on what values can be accepted. If the following attributes are not present the repository behavior is undefined and it MAY throw an exception if a runtime constraint is encountered.

`schemaURI`	URI

The location of an XML schema to which the value of this property SHALL conform.

If an application attempts to set the value of this property to an XML value that does NOT conform to the schema, the repository SHALL throw a ConstraintViolation exception.

2.9.4 CMIS Base Object-Type Definitions

This section specifies the Object-Type Definitions for each of the four CMIS base types (2.3 Document, 2.4 Folder, 2.5 Relationship, 1.1 Policy).

2.9.4.1 Document Object-Type Definition

2.9.4.1.1 Attribute Values

The Document base Object-Type SHALL have the following attribute values.

Notes:

- A value of <repository-specific> indicates that the value of the property MAY be set to any valid value for the attribute type.
- Unless explicitly stated otherwise, all values specified in the list SHALL be followed for the Object-Type definition.

`id`

Value: cmis:Document

`localName`

Value: <repository-specific>

`localNamespace`

Value: <repository-specific>

queryName

Value: cmis:Document

displayName

Value: <repository-specific>

baseTypeId

Value: cmis:Document

parentId

Value: Not set

description

Value: <repository-specific>

creatable

Value: <repository-specific>

fileable

Value:

- If the repository does NOT support the “un-filing” capability:
    - TRUE
- If the repository does support the “un-filing” capability:
    - <repository-specific>, but SHOULD be TRUE

queryable

Value: SHOULD be TRUE

controllablePolicy

Value: <repository-specific>

includedInSupertypeQuery

Value: TRUE

versionable

Value: <repository-specific>

contentStreamAllowed

Value: <repository-specific>

`controllableACL`

Value: <repository-specific>

2.9.4.1.2 Property Definitions

The Document base Object-Type SHALL have the following property definitions, and MAY include additional property definitions. Any attributes not specified for the property definition are repository specific. The repository MUST have the following property definitions on the Document Type:

cmis:Name	Name of the object
Inherited:	False
Property Type:	String
Cardinality:	Single

cmis:ObjectId	Id of the object
Required:	True
Inherited:	False
Property Type:	ID
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

cmis:BaseTypeId	Id of the base object type for the object
Required:	True
Inherited:	False
Property Type:	ID
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

cmis:ObjectTypeId	Id of the object's type
Required:	True
Inherited:	False
Property Type:	ID
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

cmis:CreatedBy	User who created the object.
Required:	True
Inherited:	False

Property Type:	String
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	True
Orderable:	True

cmis:CreationDate	DateTime when the object was created.
Required:	True
Inherited:	False
Property Type:	DateTime
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	True
Orderable:	True

cmis:LastModifiedBy	User who created the object.
Required:	True
Inherited:	False
Property Type:	String
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	True
Orderable:	True

cmis:LastModificationDate	DateTime when the object was last modified.
Required:	True
Inherited:	False
Property Type:	DateTime
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	True
Orderable:	True

cmis:ChangeToken	Opaque token used for optimistic locking & concurrency checking. (see 3.1.5 Change Tokens section)
Inherited:	False
Property Type:	String
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

cmis:IsImmutable	TRUE if the repository SHALL throw an error at any attempt to update or delete the object.
Inherited:	False
Property Type:	Boolean
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

cmis:IsLatestVersion	See 2.10 Versioning section.
Required:	True
Inherited:	False
Property Type:	Boolean
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

cmis:IsMajorVersion	See 2.10 Versioning section.
Inherited:	False
Property Type:	Boolean
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

cmis:IsLatestMajorVersion	See 2.10 Versioning section.
Inherited:	False
Property Type:	Boolean
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

cmis:VersionLabel	See 2.10 Versioning section.
Required:	True
Inherited:	False
Property Type:	String
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

cmis:VersionSeriesId	See 2.10 Versioning section.
Required:	True
Inherited:	False
Property Type:	ID
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

cmis:IsVersionSeriesCheckedOut	See 2.10 Versioning section.
Required:	True
Inherited:	False
Property Type:	Boolean
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

cmis:VersionSeriesCheckedOutBy	See 2.10 Versioning section.
Required:	False
Inherited:	False
Property Type:	String
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

cmis:VersionSeriesCheckedOutId	See 2.10 Versioning section.
Required:	False
Inherited:	False
Property Type:	ID
Cardinality:	Single
Updateability:	Read Only

Choices:	Not Applicable
Open Choice:	Not Applicable

cmis:CheckinComment	See 2.10 Versioning section.
Required:	False
Inherited:	False
Property Type:	ID
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

The following properties MAY be NOT SET if the document does not have a content stream:

cmis:ContentStreamLength	Length of the content stream (in bytes).
Required:	False. This property MUST be Not Set if the object does not have a content-stream. It is required if the object has a content stream.
Inherited:	False
Property Type:	Integer
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

cmis:ContentStreamMimeType	MIME type of the Content Stream
Required:	False. This property MUST be Not Set if the object does not have a content-stream. It is required if the object has a content stream.
Inherited:	False
Property Type:	String
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

cmis:ContentStreamFileName	MIME type of the Content Stream
Required:	False. This property MUST be Not Set if the object does not have a content-stream. It is required if the object has a content stream.
Inherited:	False
Property Type:	String
Cardinality:	Single

Updateability:	Repository Specific
Choices:	Repository Specific
Open Choice:	Repository Specific

cmis:ContentStreamId	Id of the stream
Required:	False. This property MUST be Not Set if the object does not have a content-stream. It is required if the object has a content stream.
Inherited:	False
Property Type:	ID
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

2.9.4.2 Folder Object-Type Definition

2.9.4.2.1 Attribute Values

The Folder base Object-Type SHALL have the following attribute values.

Notes:

- A value of <repository-specific> indicates that the value of the property MAY be set to any valid value for the attribute type.
- Unless explicitly stated otherwise, all values specified in the table SHALL be followed for the Object-Type definition.

`id`

Value: cmisFolder

`localName`

Value: <repository-specific>

`localNamespace`

Value: <repository-specific>

`queryName`

Value: cmis:Folder

`displayName`

Value: <repository-specific>

`baseTypeId`

Value: cmis:Folder

`parentId`

Value: Not set

`description`

Value: <repository-specific>

`creatable`

Value: <repository-specific>

`fileable`

Value: TRUE

`queryable`

Value: SHOULD be TRUE

`controllablePolicy`

Value: <repository-specific>

`includedInSupertypeQuery`

Value: TRUE

`controllableACL`

Value: <repository-specific>

2.9.4.2.2 Property Definitions

The Folder base Object-Type SHALL have the following property definitions, and MAY include additional property definitions. Any attributes not specified for the Property Definition are repository specific. The repository MUST have the following property definitions on the Folder Type:

cmis:Name	Name of the object
Inherited:	False
Property Type:	String
Cardinality:	Single

cmis:ObjectId	Id of the object
Required:	True
Inherited:	False
Property Type:	ID
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable

Open Choice:	Not Applicable
cmis:BaseTypeId	Id of the base object type for the object
Required:	True
Inherited:	False
Property Type:	ID
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable
cmis:ObjectTypeId	Id of the object's type
Required:	True
Inherited:	False
Property Type:	ID
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable
cmis:CreatedBy	User who created the object.
Required:	True
Inherited:	False
Property Type:	String
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	True
Orderable:	True
cmis:CreationDate	DateTime when the object was created.
Required:	True
Inherited:	False
Property Type:	DateTime
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	True
Orderable:	True

cmis:LastModifiedBy	User who created the object.
Required:	True
Inherited:	False
Property Type:	String
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	True
Orderable:	True

cmis:LastModificationDate	DateTime when the object was last modified.
Required:	True
Inherited:	False
Property Type:	DateTime
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	True
Orderable:	True

cmis:ChangeToken	Token used for optimistic locking & concurrency checking. (see 3.1.5 Change Tokens section)
Inherited:	False
Property Type:	String
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

cmis:ParentId	ID of the parent folder of the folder.
Required:	True
Inherited:	False
Property Type:	ID
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

cmis:PathName	Name of the path element for this folder. See 2.4.3 Paths section.

Required:	True
Inherited:	False
Property Type:	String
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

cmis:AllowedChildObjectTypeIds	Id's of the set of Object Types that can be created, moved or filed into this folder.
Required:	False
Inherited:	False
Property Type:	ID
Cardinality:	Multi
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

2.9.4.3 Relationship Object-Type Definition

2.9.4.3.1 Attribute Values

The Relationship base Object-Type SHALL have the following attribute values.

Notes:

- A value of <repository-specific> indicates that the value of the property MAY be set to any valid value for the attribute type.
- Unless explicitly stated otherwise, all values specified in the table SHALL be followed for the Object-Type definition.

`id`

Value: cmis:Relationship

`localName`

Value: <repository-specific>

`localNamespace`

Value: <repository-specific>

`queryName`

Value: cmis:Relationship

`displayName`

Value: <repository-specific>

`baseTypeId`

Value: cmis:Relationship

`parentId`

Value: Not set

`description`

Value: <repository-specific>

`creatable`

Value: <repository-specific>

`fileable`

Value: FALSE

`queryable`

Value: FALSE

`includedInSupertypeQuery`

Value: TRUE

`controllablePolicy`

Value: <repository-specific>

`allowedSourceTypes`

Value: <repository-specific>

`allowedTargetTypes`

Value: <repository-specific>

`controllableACL`

Value: <repository-specific>

2.9.4.3.2 Property Definitions

The Relationship base Object-Type SHALL have the following property definitions, and MAY include additional property definitions. Any attributes not specified by the Property Definitions are repository specific. The repository MUST have the following property definitions on the Folder Type:

cmis:Name	Name of the object
Inherited:	False
Property Type:	String
Cardinality:	Single

cmis:ObjectId	Id of the object
Required:	True
Inherited:	False
Property Type:	ID
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

cmis:BaseTypeId	Id of the base object type for the object
Required:	True
Inherited:	False
Property Type:	ID
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

cmis:ObjectTypeId	Id of the object’s type
Required:	True
Inherited:	False
Property Type:	ID
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

cmis:CreatedBy	User who created the object.
Inherited:	False
Property Type:	String
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	True
Orderable:	True

cmis:CreationDate	DateTime when the object was created.
Inherited:	False
Property Type:	DateTime
Cardinality:	Single
Updateability:	Read Only

Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	True
Orderable:	True

cmis:LastModifiedBy	User who created the object.
Inherited:	False
Property Type:	String
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	True
Orderable:	True

cmis:LastModificationDate	DateTime when the object was last modified.
Inherited:	False
Property Type:	DateTime
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	True
Orderable:	True

cmis:ChangeToken	Opaque token used for optimistic locking & concurrency checking. (see 3.1.5 Change Tokens section)
Inherited:	False
Property Type:	String
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

cmis:SourceId	ID of the source object of the relationship.
Required:	True
Inherited:	False
Property Type:	ID
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

Queryable:	False
Orderable:	False

cmis:TargetId	ID of the target object of the relationship.
Required:	True
Inherited:	False
Property Type:	ID
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	False
Orderable:	False

2.9.4.4 Policy Object-Type Definition

2.9.4.4.1 Attribute Values

The Policy base Object-Type SHALL have the following attribute values.

Notes:

- A value of <repository-specific> indicates that the value of the property MAY be set to any valid value for the attribute type.
- Unless explicitly stated otherwise, all values specified in the table SHALL be followed for the Object-Type definition.

`id`

Value: cmis:Policy

`localName`

Value: <repository-specific>

`localNamespace`

Value: <repository-specific>

`queryName`

Value: cmis: Policy

`displayName`

Value: <repository-specific>

`baseTypeId`

Value: cmis: Policy

`parentId`

Value: Not set

`description`

Value: <repository-specific>

`creatable`

Value: <repository-specific>

`fileable`

Value: <repository-specific>

`queryable`

Value: <repository-specific>

`includedInSupertypeQuery`

Value: TRUE

`controllablePolicy`

Value: <repository-specific>

`controllableACL`

Value: <repository-specific>

2.9.4.4.2 Property Definitions

The Policy base Object-Type SHALL have the following property definitions, and MAY include additional property definitions. Any attributes not specified by the Property Definitions are repository specific. The repository MUST have the following property definitions on the Folder Type:

cmis:Name	Name of the object
Inherited:	False
Property Type:	String
Cardinality:	Single

cmis:ObjectId	Id of the object
Required:	True
Inherited:	False
Property Type:	ID
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

cmis:BaseTypeId	Id of the base object type for the object

Required:	True
Inherited:	False
Property Type:	ID
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

cmis:ObjectTypeId	Id of the object’s type
Required:	True
Inherited:	False
Property Type:	ID
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

cmis:CreatedBy	User who created the object.
Inherited:	False
Property Type:	String
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	True
Orderable:	True

cmis:CreationDate	DateTime when the object was created.
Inherited:	False
Property Type:	DateTime
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	True
Orderable:	True

cmis:LastModifiedBy	User who created the object.
Inherited:	False
Property Type:	String
Cardinality:	Single
Updateability:	Read Only

Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	True
Orderable:	True

cmis:LastModificationDate	DateTime when the object was last modified.
Inherited:	False
Property Type:	DateTime
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	True
Orderable:	True

cmis:ChangeToken	Opaque token used for optimistic locking & concurrency checking. (see 3.1.5 Change Tokens section)
Inherited:	False
Property Type:	String
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable

cmis:PolicyText	User-friendly description of the policy
Required:	True
Inherited:	False
Property Type:	String
Cardinality:	Single
Updateability:	Read Only
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	False
Orderable:	False

2.10 Versioning

CMIS supports versioning of Document objects. Folder objects, relationship objects, and policy objects cannot be versioned.

Whether or not a Document object is versionable (i.e. whether or not operations performed on the object via the Versioning Services SHALL be allowed) is specified by the "*versionable"* attribute on its Object-type.

