

Content Management Interoperability Services – Domain Model Version 0.6

DRAFT

25 February 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] .html)
[http://docs.oasis-open.org/\[tc-short-name\] / \[additional path/filename\] .doc](http://docs.oasis-open.org/[tc-short-name] / [additional path/filename] .doc)
[http://docs.oasis-open.org/\[tc-short-name\] / \[additional path/filename\] .pdf](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] .html)
[http://docs.oasis-open.org/\[tc-short-name\] / \[additional path/filename\] .doc](http://docs.oasis-open.org/[tc-short-name] / [additional path/filename] .doc)
[http://docs.oasis-open.org/\[tc-short-name\] / \[additional path/filename\] .pdf](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] .html)
[http://docs.oasis-open.org/\[tc-short-name\] / \[additional path/filename\] .doc](http://docs.oasis-open.org/[tc-short-name] / [additional path/filename] .doc)
[http://docs.oasis-open.org/\[tc-short-name\] / \[additional path/filename\] .pdf](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

Related work:


This specification replaces or supercedes:

- 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:

This document was last revised or approved by the CMIS TC on the above date. The level of approval is also listed above. Check the "Latest Version" or "Latest Approved Version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at <http://www.oasis-open.org/committees/CMIS/>.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (<http://www.oasis-open.org/committees/CMIS/ipr.php>).

The non-normative errata page for this specification is located at <http://www.oasis-open.org/committees/CMIS/>.

Notices

Copyright © OASIS® 2008. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The names "OASIS", [insert specific trademarked names and abbreviations here] are trademarks of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see <http://www.oasis-open.org/who/trademark.php> for above guidance.


Table of Contents

1	Introduction.....	7
1.1	Terminology.....	7
1.2	Normative References.....	7
1.3	Non-Normative References.....	7
2	Data Model.....	9
2.1	Repository.....	9
2.1.1	Optional Capabilities.....	9
2.1.2	Related Repositories.....	11
2.1.3	Implementation Information.....	11
2.2	Object.....	11
2.2.1	Property.....	12
2.3	Document Object and Content Stream.....	14
2.3.1	Content Stream.....	14
2.4	Folder Object.....	14
2.4.1	File-able Objects.....	15
2.4.2	Folder Hierarchy.....	16
2.4.3	Folder Paths in CMIS.....	17
2.5	Relationship Object.....	18
2.6	Policy Object.....	19
2.7	Object-Type.....	20
2.7.1	Object-Type Hierarchy and Inheritance.....	21
2.7.2	Object-Type Attributes.....	21
2.7.3	Object-Type Property Definitions.....	24
2.7.4	CMIS Base Object-Type Definitions.....	28
2.8	Versioning.....	39
2.8.1	Version Series.....	39
2.8.2	Latest Version.....	39
2.8.3	Major Versions.....	39
2.8.4	Services that modify Version Series.....	40
2.8.5	Versioning Properties on Document Objects.....	41
2.8.6	Object Creation and Initial Versioning State.....	42
2.8.7	Version Specific/Independent membership in Folders.....	42
2.8.8	Version Specific/Independent membership in Relationships.....	43
2.8.9	Versioning visibility in Query Services.....	43
2.9	Query.....	44
2.9.1	Relational View Projection of the CMIS Data Model.....	44
2.9.2	CMIS-SQL Definition.....	46
3	Services.....	54
3.1	Common Service Elements.....	54
3.1.1	Property Filters.....	54
3.1.2	Paging.....	54

3.1.3 Allowable Actions	55
3.1.4 Change Tokens	57
3.1.5 Table of Exceptions	57
3.2 Repository Services	61
3.2.1 getRepositories	61
3.2.2 getRepositoryInfo	61
3.2.3 getTypes	62
3.2.4 getTypeDefinition	63
3.3 Navigation Services	63
3.3.1 getDescendants	63
3.3.2 getChildren	64
3.3.3 getFolderParent	65
3.3.4 getObjectParents	65
3.3.5 getCheckedoutDocs	66
3.4 Object Services	66
3.4.1 createDocument	66
3.4.2 createFolder	67
3.4.3 createRelationship	68
3.4.4 createPolicy	69
3.4.5 getAllowableActions	69
3.4.6 getProperties	70
3.4.7 getContentStream	70
3.4.8 updateProperties	70
3.4.9 moveObject	71
3.4.10 deleteObject	72
3.4.11 deleteTree	72
3.4.12 setContentStream	73
3.4.13 deleteContentStream	74
3.5 Multi-filing Services	74
3.5.1 addObjectToFolder	74
3.5.2 removeObjectFromFolder	75
3.6 Discovery Services	75
3.6.1 query	75
3.7 Versioning Services	76
3.7.1 checkOut	76
3.7.2 cancelCheckOut	76
3.7.3 checkIn	77
3.7.4 getPropertiesOfLatestVersion	78
3.7.5 getAllVersions	78
3.7.6 deleteAllVersions	79
3.8 Relationships Services	79
3.8.1 getRelationships	79
3.9 Policy Services	80