A **version** of a Document object is an explicit/"deep" copy of the object, preserving its state at a certain point in time. Each version of a Document object is itself a Document object, i.e. has its own *ObjectId*, property values, MAY be acted upon using all CMIS services that act upon Document objects, etc.

2.10.1 Version Series

A **version series** for a Document object is a transitively closed collection of all Document objects that have been created from an original Document in the Repository. Each version series has a unique, system-assigned, and immutable **version series ID.**

The version series has transitive closure -- that is, if object B is a version of object A, and object C is a version of object B, then object C is also a version of object A. The objects in a version series can be conceptually sequenced by their respective *LastModificationDate* properties.

Additionally, the repository MAY expose a textual **VersionLabel** that describes to a user the position of an individual object with respect to the version series. (For example, version 1.0).

Note: A Document object that is NOT versionable will always have a single object in its Version Series. A versionable Document object MAY have one or more objects in its Version Series.

2.10.2 Latest Version

The version that has the most recent *LastModificationDate* is called the ***Latest Version*** of the series, or equivalently, the latest version of any Document object in the series.

When the latest version of a version series is deleted, a previous version (if there is one) becomes the latest version. When the latest major version is deleted, a previous major version (if there is one) becomes the latest major version.

2.10.2.1 Behavioral constraints on non-Latest Versions

Repositories NEED NOT allow the non-latest versions in a Version Series to be updated, queried, or searched.

2.10.3 Major Versions

A Document object in a Version Series MAY be designated as a ***Major Version.***

The CMIS specification does not define any semantic/behavioral differences between Major and non-Major versions in a Version Series. Repositories may enforce/apply additional constraints or semantics for Major versions, if the effect on CMIS services remains consistent with an allowable behavior of the CMIS model.

If the Version Series contains one or more Major versions, the one that has the most recent *LastModificationDate* is the ***Latest Major Version*** of the version series.

(Note that while a Version Series SHALL always have a *Latest Version,* it NEED NOT have a *Latest Major Version.)*

When the latest major version is deleted, a previous major version (if there is one) becomes the latest major version.

2.10.4 Services that modify Version Series

2.10.4.1 Checkout

A new version of a Versionable Document object is created when the *checkout* service is invoked on a Document object. A repository MAY allow *any* Document object in a version series to be checked out, or MAY only allow the *Latest Version* to be checked out.

The effects of invoking the *checkout* service SHALL be as follows:

- A new Document object , referred to herein as the ***Private Working Copy* (PWC),** is created.
  - The PWC NEED NOT be visible to users who have permissions to view other Document objects in the Version Series.
  - Until it is checked in (using the *checkIn* service), the PWC SHALL NOT be considered the *LatestVersion* or *LatestMajorVersion* in the Version Series.
  - The property values for the PWC SHALL be identical to the properties of the Document object on which the *checkout* service was invoked. The content-stream of the PWC MAY be identical to the content-stream of the Document object on which the *checkout* service was invoked, or MAY be "not set".

After a successful *checkout* operation is completed, and until such time when the PWC is deleted (via the *cancelCheckOut* service) or checked-in (via the *checkIn)* service, the effects on other Documents in the Version Series SHALL be as follows:

- The repository SHALL throw an exception if the *checkout* service is invoked on any Document in the Version Series. (I.e. there can only be one PWC for a version series at a time.)
- The value of the isVersionSeriesCheckedOut property SHALL be TRUE.
- The value of the IsVersionSeriesCheckedOutBy property MAY be set to a value indicating which user created the PWC. (The Repository MAY still show the "not set" value for this property.)
- The value of the VersionSeriesCheckedOutId property MAY be set to the ObjectId of the PWC. (The Repository MAY still show the "not set" value for this property).
- The repository MAY prevent operations that modify or delete the other Documents in the Version Series.

2.10.4.2 Updates to the Private Working Copy

If the repository supports the optional "PWCUpdateable" capability, then the repository SHALL allow authorized users to modify the PWC Object using the Object services (e.g. *UpdateProperties).*

If the repository does NOT supp[illegible]he "PWCUpdateable" capability, then the PWC object can only be modified as part of the *checkIn* s[illegible]ce call.

2.10.4.3 Discarding Check out

An authorized user MAY discard the check-out using the *cancelCheckOut* service on any Document in the Version Series or by using the *deleteObject* service on the PWC Object. The effects of discarding a check-out SHALL be as follows:

- The PWC Object SHALL be deleted.
- For all other Documents in the Version Series:
  - The value of the isVersionSeriesCheckedOut property SHALL be FALSE.
  - The value of the IsVersionSeriesCheckedOutBy property SHALL be "not set".
  - The value of the VersionSeriesCheckedOutId property SHALL be "not set".
  - The repository SHALL allow authorized users to invoke the *checkout* service.

2.10.4.4 Checkin

An authorized user/application MAY “check in” the Private Working Copy object via the *checkin* service.

The *checkin* service allows users/applications to provide update property values and a content-stream for the PWC object.

The effects of the checkin service SHALL be as follows:

- The PWC object SHALL be updated as specified by the inputs to the *checkin* service. (Note that for repositories that do NOT support the “PWCUpdateable” property, this is the only way to update the PWC object.)
- The PWC shall be considered the *Latest Version* in the Version Series.
- If the inputs to the *checkIn* service specified that the PWC SHALL be a “major version”, then the PWC SHALL be considered the *Latest Major Version* in the Version Series.
- For all Documents in the Version Series:
    - The value of the isVersionSeriesCheckedOut property SHALL be FALSE.
    - The value of the IsVersionSeriesCheckedOutBy property SHALL be “not set”.
    - The value of the VersionSeriesCheckedOutId property SHALL be “not set”.
    - The repository SHALL allow authorized users to invoke the *checkout* service.

Note: The Repository MAY change the ID of the PWC upon completion of the *checkin* service invocation.

Note: A repository MAY automatically create new versions of Document objects without an explicit invocation of the checkout/checkin services.

2.10.5 Versioning Properties on Document Objects

All Document objects will have the following read-only property values pertaining to versioning:

`cmis:IsLatestVersion` `Boolean`

TRUE if the Document object is the *Latest Version* in its *Version Series.* FALSE otherwise.

`cmis:IsMajorVersion` `Boolean`

TRUE if the Document object is a *Major Version* in its *Version Series.* FALSE otherwise.

`cmis:IsLatestMajorVersion` `Boolean`

TRUE if the Document object is the *Latest Major Version* in its *Version Series.* FALSE otherwise.

`cmis:VersionLabel` `String`

Textual description the position of an individual object with respect to the version series. (For example, version 1.0).

`cmis:VersionSeriesId` `ID`

ID of the Version Series for this Object.

`cmis:IsVersionSeriesCheckedOut` `Boolean`

TRUE if there currenly exists a Private Working Copy for this Version Series. FALSE otherwise

`cmis:VersionSeriesCheckedOutBy` `String`

If IsVersionSeriesCheckedOut is TRUE: then an identifier for the user who created the Private Working Copy. “Not set” otherwise.

`cmis:VersionSeriesCheckedOutId` `ID`

If IsVersionSeriesCheckedOut is TRUE: The Identifier for the Private Working Copy. “Not set” otherwise.

`cmis:CheckinComment` `String`

Textual comment associated with the given version.

Note: Changes made via the Versioning Services that affect the values of these properties SHALL NOT constitute modifications to the Document objects in the Version Series (e.g. SHALL NOT affect the cmis:LastModificationDate, etc.)

2.10.6 Object Creation and Initial Versioning State

A repository MAY create new Document objects in a “Private Working Copy” state when they are created via the *createObject* service. This state is logically equivalent to having a Version Series that contains exactly one object (the PWC) and 0 other documents.

The repository MAY also create new Document objects in a “Major Version” state. This state is logically equivalent to having a Version Series that contains exactly one Major Version and 0 other documents.

The repository MAY also create new Document objects in a “Non-Major Version” state. This state is logically equivalent to having a Version Series that contains exactly one Non-Major Version and 0 other documents.

2.10.7 Version Specific/Independent membership in Folders

Repositories MAY treat membership of a Document object in a folder collection as “version-specific” or “version-independent”.

Repositories SHALL indicate whether they support version-specific membership in a folder via the “VersionSpecificFiling” optional capability flag.

If the repository is treating folder collection membership as “version-independent”, then:

- Moving or Filing a Document Object into a folder SHALL result in ALL Documents in the Version Series being moved/filed into the folder.
- The Repository MAY return only the latest-version OR latest major-version Document object in a version series in the response to Navigation service requests (getChildren, getDescendants), and NEED NOT return other Document Objects filed in the folder that are in the Version Series.

If the repository is treating folder collection membership as “version-specific”, then moving or Filing a Document Object into a folder SHALL NOT result in other Documents in the Version Series being moved/filed.

2.10.8 Version Specific/Independent membership in Relationships

A relationship object MAY have either a version-specific or version-independent binding to its source and/or target objects. This behavior MAY vary between repositories and between individual relationship types defined for a Repository.

If a relationship object has a version-independent binding to its source/target object, then:

- The getRelationships service invoked on a Document Object SHALL return the relationship if Relationship was source/target is set to ANY Document Object in the Version Series.

If a relationship object has a version-specific binding to its source/target object, then:

- The getRelationships service invoked on a Document Object SHALL return the relationship if Relationship was source/target is set to the ID of the Document Object on which the service was invoked.

2.10.9 Versioning visibility in Query Services

Repositories MAY include non-latest-versions of Document Objects in results to the Discovery Services *(query).*

Repositories SHALL indicate whether they support querying for non-latest-versions via the “AllVersionsSearchable” optional capability flag.

If “AllVersionsSearchable” is TRUE then the Repository SHALL include in the query results ANY Document Object in the Version Series that matches the query criteria. (subject to other query constraints such as security.)

If “AllVersionsSearchable” is FALSE then the Repository SHALL include only the latest version or latest major version of a Version Series in the query results if the latest or latest major version matches the query criteria. (subject to other query constraints such as security.)

Additionally, repositories MAY include Private Working Copy objects in results in results to the Discovery Services *(query).*

Repositories SHALL indicate whether they support querying for Private Working Copy objects via the “PWCSearchable” optional capability flag.

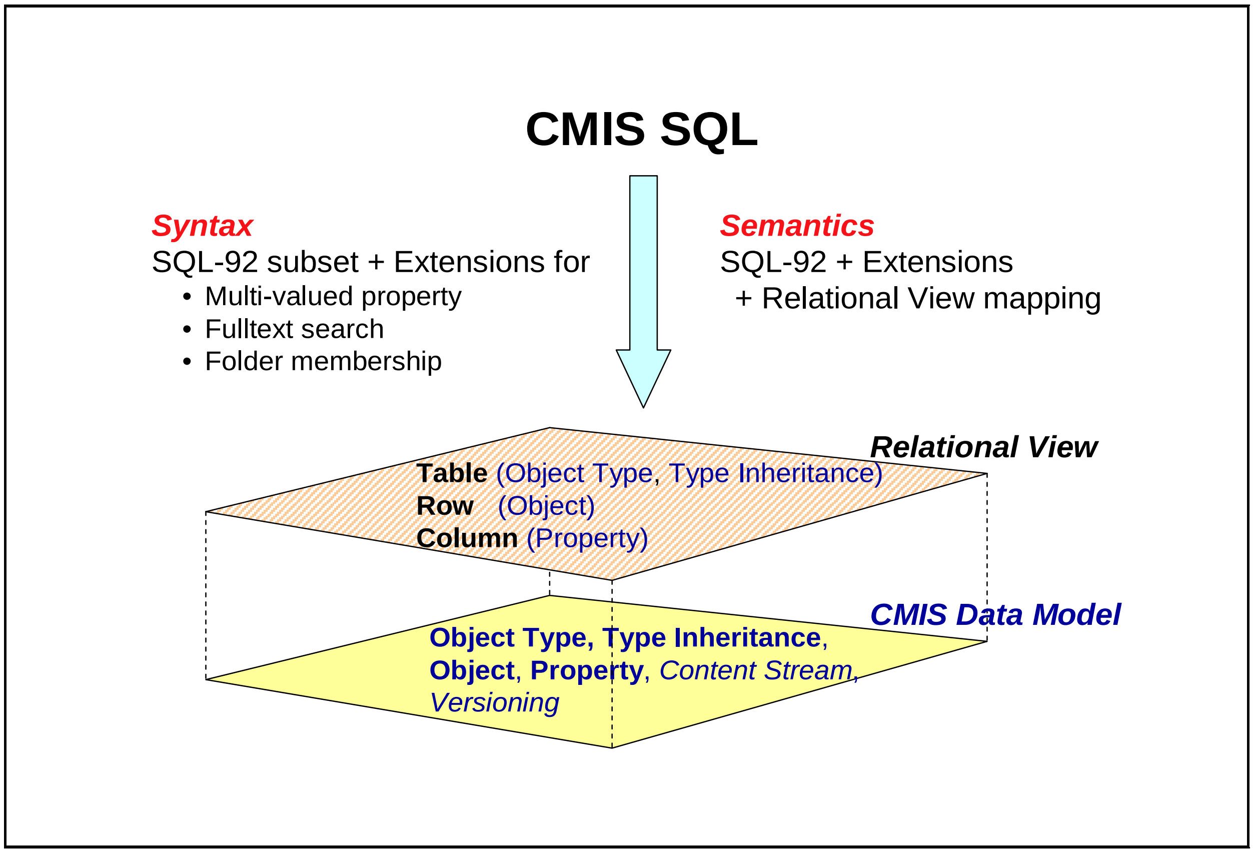
If “PWCSearchable” is TRUE then the Repository SHALL include in the query results ANY Private Working Copy Document Objects that matches the query criteria (subject to other query constraints such as security.)

If “PWCSearchable” is TRUE then the Repository SHALL NOT include in the query results ANY Private Working Copy Document Objects that match the query criteria (subject to other query constraints such as security.)

2.11 Query

CMIS provides a type-based query service for discovering objects that match specified criteria, by defining a read-only projection of the CMIS data model into a *Relational View.*

Through this relational view, queries may be performed via a simplified SQL SELECT statement. This query language, called ***CMIS SQL***, is based on a subset of the SQL-92 grammar (ISO/IEC 9075: 1992 – Database Language SQL), with a few extensions to enhance its filtering capability for the CMIS data model, such as existential quantification for multi-valued property, full-text search, and folder membership. Other statements of the SQL language are not adopted by CMIS. The semantics of CMIS SQL is defined by the SQL-92 standard, plus the extensions, in conjunction with the model mapping defined by CMIS’s relational view.



2.11.1 Relational View Projection of the CMIS Data Model

The relational view of a CMIS repository consists of a collection of virtual tables that are defined on top of the CMIS data model. This relational view is used for query purposes only.

In this relational view a **Virtual Table** is implicitly defined for each *queryable* Object-Type defined in the repository. (Non-queryable Object-Types are NOT exposed through this Relational View.)

In each **Virtual Table**, a **Virtual Column** is implicitly defined for each property defined in the Object-Type Definition AND for all properties defined on ANY ancestor-type of the Object-Type but NOT defined in the Object-Type definition. Virtual Columns for properties defined on ancestor-types of the Object Type but NOT defined in the Object-Type definition SHALL contain the SQL NULL value. Virtual Columns for properties whose value is “not set” SHALL contain the SQL NULL value.

An object type’s *queryName* attribute is used as the table name for the corresponding virtual table, and a property’s *queryName* attribute is used as the column name for the corresponding table column. For this reason, these attributes MUST conform to the syntax rules for SQL identifiers.

The Virtual Column for a multi-valued property SHALL contain a single list value that includes all values of the property.

2.11.1.1 Object-Type Hierarchy in the Relational View Projection

The Relational View projection of the CMIS Data Model ensures that the Virtual Table for a particular Object Type is a complete super-set of the Virtual Table for any and all of its ancestor types.

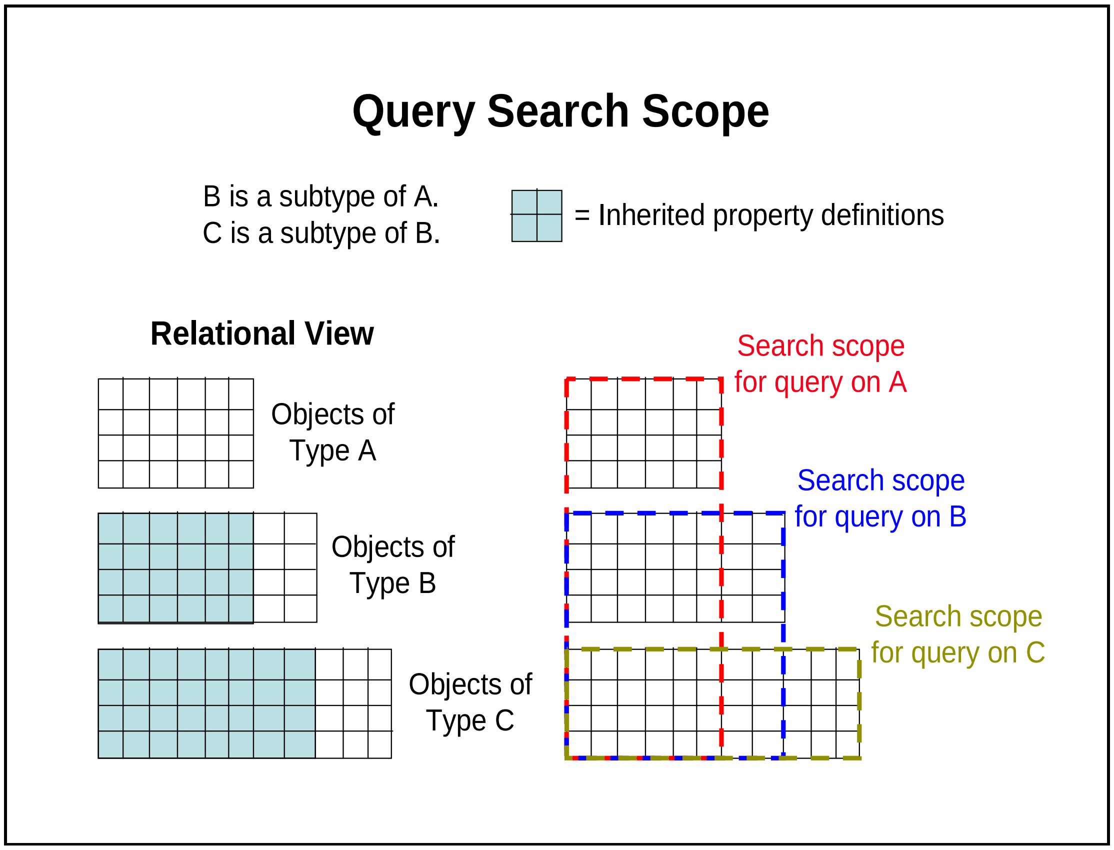
Additionally, an Object-Type definition's "*includedInSupertypeQuery"* specifies whether objects of that Object-Type SHALL be included in the Virtual Table for any of its ancestor types. If the "*includedInSupertypeQuery*" attribute of the Object-Type is FALSE, then objects of that Object-Type SHALL NOT be included in the Virtual Table for any of its ancestor types.

Thus the Virtual Table for an Object Type includes a row not only for each Object of that type, but all Objects that Object-Type AND all Objects of any of that Object Types' Descendant Types for which the *"includedInSupertypeQuery"* attribute is TRUE.

(So, for example:

- If the *Invoice* Object-Type is a child Object-type of *Document* and Invoice's *includeInSupertypeQuery* attribute is TRUE, then queries against the "*Document*" Virtual Table SHALL include all objects of Type Document OR Invoice.
- Extending the previous example, if *PurchaseOrder* is a child Object-Type of *Invoice,* and *PurchaseOrder*'s *includeInSupertypeQuery* attribute is FALSE, then queries against the "*Document*" Virtual Table SHALL include all objects of Type Document OR Invoice, but NOT objects of type *PurchaseOrder.*)

But since the Virtual Table will include only columns for properties defined in the Object-Type underlying the Virtual Table, a row that is a query result representing an Object of a Descendant Type can only include those columns for properties defined on the Object-Type underlying the Virtual Table. (So, building on the previous example, a result row for a query against the "Document" virtual table can include only columns for properties defined on the "Document" Object-Type, even if the Object-Type of the result row defines additional properties.)



2.11.1.2 Content Streams

Content-streams are NOT exposed through this relational view. However, a Repository MAY support full-text indexing of Content Streams (as SHALL be indicated via the Query Repository-optional capability flag).

If a Repository supports full-text indexing, then queries can be constructed that leverage the full-text index of content-streams using the CMIS-SQL *contains()* predicate function (see next section).

2.11.2 CMIS-SQL Definition

This query language, called ***CMIS SQL***, is based on a subset of the SQL-92 grammar. CMIS-specific language extensions to SQL-92 are called out explicitly.

The basic structure of a CMIS-SQL query is a SQL statement that SHALL include the following clauses:

- **SELECT [columns]:** This clause identifies the set of columns that will be included in the query results for each row.
- **FROM [Virtual Table Names]:** This clause identifies which Virtual Table(s) the query will run against.

Additionally, a CMIS-SQL query MAY include the following clauses:

- **WHERE [conditions]:** This clause identifies the constraints that rows SHALL satisfy to be considered a result for the query.
- **ORDER BY [sort specification]:** This clause identifies the order in which the result rows SHALL be sorted in the result row set.

2.11.2.1 SELECT Clause

The SELECT clause SHALL contain exactly one of the following:

- A comma separated list of one or more column names.
    - If an explicit column list is provided: A repository SHALL include in its result row set all of the columns specified in the SELECT clause.
- * : If this token is specified, then the repository SHALL return columns for ALL single-valued properties defined in the Object-Types whose Virtual Tables are listed in the FROM clause, and SHOULD also return all multi-valued properties.

All column names SHALL be valid “queryName” values for properties that are defined as “queryable” in the Object-Type(s) whose Virtual Tables are listed in the FROM clause.

2.11.2.2 FROM Clause

The FROM clause identifies which Virtual Table(s) the query will be run against, as described in the previous section.

The FROM clause SHALL contain only the queryNames of Object-Types whose *queryable* attribute value is TRUE.

2.11.2.2.1 Join Support in CMIS SQL

CMIS repositories MAY support the use of SQL JOIN queries, and SHALL indicate their support level using the Optional Capability attribute “capabilityJoin”.

- If the Repository’s value for the capabilityJoin attribute is **none**, then no JOIN clauses can be used in queries.
- If the Repository’s value for the capabilityJoin attribute is **inneronly**, then only inner JOIN clauses can be used in queries.

- If the Repository's value for the capabilityJoin attribute is **innerandouter**, then inner and/or outer JOIN clauses can be used in queries.

Only explicit joins using the "JOIN" keyword is supported. Queries SHALL NOT include implicit joins as part of the WHERE clause of a CMIS-SQL query.

CMIS queries SHALL only support join operations using the "equality" predicate on single-valued properties.

2.11.2.3 WHERE Clause

This clause identifies the constraints that rows SHALL satisfy to be considered a result for the query.

All column names SHALL be valid "queryName" or their aliased values for properties that are defined as "queryable" in the Object-Type(s) whose Virtual Tables are listed in the FROM clause.

Properties are defined to not support a "null" value, therefore the <null predicate> SHALL be interpreted as testing the not set or set state of the specified property.

2.11.2.3.1 Comparisons permitted in the WHERE clause.

SQL's simple comparison predicate, IN predicate, and LIKE predicate are supported, for single-valued properties only (so that SQL's semantics is preserved). Boolean conjunction (AND), disjunction (OR), and negation (NOT) of predicates are also supported.

Repositories SHOULD support the comparisons for the property types as described in the list below. Repositories MAY support additional comparisons and operators. Any additional operators not specified are repository-specific:

String

Supported Operators: =, <>, [NOT] LIKE

Supported Literal: String

String

Supported Operators: [NOT] IN

Supported Literal: List of Strings

Decimal

Supported Operators: =, <>, <, <=, >, >=

Supported Literal: Decimal

Decimal

Supported Operators: [NOT] IN

Supported Literal: List of Decimal

Integer

Supported Operators: =, <>, <, <=, >, >=

Supported Literal: Integer

Integer

Supported Operators: [NOT] IN

Supported Literal: List of Integer

Boolean

Supported Operators: =

Supported Literal: <boolean literal>

DateTime

Supported Operators: =, <>, <*, <=*, >*, >=*

Supported Literal: <datetime literal>

* - comparison is based on chronological before or after date.

DateTime

Supported Operators: [NOT] IN

Supported Literal: List of <datetime literal>'s

ID

Supported Operators: =, <>

Supported Literal: ID (compared as an opaque String)

ID

Supported Operators: [NOT] IN

Supported Literal: List of IDs (compared as an opaque String)

URI

Supported Operators: =, <>

Supported Literal: URI (compared as an opaque String)

URI

Supported Operators: [NOT] IN

Supported Literal: List of URIs (compared as an opaque String)

URI

Supported Operators: [NOT] LIKE

Supported Literal: URI (compared as an opaque String)

Operations on the SCORE() output MUST be treated the same as decimal operations.

When using properties in a join statement, comparison MUST be allowed on properties of the same types as defined by the table above. Repositories MAY extend this behavior.

The ANY operation argument MUST be one of the properties found in the table above which supports equality operations

2.11.2.3.2 Multi-valued property support (CMIS-SQL Extension)

CMIS-SQL includes several new non-terminals to expose semantics for querying multi-valued properties, in a way that does not alter the semantics of existing SQL-92 production rules.

2.11.2.3.2.1 Multi-valued column references

BNF grammar structure: <Multi-valued-column reference>, <multi-valued-column name>

- These are non-terminals defined for multi-valued properties whereas SQL-92's <column reference> and <column name> are retained for single-valued properties only. This is to preserve the single-value semantics of a regular "column" in the SQL-92 grammar.

2.11.2.3.2.2 <Quantified comparison predicate>

The SQL-92 production rule for <quantified comparison predicate> is extended to accept a multi-valued property in place of a <table subquery>.   This operation is restricted to equality tests only.

<Table subquery> is not supported in CMIS-SQL.

The SQL-92 <quantifier> is restricted to ANY only.

The SQL-92 <row value constructor> is restricted to a literal only.

Example:

```
SELECT          Y.CLAIM_NUM, X.PROPERTY_ADDRESS, Y.DAMAGE_ESTIMATES
FROM  POLICY AS X JOIN CLAIMS AS Y ON ( X.POLICY_NUM = Y.POLICY_NUM )
WHERE           ( 100000 = ANY Y.DAMAGE_ESTIMATES )
```

(Note: DAMAGE_ESTIMATES is a multi-valued Integer property.)

2.11.2.3.2.3 IN/ANY Predicate

BNF grammar structure: <Quantified in predicate>

CMIS-SQL exposes a new IN predicate defined for a multi-valued property. It is modeled after the SQL-92 IN predicate, but since the entire predicate is different semantically, it has its own production rule in the BNF grammar below.

The quantifier is restricted to ANY. The predicate SHALL be evaluated to TRUE if at least one of the property's values is (or, is not, if NOT is specified) among the given list of literal values. Otherwise the predicate is evaluated to FALSE.

The ANY operation argument MUST be one of the properties found in the comparison list above which supports IN operations.