3.9.1 applyPolicy	80
3.9.2 removePolicy	80
3.9.3 getAppliedPolicies	81
# Conformance.....	82
A. Acknowledgements	83
B. Non-Normative Text	84
C. Revision History.....	85

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", "MAY", 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.
- [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 an 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.

Capability Name	Description	See section
Multifiling	Ability for an application to file a document or other file-able object in more than one folder	Folder Object
Unfiling	Ability for an application to leave a document or other file-able object not filed in any folder	Folder Object
VersionSpecificFiling	Ability for an application to file individual versions (i.e., not all versions) of a document in a folder	Versioning
PWCUpdateable	Ability for an application to update the “Private Working Copy” of a checked-out document	Versioning

PWCSearchable	Ability of the Repository to include the "Private Working Copy" of checked-out documents in query search scope; otherwise PWC's are not searchable	Versioning
AllVersionsSearchable	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	Versioning
Query (non-Boolean)	Indicates the types of queries that the Repository has the ability to fulfill. Query support levels are: <ul style="list-style-type: none"> • 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. • Both: The Repository can fulfill queries that filter based on object metadata and/or document full-text content. 	Query
 Text (non-Boolean)	Indicates the types of full-text queries that the Repository has the ability to fulfill. FullText support levels are: <ul style="list-style-type: none"> • None: The Repository cannot fulfill any queries that filter based on full-text content of documents. • FullTextOnly: Only queries that filter based on the full-text content of documents can be fulfilled. • FullTextAndStructured: The repository can fulfill queries that filter on both the full-text content of documents and their properties. 	Query
Join (non-Boolean)	Indicates the types of SQL JOIN keywords that the Repository can fulfill in queries. Support levels are: <ul style="list-style-type: none"> • NoJoin: The repository cannot fulfill 	Query

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.

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, an URI, and a Repository Relationship Name are returned. The Relationship Name is opaque to CMIS. Suggested/example values for the “Relationship Name” include:

- “Parent”, “Child”, “Sibling”: 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

Furthermore, 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.

Note: Besides “Get Repository Information”, an application can also get object-type definitions using the “Get Types” and “Get Type Definition” services ([getTypes](#), [getTypeDefinition](#)).

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 “controllable” 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.

Besides ID, a repository MAY assign a **Uniform Resource Identifier** (URI), as defined by [RFC3986](#), to an object. A URI allows an application to access an object as a web resource using web protocols and tools. However, such accesses beyond the protocol bindings specified by CMIS are outside the scope of CMIS. URIs assigned by a repository to objects MAY be permanent.

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. A content-stream is not named. Instead, it has a repository-assigned URI.

Properties and content-streams cannot 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)

- Decimal (xsd:decimal)
- Integer (xsd:integer)
- Boolean (xsd:boolean)
- 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)

2.2.1.1 ID Property

An ID property holds a system-generated, read-only identifier, 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 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 valid fragment of Extensible Markup Language (XML) content.

2.2.1.3 HTML Property

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

2.2.1.4 Property Attributes

Besides name, type, and whether single- or multi-valued, a property has other defining attributes. They are described in the “[Object-Type](#)” section.

2.3 Document Object and Content Stream

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 folders 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 “Policy Object” 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 handled as properties of the content-stream’s containing document object. There is no MIME-type-specific attribute for a content-stream.

A content-stream is unnamed. But it has a repository-assigned URI which is distinct from the URI (if there is one) assigned to the document object that contains the content-stream. The permanency of this URI is repository-specific.

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 “Check-in” service MAY also take a content-stream as an optional input (see the “Versioning” section). 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.

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 “[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.

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 table defines whether the base CMIS Object Types are file-able:

Base Object-Type	File-able?
Folder	SHALL be file-able
Document	<ul style="list-style-type: none"> 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
Relationship	SHALL NOT be file-able
Policy	MAY be file-able

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

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

Multi-filing capability supported?	Un-filing capability Supported?	Allowed number of Parent Folders
False	False	1
False	True	0 or 1
True	False	1 or more
True	True	0 or more

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 *AllowedChildObjectTypes* 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.