Example:

```
SELECT          *
FROM  CAR_REVIEW
WHERE           (MAKE = “buick” ) OR
                ( ANY FEATURES IN (“NAVIGATION SYSTEM”, “SATELLITE RADIO”, ”MP3”) )
```

(Note: FEATURES is a multi-valued String property.)

2.11.2.3.3 CONTAINS() predicate function (CMIS-SQL Extension)

BNF grammar structure:: CONTAINS ([<qualifier> ,] <text search expression>)

Usage: This is a predicate function that encapsulates the full-text search capability that MAY be provided by a Repository (See previous section.)

Inputs:

<Qualifier>
The value of this optional parameter SHALL be the name of one of the Virtual Tables listed in the FROM clause for the query.

- If specified, then the predicate SHOULD only be applied to objects in the specified Virtual Table, but a repository MAY ignore the value of the parameter.
- If not specified, applies to the single virtual table. If the query is a join, a server SHOULD throw an exception if the qualifier is not specified.

<Text Search Expression>
The <text search expression> parameter SHALL be a character string literal in quotes, specifying the full-text search criteria.

The Text Search Expression may be a set of terms or phrases with an optional '-' to signal negation. A phrase is defined as a word or group of words. A group of words must be surrounded by quotes to be considered a single phrase.

Terms separated by whitespace are AND'ed together.

Terms separated by "OR" are OR'ed together

Implicit "AND" has higher precedence than "OR"

Within a word or phrase, each double quote must also be escaped by a preceding backslash "\"

Return value:

The predicate returns a Boolean value.

The predicate SHALL return TRUE if the object is considered by the repository as "relevant" with respect to the given <text search expression> parameter.

The predicate SHALL return FALSE if the object is considered by the repository as not "relevant" with respect to the given <text search expression> parameter.

Constraints:

At most one CONTAINS() function SHALL be included in a single query statement. The repository MUST throw an exception if more than one CONTAINS() function is found.

The return value of the CONTAINS() function MAY only be included conjunctively (ANDed) with the aggregate of all other predicates, if there is any, in the WHERE clause.

2.11.2.3.4 SCORE() predicate function

BNF grammar structure: SCORE ()

Usage: This is a predicate function that encapsulates the full-text search capability that MAY be provided by a Repository (See previous section.)

Inputs: No inputs SHALL be provided for this predicate function.

Return value:

The SCORE() predicate function returns a decimal value in the interval [0,1] .

A repository SHALL return the value 0 if the object is considered by the repository as having absolutely no relevance with respect to the CONTAINS() function specified in the query.

A repository SHALL return the value 1 if the object is considered by the repository as having absolutely complete relevance with respect to the CONTAINS() function specified in the query.

A repository SHALL return some value in the interval [0,1] based on the degree of relevancy the repository considers the object to have with respect to the CONTAINS() function specified in the query.

Constraints:

The SCORE() function SHALL only be used in queries that also include a CONTAINS() predicate function

The SCORE() function SHALL only be used in the SELECT clause of a query. It SHALL NOT be used in the WHERE clause or in the ORDER BY clauses.

An alias column name defined for the SCORE() function call in the SELECT clause (i.e., "SELECT SCORE() AS column_name …") may be used in the ORDER BY clause.

If SCORE() is included in the SELECT clause and an alias column name is not provided, then a default column name of SEARCH_SCORE is used for the query output.

2.11.2.3.5 IN_FOLDER() predicate function

BNF grammar structure: IN_FOLDER([<qualifier>,] <folder id>)

Usage: This is a predicate function that tests whether or not a candidate object is a child-object of the folder object identified by the given <folder id>..

Inputs:

<qualifier>
The value of this optional parameter SHALL be the name of one of the Virtual Tables listed in the FROM clause for the query.

- If specified, then the predicate SHOULD only be applied to objects in the specified Virtual Table, but a repository MAY ignore the value of the parameter.
- If not specified, applies to the single virtual table. If the query is a join, a server SHOULD throw an exception if the qualifier is not specified.

<folder id>

The value of this parameter SHALL be the ID of a folder object in the repository.

Return value:

The predicate function SHALL return TRUE if the object is a child-object of the folder specified by <folder id>.
The predicate function SHALL return FALSE if the object is a NOT a child-object of the folder specified by <folder id>.

2.11.2.3.6 IN_TREE() predicate function

BNF grammar structure: IN_TREE([<qualifier>,] <folder id>)

Usage: This is a predicate function that tests whether or not a candidate object is a descendant-object of the folder object identified by the given <folder id>.

Inputs:

<qualifier>

The value of this optional parameter SHALL be the name of one of the Virtual Tables listed in the FROM clause for the query.

- If specified, then the predicate SHOULD only be applied to objects in the specified Virtual Table, but a repository MAY ignore the value of the parameter.
- If not specified, applies to the single virtual table. If the query is a join, a server SHOULD throw an exception if the qualifier is not specified.

<folder id>

The value of this parameter SHALL be the ID of a folder object in the repository.

Return value:

The predicate function SHALL return TRUE if the object is a descendant-object of the folder specified by <folder id>.
The predicate function SHALL return FALSE if the object is a NOT a descendant -object of the folder specified by <folder id>.

2.11.2.4 ORDER BY Clause

This clause SHALL contain a comma separated list of one or more column names.

All column names referenced in this clause SHALL be valid “queryName” or their aliased values for properties defined as *orderable* in the Object Type(s) whose Virtual Tables are listed in the FROM clause.

Collation rules for the ORDER BY clause are repository specific.

2.11.2.5 BNF Grammar

This BNF grammar is a “subset” of the SQL-92 grammar (ISO/IEC 9075: 1992 – Database Language SQL), except for the production alternatives that are shown in ***red bold italic*** face. Specifically, except for these extensions, the following production rules are derived from the SQL-92 grammar. Therefore, all the clauses and statements generated by this grammar without the red tokens are valid SQL-92 clauses and statements. The non-terminals used in this grammar are also borrowed from the SQL-92 grammar without altering their semantics. Accordingly, the non-terminal <column name> is used for single-valued properties only so that the semantics of SQL can be preserved and borrowed. This approach not only facilitates comparison of the two query languages, and simplifies the translation of a CMIS query to a SQL query for a RDBMS-based implementation, but also allows future expansion of CMIS SQL to cover a larger subset of SQL with minimum conflict. The CMIS extensions are introduced primarily to support multi-valued properties and full-text search, and to test folder membership. Multi-valued properties are handled separately from single-valued properties, using separate non-terminals and separate production rules to prevent the extensions from corrupting SQL-92 semantics.

```
<CMIS 1.0 query statement> ::= <simple table> [ <order by clause> ]
<simple table> ::= SELECT <select list> <from clause> [ <where clause> ]
<select list> ::= “*”

          | <select sublist> [ { “,” <select sublist> }… ]
<select sublist> ::= <value expression> [ [ AS ] <column name> ]
          | <qualifier> “.*”
          | <multi-valued-column reference>
<value expression> ::= <column reference>  | <numeric value function>
<column reference> ::= [ <qualifier> “.” ] <column name>
<multi-valued-column reference> ::= [ <qualifier> “.” ] <multi-valued-column name>
<numeric value function> ::= SCORE()
<qualifier> ::= <table name> | <correlation name>
<from clause> ::= FROM <table reference>
<table reference> ::= <table name> [ [ AS ] <correlation name> ]
          | <joined table>
<joined table> ::= “(“ <joined table> “)”
          | <table reference> [ <join type> ] JOIN <table reference> [ <join specification> ]
<join type> ::= INNER | LEFT [ OUTER ]
<join specification> ::= ON ( <column reference> "=" <column reference>  )
<where clause> ::= WHERE <search condition>
<search condition> ::= <boolean term> | <search condition> OR <boolean term>
<boolean term> ::= <boolean factor> | <boolean term> AND <boolean factor>
<boolean factor> ::= [ NOT ] <boolean test>
<boolean test> ::= <predicate> | “(“ <search condition> “)”
<predicate> ::= <comparison predicate> | <in predicate> | <like predicate> | <null predicate>
          | <quantified comparison predicate> | <quantified in predicate>
          | <text search predicate> | <folder predicate>
<comparison predicate> ::= <value expression> <comp op> <literal>
<comp op> ::= “=” | “<>” | “<” | “>” | “<=” | “>=”
<literal> ::= <signed numeric literal> | <character string literal> | <datetime literal> | <boolean literal>
<in predicate> ::= <column reference> [ NOT ] IN “(“ <in value list> “)”
<in value list> ::= <literal> { “,” <literal> }…
<like predicate> ::= <column reference> [ NOT ] LIKE <character string literal>
<null predicate> ::= { <column reference> | <multi-valued-column reference> } IS [ NOT ] NULL
<quantified comparison predicate> ::= <literal> “=” ANY <multi-valued-column reference>
<quantified in predicate> ::= ANY <multi-valued-column reference> [ NOT ] IN “(“ <in value list> “)”
<text search predicate> ::= CONTAINS “(“ [ <qualifier>  “,” ] “'”<text search expression> “'” “)”
<folder predicate> ::= { IN_FOLDER | IN_TREE } “(“ [ <qualifier> “,” ] <folder id> “)”
<order by clause> ::= ORDER BY <sort specification> [ { “,” <sort specification> }… ]
<sort specification> ::= <column name> [ ASC | DESC ]
<correlation name> ::= <identifier>
<table name> ::= <identifier>      !! This MUST be the name of an object type.
<column name> ::= <identifier>  !! This MUST be the name of a single-valued property,
                                  or an alias for a scalar output value.
<multi-valued-column name> ::= <identifier>   !! This MUST be the name of a multi-valued property.
<folder id> ::= <character string literal>    !! This MUST be the object identity of a folder object.
<identifier> ::=                 !! As defined by SQL-92 grammar.
<signed numeric literal> ::=     !! As defined by SQL-92 grammar.
<character string literal> ::=   !! As defined by SQL-92 grammar.

!! This is full-text search criteria.
<text search expression> ::= <disjunct> {<space> OR <space> <disjunct>}
<disjunct> ::= <term> {<space> <term>}
```

```
<term> ::= ['-'] <simple term>
<simple term> ::= <word> | <phrase>
<word> ::= <non space char> {<non space char>}
<phrase> ::= <quote> <word> {<space> <word>} <quote>
<space> ::= <space char> {<space char>}
<non space char> ::= <char> - <space char> /* Any Char except SpaceChar */
<space char> ::= ' '
<char> ::= /* Any character */

<datetime literal> ::= TIMESTAMP <quote> <datetime string> <quote>
<datetime string> ::= YYYY-MM-DDThh:mm:ss.sss[Z | +hh:mm | -hh:mm]
<boolean literal> ::= TRUE | FALSE | true | false
<quote> ::= '"' | "'"
```

2.11.3 Escaping

Repositories MUST support the escaping of characters using a backslash (\) in the query statement. The backslash character (\) will be used to escape characters *within quoted strings* in the query as follows:

- \" will represent a double-quote (") character
- \ \ will represent a backslash (\) character
- Within a LIKE string, \% and _ will represent the literal characters % and _, respectively.
- All other instances of a \ are errors.

2.11.4 Exceptions

When a client submits a query that is inappropriate, either because of incorrect comparisons that are not allowed or by specifying query functionality that is not supported by the repository, the repository MUST throw an exception.

If the repository does not support functionality specified by the query statement, the repository MUST throw a notSupported exception.

2.12 Change Log

CMIS provides a "change log" mechanism to allow applications to easily discover the set of changes that have occurred to objects stored in the repository since a previous point in time. This change log can then be used by applications such as search services that maintain an external index of the repository to efficiently determine how to synchronize their index to the current state of the repository (rather than having to query for all objects currently in the repository).

Entries recorded in the change log are referred to below as "change events".

Note that change events in the change log SHALL be returned in ascending order from the time when the change event occurred.

2.12.1 Completeness of the Change Log

The Change Log mechanism exposed by a repository MAY be able to return an entry for every change ever made to content in the repository, or may only be able to return an entry for all changes made since a particular point in time. This "completeness" level of the change log is indicated via the optional `changesIncomplete` capability.

However, repositories SHALL ensure that if an application requests the entire contents of the repository's change log, that the contents of the change log includes ALL changes made to any object in the repository *after* the first change listed in the change log. (I.e. repositories MAY truncate events from the change log on a "first-in first-out" basis, but not in any other order.)

A Repository MAY record events such as filing/unfiling/moving of Documents as change events on the Documents, their parent Folder(s), or both the Documents and the parent Folders.

2.12.2 Change Log Token

The primary index into the change log of a repository is the "change log token". The change log token is an opaque string that uniquely identifies a particular change in the change log.

2.12.2.1 "Latest Change Token" repository information

Repositories that support the changeToken event SHALL expose the latest change token (i.e. the change token corresponding to the most recent change to any object in the repository) as a property returned by the getRepositoryInfo service.

This will enable applications to begin "subscribing" to the change log for a repository by discovering what change log token they should use on a going-forward basis to discover change events to the repository.

2.12.3 Change Event

A change event represents a single action that occurred to an object in the repository that affected the persisted state of the object.

Minimally, a Repository SHALL expose the following information for each change object:

- **ID ObjectId:** The ObjectId of the object to which the change occurred
- **Enum ChangeType:** An enumeration that indicates the type of the change. Valid values are:
  - `created:` The object was created.
  - `updated:` The object was updated.
  - `deleted:` The object was deleted
  - `security:` The access control or security policy for the object were changed.
- **<Properties> properties:** Additionally, for events of changeType "updated", the repository MAY optionally include the new values of properties on the object (if any).

Repositories SHALL indicate whether they include properties for "updated" change events via the optional `enumCapabilityChanges` capability.

3 Services

Part I of the CMIS specification defines a set of services that are described in a protocol/binding-agnostic fashion.

Every protocol binding of the CMIS specification SHALL implement all of the methods described in this section. However, the details of how each service & method is implemented will be described in those protocol binding specifications.

3.1 Common Service Elements

The following elements are common across many of the CMIS services.

3.1.1 Property Filters

All of the methods that allow for the retrieval of properties for CMIS Objects have a “Property Filter” as an optional parameter, which allows the caller to specify a subset of properties for Objects that SHALL be returned by the repository in the output of the method.

Valid values for this parameter are:

- **Not set:** The set of properties to be returned SHALL be determined by the repository.
- **“[queryName1], [queryName2], …”:** The properties explicitly listed SHALL be returned.
  - Note: The [queryName] tokens in the above expression SHALL be valid “queryName” values as defined in the Object-types for the Objects whose properties are being returned.
- *** :** All properties SHALL be returned for all objects.

Repositories SHOULD return only the properties specified in the property filter.

3.1.2 Paging

All of the methods that allow for the retrieval of a collection of CMIS objects support paging of their result sets except where explicitly stated otherwise. The following pattern is used:

Input Parameters:

- **(optional) Integer maxItems:** This is the maximum number of items that the repository SHALL return in its response. Default is repository-specific.
- **(optional) Integer skipCount:** This is the number of potential results that the repository SHALL skip/page over before returning any results. Defaults to 0.

Output Parameters:

- **Boolean** `hasMoreItems`**:** TRUE if the Repository contains additional items in the result set of the method that were not returned based on the values of the input parameters. This implies that the result set has more pages after the response. FALSE otherwise.

If the caller of a method does not specify a value for maxItems, then the Repository MAY select an appropriate number of items to return, and SHALL use the hasMoreItems output parameter to indicate if any additional results were not returned.

3.1.3 Allowable Actions

CMIS provides a service method (*getAllowableActions*) that allows a caller to discover the set of actions that can currently be performed on a particular object.

The set of allowable actions on an object MAY reflect not only the behavior allowed by the CMIS domain model, but also constraints imposed by the repository or any management policy in effect on the item (such as locking, access control, litigation hold, or other lifecycle constraints.) The modeling/discovery of all management policies is outside the scope of CMIS.

Many of the methods that allow for the retrieval of a collection of CMIS objects support also retrieving the set of allowable actions for each object, via the following pattern:

Input Parameters:

- **Boolean** `includeAllowableActions`**:** If TRUE, then the Repository SHALL return the available actions for each object in the result set. Defaults to FALSE.

Output Parameters:

- **<Array> AllowableActions:** A collection listing, for each object, the set of allowable actions. See 2.8.5.2 AllowableActions Mapping section.

3.1.4 Include Relationships

Several CMIS services have the ability to return relationships as part of the service response. A common pattern describing the input is employed as follows:

Enum includeRelationships: Value indicating what relationships in which the objects returned participate SHALL be returned, if any. Values are:

`none:` No relationships SHALL be returned.

`source:` Only relationships in which the objects returned are the source SHALL be returned.

`target:` Only relationships in which the objects returned are the target SHALL be returned.

`both:` Relationships in which the objects returned are the source or the target SHALL be returned.

3.1.5 Change Tokens

The CMIS base object type definitions include an opaque string "ChangeToken" property that a Repository MAY use for optimistic locking and/or concurrency checking to ensure that user updates do not conflict.

If a Repository provides values for the ChangeToken property for an Object, then all invocations of the "update" methods on that object (updateProperties, setContentStream, deleteContentStream) SHALL provide the value of the changeToken property as an input parameter, and the Repository SHALL throw an updateConflictException if the value specified for the changeToken does NOT match the changeToken value for the object being updated.

3.1.6 Exceptions

The following sections list the complete set of exceptions that MAY be returned by a repository in response to a CMIS service method call.

3.1.6.1 General Exceptions

The following exceptions MAY be [illegible]rned by a repository in response to ANY CMIS service method call.

The "Cause" field indicates the circumstances under which a repository SHOULD return a particular exception.

`invalidArgument`

Cause: One or more of the input parameters to the service method is missing or invalid.

`objectNotFound`

Cause: The service call has specified an object that does not exist in the Repository.

`notSupported`

Cause: The service method invoked requires an optional capability not supported by the repository.

`permissionDenied`

Cause: The caller of the service method does not have sufficient permissions to perform the operation.

`runtime`

Cause: Any other cause not expressible by another CMIS exception.

3.1.6.2 Specific Exceptions

The following exceptions MAY be returned by a repositiory in response to one or more CMIS service methods calls.

For each exception, the general intent is listed as well as a list of the methods which MAY cause the exception to be thrown.

`constraint`

Intent: The operation violates a Repository- or Object-level constraint defined in the CMIS domain model.

Methods:

- **Repository Services:**
  - getObjectParents
- **Object Services:**
  - createDocument
  - createFolder
  - createRelationship
  - createPolicy
  - updateProperties
  - moveObject
  - deleteObject
  - setContentStream
  - deleteContentStream
- **Multi-filing Services:**
  - addObjectToFolder
- **Versioning Services:**
  - checkOut
  - cancelCheckOut
  - checkIn
- **Policy Services:**
  - applyPolicy
  - removePolicy
- **Change Log Services:**

    - getContentChanges

`contentAlreadyExists`

Intent: The operation attempts to set the content stream for a Document that already has a content stream without explicitly specifying the “overwrite” parmeter.

Methods:

- **Object Services:**
    - setContentStream

`filterNotValid`

Intent: The property filter input to the operation is not valid.

Methods:

- **Repository Services:**
    - getDescendants
    - getChildren
    - getFolderParent
    - getObjectParents
    - getCheckedOutDocs
- **Object Services:**
    - getProperties
    - getRenditions
- **Versioning Services:**
    - getPropertiesOfLatestVersion
    - getAllVersions
- **Policy Services:**
    - getAppliedPolicies

`folderNotValid`

Intent: The operation is attempting to create an object in an “unfiled” state in a repository that does not support the “Unfiling” optional capability.

Methods:

- **Object Services:**
    - createDocument
- **Multi-filing Services:**
    - removeObjectFromFolder

`storage`

Intent: The repository is not able to store the object that the user is creating/updating due to an internal storage problem.

Methods:

- **Object Services:**
    - createDocument

  - createFolder
  - createRelationship
  - createPolicy
  - updateProperties
  - moveObject
  - setContentStream
  - deleteContentStream
- **Versioning Services:**
  - checkOut
  - checkIn

`streamNotSupported`

Intent: The operation is attempting to get or set a contentStream for a Document whose Object Type specifies that a content stream is not allowed for Document's of that type.

Methods:

- **Object Services:**
  - createDocument
  - getContentStream
  - setContentStream
- **Versioning Services:**
  - checkIn

`updateConflict`

Intent: The operation is attempting to update an object that is no longer current (as determined by the repository).

Methods:

- **Object Services:**
  - updateProperties
  - moveObject
  - deleteObject
  - deleteTree
  - setContentStream
  - deleteContentStream
- **Versioning Services:**
  - checkOut
  - cancelCheckOut
  - checkIn

`versioning`

Intent: The operation is attempting to perform an action on a non-current version of a Document that cannot be performed on a non-current version.

Methods:

- **Object Services:**
  - updateProperties
  - moveObject
  - setContentStream
  - deleteContentStream
- **Versioning Services:**
  - checkOut
  - cancelCheckOut
  - checkIn

3.1.7 Rendition Filter

CMIS provides a service method (getRenditions) that allows a caller to discover the associated Renditions of a particular Document / Folder. Other CMIS (read) services allow for the inclusion of Renditions in their responses.

The Rendition Filter pattern is defined as follows:

```
<renditionInclusion> ::= <none> | <wildcard> | <termlist>
<termlist> ::= <term> | <term> ',' <termlist>
<term> ::= <kind> | <mimetype>
<mimetype> ::= <type> '/' <subtype>
<type> ::= <text>
<subtype> ::= <text> | <wildcard>
<text> ::= /* any char except whitespace */
<wildcard> ::= '*'
<none> ::= 'CMIS.None'
```

An inclusion pattern allows:

- **Wildcard** : include all associated Renditions
- **Comma-separated list of Rendition kinds or mimetypes** : include only those Renditions that match one of the specified kinds or mimetypes
- **`CMIS.None`** : exclude all associated Renditions (typically the default value)

Examples:

- * (include all Renditions)
- `CMIS.Thumbnail` (include only Thumbnails)
- Image/* (include all image Renditions)
- application/pdf, application/x-shockwave-flash (include web ready Renditions)
- `CMIS.None` (exclude all Renditions)

3.2 Repository Services

The Repository Services (getRepositories, getRepositoryInfo, getTypes, getTypeDefinition) are used to discover information about the repository, including information about the repository, the object-types defined for the repository, and other “related” CMIS repositories.

3.2.1 getRepositories

Description: Returns a list of CMIS repositories available from this CMIS service endpoint.

3.2.1.1 Inputs

None.

3.2.1.2 Outputs

A list of repository information, with (at least) the following information for each entry:

- **ID repositoryId:** The identifier for the Repository.
- **String repositoryName:** A display name for the Repository.

The `cmisRepositoryEntryType` schema describes the output of this service.

3.2.1.3 Exceptions Thrown & Conditions

3.1.6.1 General Exceptions

3.2.2 getRepositoryInfo

Description: Returns information about the CMIS repository, the optional capabilities it supports and its Access Control information if applicable. .

3.2.2.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.

3.2.2.2 Outputs

- **ID** `repositoryId`**:** The identifier for the Repository.
    - o Note: This SHALL be the same identifier as the input to the method.
- **String** `repositoryName`**:** A display name for the Repository.
- **String** `repositoryRelationship`**:** A string that MAY describe how this repository relates to other repositories. (See “2.1.2 Related Repositories” section.)
- **String** `repositoryDescription`**:** A display description for the Repository.
- **String** `vendorName`**:** A display name for the vendor of the Repository’s underlying application.
- **String** `productName`**:** A display name for the Repository’s underlying application.
- **String** `productVersion`**:** A display name for the version number of the Repository’s underlying application.
- **ID** `rootFolderId`**:** The ID of the Root Folder Object for the Repository.
- **<List of capabilities>:** The set of values for the repository-optional capabilities specified in section 2.1.1
- **String** `latestChangeLogToken:` The change log token corresponding to the most recent change event for any object in the repository.
- **String** `cmisVersionSupported:`A string that indicates what version of the CMIS specification this repository supports as specified in 2.1.3 Implementation Information.

- **URI** `thinClientURI`: A optional repository-specific URI pointing to the repository's web interface.
- **Boolean** changesIncomplete: Indicates whether or not the repository's change log can return all changes ever made to any object in the repository or only changes made after a particular point in time. Applicable when the repository's optional capability `capabilityChanges` is not `none`.
    - If FALSE, then the change log can return all changes ever made to every object.
    - If TRUE, then the change log includes all changes made since a particular point in time, but not all changes ever made.
- **Enum supportedPermissions** specifies which types of permissions are supported.
    - basic: indicates that the CMIS Basic permissions are supported.
    - repository: Indicates that repository specific permissions are supported.
    - both: indicates that both CMIS basic permissions and repository specific permissions are supported.
- **<Array> Enum ACLPropagation:** The list of allowed values for applyACL, which control how non-direct ACEs are handled by the repository:
    - `objectonly:` indicates that the repository is able to apply ACEs without changing the ACLs of other objects – i.e. ACEs are applied, potentially "breaking" the "sharing" dependency for non-direct ACEs.
    - `propagate:` indicates that the repository is able to apply ACEs to a given object and propagate this change to all "inheriting" objects – i.e. ACEs are applied with the (intended) side effect to "inheriting" objects (objects that "share" the ACL with the given object).
    - `repositorydetermined:` indicates that the repository uses its own mechanisms to handle non-direct ACEs when applying ACLs.
- **<Array> Permission repositoryPermissions:** The list of repository-specific permissions the repository supports for managing ACEs (see Access Control Lists).
- **<Array> PermissionMapping mappings:** The list of mappings for the CMIS Basic permissions to allowable actions (see Access Control Lists).
- **XML** `repositorySpecificInformation`**:** MAY be used by the Repository to return additional XML.

The `cmisRepositoryInfoType` schema describes the markup that will be included in all CMIS protocol bindings to implement this service.

3.2.2.3 Exceptions Thrown & Conditions

3.1.6.1 General Exceptions

3.2.3 getTypeChildren

Description: Returns the list of Object-Types defined for the Repository under the specified Type.

3.2.3.1 Inputs

Required:

- **string repositoryId:** The identifier for the Repository.

Optional:

- **string typeId:** The typeId of an Object-Type specified in the Repository.
    - If specified, then the Repository SHALL return only the specified Object-Type AND all of its child types.

    - If not specified, then the Repository SHALL return all Base Object-Types.
- **Boolean includePropertyDefinitions:** If TRUE, then the Repository SHALL return the property definitions for each Object-Type returned.
    - If False (default), the Repository SHALL return only the attributes for each Object-Type.
- **Integer maxItems:** See 3.1.2 Paging section.
- **Integer skipCount:** See 3.1.2 Paging section.

3.2.3.2 Outputs

- **<Array> Object-Types:** The list of Object-Types defined for the Repository.
- **Boolean hasMoreItems:** See 3.1.2 Paging section.

3.2.3.3 Exceptions Thrown & Conditions

3.1.6.1 General Exceptions

3.2.4 getTypeDescendants

Description: Returns the set of descendant Object-Types defined for the Repository under the specified Type.

Notes:

- This method does NOT support paging as defined in the 3.1.2 Paging section.
- The order in which results are returned SHALL be determined by the repository.

3.2.4.1 Inputs

Required:

- **string repositoryId:** The identifier for the Repository.

Optional:

- **string typeId:** The typeId of an Object-Type specified in the Repository.
    - If specified, then the Repository SHALL return only the specified Object-Type AND all of its descendant types.
    - If not specified, then the Repository SHALL return all types.
- **Integer depth:** The number of levels of depth in the type hierarchy from which to return results. Valid values are:
    - **1 (default):** Return only types that are children of the type.
    - **<Integer value greater than 1>:** Return only types that are children of the type and descendants up to <value> levels deep.
    - **-1:** Return ALL descendant types at all depth levels in the CMIS hierarchy.
- **Boolean includePropertyDefinitions:** If TRUE, then the Repository SHALL return the property definitions for each Object-Type returned.
    - If False (default), the Repository SHALL return only the attributes for each Object-Type.

3.2.4.2 Outputs

- **<Array> Object-Types:** The list of Object-Types defined for the Repository.

3.2.4.3 Exceptions Thrown & Conditions

3.1.6.1 General Exceptions

3.2.5 getTypeDefinition

Description: Gets the definition of the specified Object-Type.Inputs

Required:

- **string repositoryId:** The identifier for the Repository.
- **string typeId:** The typeId of an Object-Type specified in the Repository.

3.2.5.1 Outputs

- Object Type including all property definitions.

3.2.5.2 Exceptions Thrown & Conditions

3.1.6.1 General Exceptions

3.3 Navigation Services

The Navigation Services (getDescendants, getChildren, getFolderParent, getObjectParents, getCheckedoutDocs), are used to traverse the folder hierarchy in a CMIS **Repository**, and to locate Documents that are checked out.

3.3.1 getFolderTree

Description: Gets the set of descendant folder objects contained in the specified folder.

Notes:

- This method does NOT support paging as defined in the 3.1.2 Paging section.
- The order in which results are returned SHALL be determined by the repository.

3.3.1.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID folderId:** The identifier for the folder.

Optional:

- **Integer depth:** The number of levels of depth in the folder hierarchy from which to return results. Valid values are:
  - **1 (default):** Return only folders that are children of the folder and their child folders.
  - **<Integer value greater than 1>:** Return only folders that are children of the folder and descendant folders up to <value> levels deep.
  - **-1:** Return ALL descendant folders at all depth levels in the CMIS hierarchy.
- **Boolean includeAllowableActions:** See “3.1.3 Allowable Actions” section.
- **Enum includeRelationships:** See “3.1.4 Include Relationships” section.
- **String filter:** See “3.1.1 Property Filters” section.

3.3.1.2 Outputs

- **<Array> Objects:** A list of the descendant folders for the specified folder.

3.3.1.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions_General_Exceptions
- `filterNotValid`**:** The Repository SHALL throw this exception if this property filter input parameter is not valid.

3.3.2 getDescendants

Description: Gets the set of descendant objects contained in the specified folder or any of its child-folders.

Notes:

- This method does NOT support paging as defined in the 3.1.2 Paging section.
- The order in which results are returned SHALL be determined by the repository.
- If the Repository supports the optional capability `capabilityVersionSpecificFiling`, then the repository SHALL return the document versions filed in the specified folder or its descendant folders. Otherwise, thelatest version of the documents SHALL be returned.
- If the Repository supports the optional capability `capabilityMutlifiling` and the same document is encountered multiple times in the hierarchy, then the repository SHALL return that document each time is encountered.

3.3.2.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID folderId:** The identifier for the folder.

Optional:

- **Boolean includeAllowableActions:** See “3.1.3 Allowable Actions” section.
- **Enum includeRelationships:** See “3.1.4 Include Relationships” section.
- **Integer depth:** The number of levels of depth in the folder hierarchy from which to return results. Valid values are:
  - **2 (default):** Return only objects that are children of the folder and their children.
  - **<Integer value greater than 1>:** Return only objects that are children of the folder and descendants up to <value> levels deep.
  - **-1:** Return ALL descendant objects at all depth levels in the CMIS hierarchy.
- **String filter:** See “3.1.1 Property Filters” section.

3.3.2.2 Outputs

- **<Array> Objects:** A list of the descendant objects for the specified folder.

3.3.2.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `filterNotValid`**:** The Repository SHALL throw this exception if this property filter input parameter is not valid.

3.3.3 getChildren

Description: Gets the list of child objects contained in the specified folder.

Notes:

- If the Repository supports the optional “VersionSpecificFiling” capability, then the repository SHALL return the document versions filed in the specified folder.
    - Otherwise, the latest version of the documents SHALL be returned.

3.3.3.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID folderId:** The identifier for the folder.

Optional:

- **Boolean includeAllowableActions:** See “3.1.3 Allowable Actions” section.
- **Enum includeRelationships:** See “3.1.4 Include Relationships” section.
- **String filter:** See “3.1.1 Property Filters” section.
- **Integer maxItems:** See 3.1.2 Paging section..
- **Integer skipCount:** See 3.1.2 Paging section..

3.3.3.2 Outputs

- **<Array> Objects:** A list of the child objects for the specified folder.
- **Boolean hasMoreItems**: See 3.1.2 Paging section..

3.3.3.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `filterNotValid`**:** The Repository SHALL throw this exception if this property filter input parameter is not valid.

3.3.4 getFolderParent

Description: Gets the parent folder object for the specified folder object.

Notes:

- Repositories SHOULD always include the “ObjectId” and “ParentId” properties for all objects returned.
    - If this service method is invoked on the root folder of the Repository, then the Repository shall return an empty result set.

3.3.4.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID folderId:** The identifier for the folder.

Optional:

- **String filter:** See “3.1.1 Property Filters” section.

3.3.4.2 Outputs

- **Object:** The parent folder object of the specified folder.

3.3.4.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `filterNotValid`**:** The Repository SHALL throw this exception if this property filter input parameter is not valid.

3.3.5 getObjectParents

Description: Gets the parent folder(s) for the specified non-folder, fileable object.

3.3.5.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID objectId:** The identifier for the folder.

Optional:

- **String filter:** See “3.1.1 Property Filters” section.

3.3.5.2 Outputs

- **<Array> Objects:** A list of the parent folder(s) of the specified objects. Empty for unfiled objects.

3.3.5.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `constraint`**:** The Repository SHALL throw this exception if this method is invoked on an object who Object-Type Definition specifies that it is not fileable.
- `filterNotValid`**:** The Repository SHALL throw this exception if this property filter input parameter is not valid.

3.3.6 getCheckedOutDocs

Description: Gets the list of documents that are checked out that the user has access to.

3.3.6.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.

Optional:

- **Boolean includeAllowableActions:** See “3.1.3 Allowable Actions” section.
- **Enum includeRelationships:** See “3.1.4 Include Relationships” section.
- **ID folderId:** The identifier for a folder in the repository from which documents should be returned.
    - If specified, the Repository SHALL only return checked out documents that are child-objects of the specified folder.
    - If not specified, the Repository SHALL return checked out documents from anywhere in the repository hierarchy.
- **String filter:** See “3.1.1 Property Filters” section.
- **Integer maxItems:** See 3.1.2 Paging section..
- **Integer skipCount:** See 3.1.2 Paging section..

3.3.6.2 Outputs

- **<Array> Objects:** A list of checked out documents.
- **Boolean hasMoreItems**: See 3.1.2 Paging section..

3.3.6.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `filterNotValid`**:** The Repository SHALL throw this exception if this property filter input parameter is not valid.

3.4 Object Services

CMIS provides ID-based CRUD (**C**reate, **R**etrieve, **U**pdate, **D**elete), operations on objects in a Repository.

3.4.1 createDocument

Description: Creates a document object of the specified type in the (optionally) specified location.

3.4.1.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID typeId:** The identifier for the Object-Type of the Document object being created.
- **<Array> properties:** The property values that SHALL be applied to the newly-created Document Object.

Optional:

- **ID folderId:** If specified, the identifier for the folder that SHALL be the parent folder for the newly-created Document Object.
    - o This parameter MUST be specified if the Repository does NOT support the optional “unfiling” capability.
- **<contentStream> contentStream:** The Content Stream that SHALL be stored for the newly-created Document Object. The method of passing the contentStream to the server and the encoding mechanism will be specified by each specific binding. SHALL be required if the type requires it.
- **Enum versioningState:** An enumeration specifying what the versioing state of the newly-created object SHALL be. Valid values are:
    - o `checkedout`**:** The document SHALL be created in the checked-out state.
    - o `major`**:** The document SHALL be created as a major version
    - o `minor` **(default):** The document SHALL be created as a minor version.
- **<Array> policies:** A list of policy IDs that SHALL be applied to the newly-created Document object.
- **<Array> ACE addACEs:** A list of ACEs that SHALL be added to the newly-created Document object, either using the ACL from folderId if specified, or being applied if no folderId is specified.
- **<Array> ACE removeACEs:** A list of ACEs that SHALL be removed from the newly-created Document object, either using the ACL from folderId if specified, or being ignored if no folderId is specified.

3.4.1.2 Outputs

- **ID objectId:** The ID of the newly-created document.

3.4.1.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `constraint`**:** The Repository SHALL throw this exception if ANY of the following conditions are met:
  - The typeId is not an Object-Type whose baseType is “Document”.
  - The typeId is NOT in the list of AllowedChildObjectTypeIds of the parent-folder specified by folderId.
  - The value of any of the properties violates the min/max/required/length constraints specified in the property definition in the Object-Type.
  - The Object-Type definition specified by the typeId parameter’s “contentStreamAllowed” attribute is set to “required” and no contentStream input parameter is provided.
  - The Object-Type definition specified by the typeId parameter’s “versionable” attribute is set to FALSE and a value for the versioningState input parameter is provided.
  - The Object-Type definition specified by the typeId parameter’s “controllablePolicy” is set to FALSE and at least one policy is provided.
  - The Object-Type definition specified by the typeId parameter’s “controllableACL” is set to FALSE and at least one ACE is provided.
  - At least one of the permissions is used in an ACE provided which is not supported by the repository.
- `storage`: See “3.1.6.2 Specific Exceptions” section.
- `streamNotSupported`**:** The Repository SHALL throw this exception if the Object-Type definition specified by the typeId parameter’s “contentStreamAllowed” attribute is set to “not allowed” and a contentStream input parameter is provided.

3.4.2 createFolder

Description: Creates a folder object of the specified type in the specified location.

3.4.2.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID typeId:** The identifier for the Object-Type of the Folder object being created.
- **<Array> properties:** The property values that SHALL be applied to the newly-created Folder Object.
- **ID folderId:** The identifier for the folder that SHALL be the parent folder for the newly-created Folder Object.

Optional:

- **<Array> policies:** A list of policy IDs that SHALL be applied to the newly-created Document object.
- **<Array> ACE addACEs:** A list of ACEs that SHALL be added to the newly-created Document object, either using the ACL from folderId if specified, or being applied if no folderId is specified.
- **<Array> ACE removeACEs:** A list of ACEs that SHALL be removed from the newly-created Document object, either using the ACL from folderId if specified, or being ignored if no folderId is specified.

3.4.2.2 Outputs

- **ID objectId:** The ID of the newly-created folder.

3.4.2.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- **ConstraintViolationException:** The Repository SHALL throw this exception if ANY of the following conditions are met:
    - o The typeId is not an Object-Type whose baseType is “Folder”.
    - o The value of any of the properties violates the min/max/required/length constraints specified in the property definition in the Object-Type.
    - o The typeId is NOT in the list of AllowedChildObjectTypeIds of the parent-folder specified by folderId.
    - o The Object-Type definition specified by the typeId parameter’s “controllablePolicy” is set to FALSE and at least one policy is provided.
    - o The Object-Type definition specified by the typeId parameter’s “controllableACL” is set to FALSE and at least one ACE is provided.
    - o At least one of the permissions is used in an ACE provided which is not supported by the repository.
- `storage`: See “3.1.6.2 Specific Exceptions” section.

3.4.3 createRelationship

Description: Creates a relationship object of the specified type

3.4.3.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID typeId:** The identifier for the Object-Type of the Relationship object being created.
- **<Array> properties:** The property values that SHALL be applied to the newly-created Relationship Object.
- **ID sourceObjectId:** The ID of the source object for the newly-created Relationship.
- **ID targetObjectId:** The ID of the taret object for the newly-created Relationship.

Optional:

- **<Array> policies:** A list of policy IDs that SHALL be applied to the newly-created Document object.
- **<Array> ACE addACEs:** A list of ACEs that SHALL be added to the newly-created Document object, either using the ACL from folderId if specified, or being applied if no folderId is specified.
- **<Array> ACE removeACEs:** A list of ACEs that SHALL be removed from the newly-created Document object, either using the ACL from folderId if specified, or being ignored if no folderId is specified.

3.4.3.2 Outputs

- **ID objectId:** The ID of the newly-created relationship.

3.4.3.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `constraint`**:** The Repository SHALL throw this exception if ANY of the following conditions are met:
    - o The typeId is not an Object-Type whose baseType is “Relationship”.

  - The value of any of the properties violates the min/max/required/length constraints specified in the property definition in the Object-Type.
  - The sourceObjectId's ObjectType is not in the list of "allowedSourceTypes" specified by the Object-Type definition specified by typeId.
  - The targetObjectId's ObjectType is not in the list of "allowedTargetTypes" specified by the Object-Type definition specified by typeId.
  - The Object-Type definition specified by the typeId parameter's "controllablePolicy" is set to FALSE and at least one policy is provided.
  - The Object-Type definition specified by the typeId parameter's "controllableACL" is set to FALSE and at least one ACE is provided.
  - At least one of the permissions is used in an ACE provided which is not supported by the repository.
- `storage`: See "3.1.6.2 Specific Exceptions" section.

3.4.4 createPolicy

Description: Creates a policy object of the specified type

3.4.4.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID typeId:** The identifier for the Object-Type of the Policy object being created.
- **<Array> properties:** The property values that SHALL be applied to the newly-created Policy Object.

Optional:

- **ID folderId:** If specified, the identifier for the folder that SHALL be the parent folder for the newly-created Policy Object.
  - This parameter SHALL be specified if the Repository does NOT support the optional "unfiling" capability.
- **<Array> policies:** A list of policy IDs that SHALL be applied to the newly-created Document object.
- **<Array> ACE addACEs:** A list of ACEs that SHALL be added to the newly-created Document object, either using the ACL from folderId if specified, or being applied if no folderId is specified.
- **<Array> ACE removeACEs:** A list of ACEs that SHALL be removed from the newly-created Document object, either using the ACL from folderId if specified, or being ignored if no folderId is specified.

3.4.4.2 Outputs

- **ID objectId:** The ID of the newly-created Policy Object.

3.4.4.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `constraint`**:** The Repository SHALL throw this exception if ANY of the following conditions are met:
  - The typeId is not an Object-Type whose baseType is "Policy".
  - The value of any of the properties violates the min/max/required/length constraints specified in the property definition in the Object-Type.

    - The typeId is NOT in the list of AllowedChildObjectTypeIds of the parent-folder specified by folderId.
    - The Object-Type definition specified by the typeId parameter's "controllablePolicy" is set to FALSE and at least one policy is provided.
    - The Object-Type definition specified by the typeId parameter's "controllableACL" is set to FALSE and at least one ACE is provided.
    - At least one of the permissions is used in an ACE provided which is not supported by the repository.
- `storage`: See "3.1.6.2 Specific Exceptions" section.

3.4.5 getAllowableActions

Description: Gets the list of allowable actions for an Object (see "3.1.3 Allowable Actions" section).

3.4.5.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID objectId:** The identifier for the object

3.4.5.2 Outputs

- **<Array> AllowableActions:** see "3.1.3 Allowable Actions" section

The `cmisAllowableActionsType` schema describes the output of this service.

3.4.5.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions

3.4.6 getProperties

Description: Gets the list of properties for an Object.

3.4.6.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID objectId:** The identifier for the object

Optional:

- **Boolean includeAllowableActions:** See "3.1.3 Allowable Actions" section.
- **Enum includeRelationships:** See "3.1.4 Include Relationships" section.
- **String filter:** See "3.1.1 Property Filters" section.
- **Boolean includeACLs:** Include the ACL's associated with the object in the response.

3.4.6.2 Outputs

- **<Array> Properties:** The list of properties for the object.
- **<Array>** ACEs: The list of ACEs as returned by getACL for the object, if includeACL was provided

3.4.6.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions

- `filterNotValid`**:** The Repository SHALL throw this exception if this property filter input parameter is not valid.

3.4.7 getFolderByPath

Description: Gets the specified Folder object.

3.4.7.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **String folderPath:** The path to the folder. See "2.4.3 Paths" section.

Optional:

- **Boolean includeAllowableActions:** See "3.1.3 Allowable Actions" section.
- **Enum includeRelationships:** See "3.1.4 Include Relationships" section.
- **String filter:** See "3.1.1 Property Filters" section.

3.4.7.2 Outputs

- **<Array> Properties:** The list of properties for the Folder.

3.4.7.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `filterNotValid`**:** The Repository SHALL throw this exception if this property filter input parameter is not valid.

3.4.8 getContentStream

Description: Gets the content stream for the specified Document object

Notes: Each CMIS protocol binding MAY provide a way for fetching a sub-range within a content stream, in a manner appropriate to that protocol.

3.4.8.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID objectId:** The identifier for the object

Optional:

- **ID streamId:** The identifier for the content stream. For Documents, if not provided, defaults to the primary content stream of the Document. For Folder, it must be provided.

3.4.8.2 Outputs

- **<ContentStream> ContentStream:** The content stream for the object.

3.4.8.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `constraint:` The Repository SHALL throw this exception if the Document specified by objectId does NOT have a content stream.

3.4.9 getRenditions

Description: Gets the list of associated Renditions for the specified object

Notes: Each CMIS protocol binding MAY provide a way for fetching a sub-range within a content stream, in a manner appropriate to that protocol.

3.4.9.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID objectId:** The identifier for the object

Optional:

- **String renditionFilter**: See 3.1.7 Rendition Filter section. If not provided, defaults to '*'
- **Integer maxItems**: See 3.1.2 Paging section
- **Integer skipCount**: See 3.1.2 Paging section

3.4.9.2 Outputs

- **<Array> Renditions:** The set of renditions available on this object

3.4.9.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `notSupported:` The service method requires functionality that is not supported by the repository
- `filterNotValid` : The filter specified is not valid

3.4.10 updateProperties

Description: Updates properties of the specified object.

Notes:

- A Repository MAY automatically create new Document versions as part of an update properties operation. Therefore, the ObjectId output NEED NOT be identical to the ObjectId input.
- If this method is invoked on an object and any of the properties defined for the Object-Type are NOT included in the input to the method, repositories SHALL perform the update operation as if those properties had been included and set to their original values.

3.4.10.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID objectId:** The identifier of the object to be updated.
- **<Array> properties:** The updated property values that SHALL be applied to the Object.

Optional:

- **String changeToken:** See "Change Tokens" section.

3.4.10.2 Outputs

- **ID objectId:** The ID of the updated object.

3.4.10.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `constraint`**:** The Repository SHALL throw this exception if ANY of the following conditions are met:
  - The value of any of the properties violates the min/max/required/length constraints specified in the property definition in the Object-Type.
- `storage`: See “3.1.6.2 Specific Exceptions” section.
- `updateConflict`: See “3.1.6.2 Specific Exceptions” section.
- `versioning`: The Repository SHALL throw this exception if ANY of the following conditions are met:
  - The object is not checked out and ANY of the properties being updated are defined in their Object-Type definition have an attribute value of *updateability* whencheckedout.
  - Additionally, the repository MAY throw this exception if the object is a non-current Document Version.

3.4.11 moveObject

Description: Moves the specified file-able object from one folder to another.

3.4.11.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID objectId:** The identifier of the object to be moved.
- **ID targetFolderId:** The folder into which the object is to be moved.
- **ID sourceFolderId:** The folder from which the object is to be moved.
  - This parameter SHALL be specified if the Repository supports the optional “capabilityUnfiling” capability.

3.4.11.2 Outputs

- **ID objectId:** The identifier of the object to be moved.

3.4.11.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `constraint`**:** The Repository SHALL throw this exception if ANY of the following conditions are met:
  - The typeId is NOT in the list of AllowedChildObjectTypeIds of the parent-folder specified by targetFolderId.
- `storage`: See “3.1.6.2 Specific Exceptions” section.
- `updateConflict`: See “3.1.6.2 Specific Exceptions” section.
- `versioning`: The repository MAY throw this exception if the object is a non-current Document Version.

3.4.12 deleteObject

Description: Deletes the specified object.

Notes: Invoking this service method on an object SHALL not delete the entire version series for a Document Object. To delete an entire version series, use the deleteAllVersions() service.

3.4.12.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID objectId:** The identifier of the object to be deleted.

3.4.12.2 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `constraint`**:** The Repository SHALL throw this exception if ANY of the following conditions are met:
    - o The method is invoked on a Folder object that contains one or more objects.
- `updateConflict`: See "3.1.6.2 Specific Exceptions" section.

3.4.13 deleteTree

Description: Deletes the specified folder object and all of its child- and descendant-objects.

Notes:

- A Repository MAY attempt to delete child- and descendant-objects of the specified folder in any order.
- Any child- or descendant-object that the Repository cannot delete SHALL persist in a valid state in the CMIS domain model.
- This is not atomic.
- However, if deletesinglefiled is chosen and some objects fail to delete, then single-filed objects are either deleted or kept, never just unfiled. This is so that a user can call this command again to recover from the error by using the same tree.

3.4.13.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID folderId:** The identifier of the folder to be deleted.

Optional:

- **Enum** `unfileObjects`**:** An enumeration specifying how the repository SHALL process file-able child- or descendant-objects. Valid values are:
    - o `unfile`**:** Unfile all fileable objects.
    - o `deletesinglefiled`**:** Delete all fileable non-folder objects whose only parent-folders are in the current folder tree. Unfile all other fileable non-folder objects from the current folder tree.
    - o `delete` **(default):** Delete all fileable objects.
- **boolean continueOnFailure:** If TRUE, then the repository SHOULD continue attempting to perform this operation even if deletion of a child- or descendant-object in the specified folder cannot be deleted.
    - o If FALSE **(default)**, then the repository SHOULD abort this method when it fails to delete a single child- or descendant-object.

3.4.13.2 Outputs

- **<Array> ID** failedToDelete: A list of identifiers of objects in the folder tree that were not deleted.

3.4.13.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `updateConflict`: See “3.1.6.2 Specific Exceptions” section.

3.4.14 setContentStream

Description: Sets the content stream for the specified Document object.

Notes: A Repository MAY automatically create new Document versions as part of this service method. Therefore, the DocumentId output NEED NOT be identical to the DocumentId input.

3.4.14.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID documentId:** The identifier for the Document object.
- **<contentStream> contentStream:** The Content Stream that SHALL be stored for the newly-created Document Object.

Optional:

- **Boolean overwriteFlag:** If TRUE **(default)**, then the Repository SHALL replace the existing content stream for the object (if any) with the input contentStream.
  - o If FALSE, then the Repository SHALL only set the input contentStream for the object if the object currently does not have a content-stream.
- **String changeToken:** See “3.1.5 Change Tokens” section.

3.4.14.2 Outputs

- **ID documentId:** The ID of the document.

3.4.14.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `constraint`**:** The Repository SHALL throw this exception if ANY of the following conditions are met:
  - o The Object’s Object-Type definition “contentStreamAllowed” attribute is set to “*notAllowed*”.
- `contentAlreadyExists`**:** The Repository SHALL throw this exception if the input parameter overwriteFlag is FALSE and the Object already has a content-stream.
- `storage`: See “3.1.6.2 Specific Exceptions” section.
- `streamNotSupported`**:** The Repository SHALL throw this exception if the Object-Type definition specified by the typeId parameter’s “contentStreamAllowed” attribute is set to “not allowed”.
- `updateConflict`: See “3.1.6.2 Specific Exceptions” section.
- `versioning`: The repository MAY throw this exception if the object is a non-current Document Version.

3.4.15 deleteContentStream

Description: Deletes the content stream for the specified Document object.

Notes: A Repository MAY automatically create new Document versions as part of this service method. Therefore, the DocumentId output NEED NOT be identical to the DocumentId input.

3.4.15.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID documentId:** The identifier for the Document object.

Optional:

- **String changeToken:** See “3.1.5 Change Tokens” section.

3.4.15.2 Outputs

- **ID documentId:** The ID of the Document object.

3.4.15.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `constraint`**:** The Repository SHALL throw this exception if ANY of the following conditions are met:
    - o The Object’s Object-Type definition “contentStreamAllowed” attribute is set to “*required*”.
- `storage`: See “3.1.6.2 Specific Exceptions” section.
- `updateConflict`: See “3.1.6.2 Specific Exceptions” section.
- `versioning`: The repository MAY throw this exception if the object is a non-current Document Version.

3.5 Multi-filing Services

The Multi-filing services (*addObjectToFolder, removeObjectFromFolder)* are supported only if the repository supports the multifiling or unfiling optional capabilities. The Multi-filing Services are used to file/un-file objects into/from folders**.**

This service is NOT used to create or delete objects in the repository.

3.5.1 addObjectToFolder

Description: Adds an existing fileable non-folder object to a folder.

3.5.1.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID objectId:** The identifier of the object.
- **ID folderId:** The folder into which the object is to be filed.

3.5.1.2 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions.

- `constraint`**:** The Repository SHALL throw this exception if ANY of the following conditions are met:
- The typeId is NOT in the list of AllowedChildObjectTypeIds of the parent-folder specified by folderId.

3.5.2 removeObjectFromFolder

Description: Removes an existing fileable non-folder object from a folder.

3.5.2.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID objectId:** The identifier of the object.

Optional:

- **ID folderId:** The folder from which the object is to be removed.
  - o If no value is specified, then the Repository SHALL remove the object from all folders in which it is currently filed.

3.5.2.2 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions

3.6 Discovery Services

The Discovery Services (*query*) are used to search for query-able objects within the Repository.

3.6.1 Query

Description: Executes a CMIS-SQL query statement against the contents of the Repository.

3.6.1.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **String statement:** CMIS-SQL query to be executed. (See “2.11 Query” section.)

Optional:

- **Boolean searchAllVersions:**
  - o If TRUE, then the Repository SHALL include latest and non-latest versions of document objects in the query search scope.
  - o If FALSE **(default)**, then the Repository SHALL only include latest versions of documents in the query search scope.
    - ▪ If the Repository does not support the optional `capabilityAllVersionsSearchable` capability, then this parameter value MUST be set to FALSE.
- **Enum includeRelationships:** See “3.1.4 Include Relationships” section.
  - o Note: For query statements where the SELECT clause contains properties from only one virtual table reference (i.e. referenced object-type), any value for this enum may be used. If the SELECT clause contains properties from more than one table, then the value of this parameter SHALL be “`none”.`
- **Boolean includeAllowableActions:** See “3.1.3 Allowable Actions” section.

    - Note: For query statements where the SELECT clause contains properties from only one virtual table reference (i.e. referenced object-type), any value for this parameter may be used. If the SELECT clause contains properties from more than one table, then the value of this parameter SHALL be "FALSE".
- **Integer maxItems:** See 3.1.2 Paging section..
- **Integer skipCount:** See 3.1.2 Paging section..

3.6.1.2 Outputs

- **<Array> Object QueryResults:** The set of results for the query. (See "2.11 Query" section.)
- **Boolean hasMoreItems:** See 3.1.2 Paging section..

3.6.1.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions

3.6.2 getContentChanges

Description: Gets a list of content changes. This service is intended to be used by search crawlers or other applications that need to efficiently understand what has changed in the repository.

Notes:

- The content stream is NOT returned for any change event.
- The definition of the authority needed to call this service is repository specific.
- The latest change log token for a repository can be acquired via the getRepositoryInfo service.

3.6.2.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.

Optional:

- **String changeLogToken:**
    - If specified, then the Repository SHALL return the change event corresponding to the value of the specified change log token as the first result in the output.
    - If not specified, then the Repository SHALL return the first change event recorded in the change log.
- **Boolean includeProperties:**
    - If TRUE, then the Repository SHALL include the updated property values for "updated" change events.
    - If FALSE (default), then the Repository SHALL NOT include the updated property values for "updated" change events.
- **Integer maxItems:** See 3.1.2 Paging section.

3.6.2.2 Outputs

- **<Array> changeEvents:** A collection of CMIS objects that SHALL include the information as specified in 2.12.3.
- **String lastChangeLogToken:** The change token corresponding to the last change event in changeEvents.
- **Boolean hasMoreItems**: See 3.1.2 Paging section.

3.6.2.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `constraint:` The Repository SHALL throw this exception if the event corresponding to the change log token provided as an input parameter is no longer available in the change log. (E.g. because the change log was truncated).

3.7 Versioning Services

The Versioning services (checkOut, cancelCheckOut, getPropertiesOfLatestVersion, getAllVersions, deleteAllVersions) are used to navigate or update a Document Version Series.

3.7.1 checkOut

Description: Create a private working copy of the object.

3.7.1.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID documentId:** The identifier of the object.

3.7.1.2 Outputs

- **ID documentId:** The identifier for the "Private Working Copy" object.
- **Boolean contentCopied:** TRUE if the content-stream of the Private Working Copy is a copy of the contentStream of the Document that was checked out.
  - FALSE if the content-stream of the Private Working Copy is "not set".

3.7.1.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `constraint`**:** The Repository SHALL throw this exception if ANY of the following conditions are met:
  - The Document's Object-Type definition's *versionable* attribute is FALSE.
- `storage`: See "3.1.6.2 Specific Exceptions" section.
- `updateConflict`: See "3.1.6.2 Specific Exceptions" section.
- `versioning`: The repository MAY throw this exception if the object is a non-current Document Version.

3.7.2 cancelCheckOut

Description: Reverses the effect of a check-out. Removes the private working copy of the checked-out document object, allowing other documents in the version series to be checked out again.

3.7.2.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID documentId:** The identifier of the object.

3.7.2.2 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `constraint`**:** The Repository SHALL throw this exception if ANY of the following conditions are met:
  - The Document's Object-Type definition's *versionable* attribute is FALSE.
- `updateConflict`: See "3.1.6.2 Specific Exceptions" section.
- `versioning`: The repository MAY throw this exception if the object is a non-current Document Version.

3.7.3 checkIn

Description: Checks-in the Private Working Copy object.

Notes:

- For repositories that do NOT support the optional "`capabilityPWCUpdateable`" capability, the *properties* and *contentStream* input parameters MUST be provided on the checkIn method for updates to happen as part of checkIn.

3.7.3.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID documentId:** The identifier of the object.

Optional:

- **Boolean major:** TRUE **(default)** if the checked-in Document Object SHALL be a major version.
  - FALSE if the checked-in Document Object SHALL NOT be a major version.
- **<Array> properties:** The property values that SHALL be applied to the checked-in Document Object.
- **<contentStream> contentStream:** The Content Stream that SHALL be stored for the checked-in Document Object. The method of passing the contentStream to the server and the encoding mechanism will be specified by each specific binding.
- String checkinComment: See "2.10.5 Versioning Properties on Document Objects" section.

3.7.3.2 Outputs

- **ID documentId:** The ID of the checked-in document.

3.7.3.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `constraint`**:** The Repository SHALL throw this exception if ANY of the following conditions are met:
  - The Document's Object-Type definition's *versionable* attribute is FALSE.
- `storage`: See "3.1.6.2 Specific Exceptions" section.
- `streamNotSupported`**:** The Repository SHALL throw this exception if the Object-Type definition specified by the typeId parameter's "contentStreamAllowed" attribute is set to "not allowed" and a contentStream input parameter is provided.
- `updateConflict`: See "3.1.6.2 Specific Exceptions" section.

3.7.4 getPropertiesOfLatestVersion

Description: Get a subset of the properties for the latest Document Object in the Version Series.

Notes:

- If the input parameter major is TRUE and the Version Series contains no major versions, then the Repository SHALL throw objectNotFoundException.

3.7.4.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID versionSeriesId:** The identifier for the Version Series.

Optional:

- **Boolean major:** If TRUE **(default)**, then the Repository SHALL returnthe properties for the latest major version object in the Version Series.
  - o If FALSE, the Repository SHALL return the properties for the latest (major or non-major) version object in the Version Series.
- **String filter:** See "3.1.1 Property Filters" section.

3.7.4.2 Outputs

- **<Array> Properties:** The list of properties for the object.

3.7.4.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `filterNotValid`: The Repository SHALL throw this exception if this property filter input parameter is not valid.

3.7.5 getAllVersions

Description: Returns the list of all Document Objects in the specified Version Series, sorted by CREATION_DATE descending.

Notes:

- The result set for this operation SHALL include the Private Working Copy, subject to caller's access privileges.

3.7.5.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID versionSeriesId:** The identifier for the Version Series.

Optional:

- **Boolean includeAllowableActions:** See "3.1.3 Allowable Actions" section.
- **Enum includeRelationships:** See "3.1.4 Include Relationships" section.
- **String filter:** See "3.1.1 Property Filters" section.

3.7.5.2 Outputs

- **<Array> Objects:** A list of Document Objects in the specified Version Series.

3.7.5.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `filterNotValid`**:** The Repository SHALL throw this exception if this property filter input parameter is not valid.

3.7.6 deleteAllVersions

Description: Deletes all Document Objects in the specified Version Series, including the Private Working Copy.

Notes: If for any reason the repository is unable to delete all versions in the version series, the repository SHALL leave the version series in a valid state as defined by the CMIS domain model.

3.7.6.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID versionSeriesId:** The identifier of the Version Series to be deleted.

3.7.6.2 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions

3.8 Relationships Services

The Relationship Services (*getRelationships*) are used to retrieve the dependent Relationship objects associated with an independent object.

3.8.1 getRelationships

Description: Gets all or a subset of relationships associated with an independent object.

3.8.1.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID objectId:** The identifier of the object.**Boolean includeSubRelationshipTypes:** If TRUE, then the Repository SHALL return all relationships whose Object-Types are descendant-types of *typeId.*
  - If FALSE, then the Repository SHALL only return relationships whose Object-Type is *typeId.*
-

Optional:

- **Enum** `relationshipDirection`: An enumeration specifying whether the Repository SHALL return relationships where the specified Object is the source of the relationship, the target of the relationship, or both. Valid values are:
  - `source`**: (default)** The Repository SHALL return only relationship objects where the specified object is the source object.
  - `target`**:** The Repository SHALL return only relationship objects where the specified object is the target object.
  - `either`**:** The Repository SHALL return relationship objects where the specified object is either the source or the target object.
- **Boolean includeAllowableActions:** See “3.1.3 Allowable Actions” section.

- **ID typeId:** If specified, then the Repository SHALL return only relationships whose Object-Type is of the type specified (and possibly its descendant-types – see next parameter.)
    - If not specified, then the repository SHALL return Relationship objects of all types.
- **String filter:** See "3.1.1 Property Filters" section.
- **Integer maxItems:** See 3.1.2 Paging section..
- **Integer skipCount:** See 3.1.2 Paging section..

3.8.1.2 Outputs

- **<Array> Objects:** A list of the relationship objects.
- **Boolean hasMoreItems:** See 3.1.2 Paging section..

3.8.1.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `filterNotValid`**:** The Repository SHALL throw this exception if this property filter input parameter is not valid.

3.9 Policy Services

The Policy Services *(applyPolicy, removePolicy, getAppliedPolicies)* are used to apply or remove a policy object to a controllablePolicy object.

3.9.1 applyPolicy

Description: Applies a specified policy to an object.

3.9.1.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID policyId:** The identifier for the Policy to be applied.
- **ID objectId:** The identifier of the object.

3.9.1.2 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `constraint` **:** The Repository SHALL throw this exception if the specified object's Object-Type definition's attribute for *controllablePolicy* is FALSE.

3.9.2 removePolicy

Description: Removes a specified policy from an object.

3.9.2.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID policyId:** The identifier for the Policy to be removed.
- **ID objectId:** The identifier of the object.

3.9.2.2 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `constraint`**:** The Repository SHALL throw this exception if the specified object's Object-Type definition's attribute for *controllablePolicy* is FALSE.

3.9.3 getAppliedPolicies

Description: Gets the list of policies currently applied to the specified object.

3.9.3.1 Inputs

Required:

- **ID repositoryId:** The identifier for the Repository.
- **ID objectId:** The identifier of the object.

Optional:

- **String filter:** See "3.1.1 Property Filters" section.

3.9.3.2 Outputs

- **<Array> Objects:** A list of Policy Objects.

3.9.3.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `filterNotValid`**:** The Repository SHALL throw this exception if this property filter input parameter is not valid.

3.10 ACL Services

3.10.1 getACL

Description: Get the ACL currently applied to the specified document or folder object.

3.10.1.1 Inputs

Required:

- **ID repositoryId:** The identifier for the repository.
- **ID objectId:** The identifier for the object
- **Boolean includeAllowableActions:** See "Allowable Actions" section.

Optional:

- **Boolean onlyBasicPermissions:** See "Access Control Lists" section. The repository SHOULD make a best effort to fully express the native security applied to the object
  - o `TRUE:` (default value if not provided) indicates that the client requests that the returned ACL be expressed using only the CMIS Basic permissions.
  - o `FALSE:` indicates that the server may respond using either solely CMIS Basic permissions, or repository specific permissions or some combination of both.

3.10.1.2 Outputs

- **<Array> AccessControlEntryType:** The list of access control entries of the ACL for the object.

Optional:

- **Boolean exact:** An indicator that the ACL returned fully describes the permission for this object – i.e. there are no other security constraints applied to this object. Not provided defaults to FALSE.

3.10.1.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions

3.10.1.4 Notes

This service MUST be supported by a repository, if *getRepository* returns *capabilityACL=discover* or *=manage*.

How an ACL for the object is computed is up to the repository. A client MUST NOT assume that the ACEs from the ACL as returned by this service can be applied via *applyACL*.

3.10.2 applyACL

Description: Adds or removes the given ACEs to or from the ACL of document or folder object.

3.10.2.1 Inputs

Required:

- **ID repositoryId:** The identifier for the repository.
- **ID objectId:** The identifier for the object
- **<Array> AccessControlEntryType addACEs:** The ACEs to be added.
- **<Array> AccessControlEntryType removeACEs:** The ACEs to be removed.

Optional:

- **Enum ACLPropagation:** Specifies how ACEs should be handled:
    - o `objectonly:` ACEs must be applied without changing the ACLs of other objects.
    - o `propagate:` ACEs must be applied by propagate the changes to all “inheriting” objects.
    - o `repositorydetermined:` indicates that the client leaves the behavior to the repository.

3.10.2.2 Outputs

- **<Array> AccessControlEntryType:** The list of access control entries of the resulting ACL for the object

Optional:

- **Boolean exact:** An indicator that the ACL returned fully describes the permission for this object – i.e. there are no other security constraints applied to this object. Not provided defaults to FALSE.

o **String changeToken:** See “Change Tokens” section.

3.10.2.3 Exceptions Thrown & Conditions

- 3.1.6.1 General Exceptions
- `constraint:` The Repository SHALL throw this exception if ANY of the following conditions are met:
    - o The specified object’s Object-Type definition’s attribute for *controllableACL* is FALSE.
    - o The value for *ACLPropagation* does not match the values as returned via *getACLCapabilities*.
    - o At least one of the *spec*ified values for *permission* in ANY of the *ACEs does not match* ANY of the *permissionNames* as returned by *getACLCapability and is not a CMIS Basic permission*

3.10.2.4 Notes

This service MUST be supported by a repository, if *getRepository* returns *capabilityACL=manage*.

How ACEs are added or removed to or from the object is up to the repository – with respect to the *ACLPropagation* provided by the client. For “shared” ACEs (e.g. via inheritance), the repository MAY merge the ACEs provided with the ACEs of the ACL already applied to the object (i.e. the ACEs provided MAY not be completely added or removed from the effective ACL for the object).

4 Schema

Below is the CMIS-Core schema:

```
Placeholder for schema to be included for Public Review Draft.  See Schema zip
file in the meantime.
```

# Conformance

The last numbered section in the specification must be the Conformance section. Conformance Statements/Clauses go here.

A. Acknowledgements

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Participants:

[Participant Name, Affiliation | Individual Member]
[Participant Name, Affiliation | Individual Member]

B. Non-Normative Text

C. Revision History

Revision	Date	Editor	Changes Made
0.62c	6/29/2009	Ryan McVeigh	• Fixed several spec bugs tracked in JIRA. • Cleaned up Document Map / TOC.
0.62b	5/29/2009	Ryan McVeigh	• Re-factored spec sections for inaccuracies, typos. • Formatting issues and consistency. • Revised to match 0.5 for optional vs. required parameters, etc. where the spec changed without an issue to track the change.
0.62a	5/19/2009	Ethan Gur-esh	• Re-factored spec to remove tables • Include CMIS core schema in Part I rather than a separate file. • Fixed several spec bugs tracked in JIRA. •
0.6	3/8/2009	Ethan Gur-esh	• Re-wrote spec text into normative style. • Re-factored spec sections for clarity. • Fixed several spec bugs tracked in JIRA. Issues fixed for this revision: http://tools.oasis-open.org/issues/secure/IssueNavigator.jspa?reset=true&pid=10021&fixfor=10009