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:

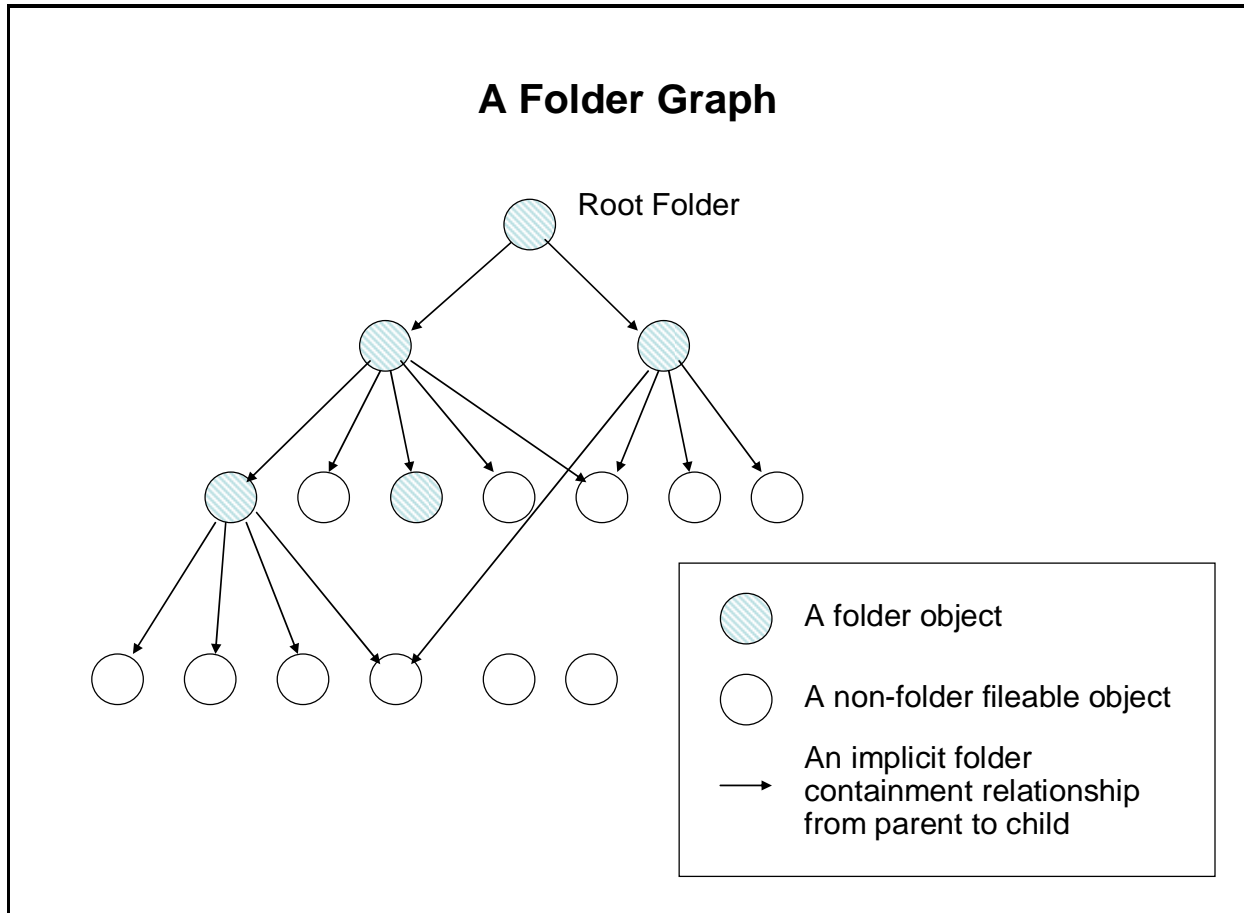
- Every folder object, except for one which is called the **Root Folder**, MUST have one and only one parent folder. The Root Folder does not have a parent.
- A cycle in folder containment relationships is not allowed. That is, a folder object cannot have itself as one of its descendant objects.
- A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



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 can not be created, deleted, or moved using CMIS services. Otherwise, it behaves like any other folder object.


2.4.3 Folder Paths in CMIS

The CMIS domain model does not include a notion of “path” as a means of addressing/accessing an object. Objects in CMIS are identified and accessed using their IDs in all CMIS services, and content-streams MAY be retrieved via a URI.

Individual protocol bindings for CMIS MAY allow for accessing/identifying objects or content-streams via URIs as is appropriate to the specific protocol. However, as is the case for Object IDs, those URIs NEED NOT be interpretable syntactically or assumed to be collate-able.

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.

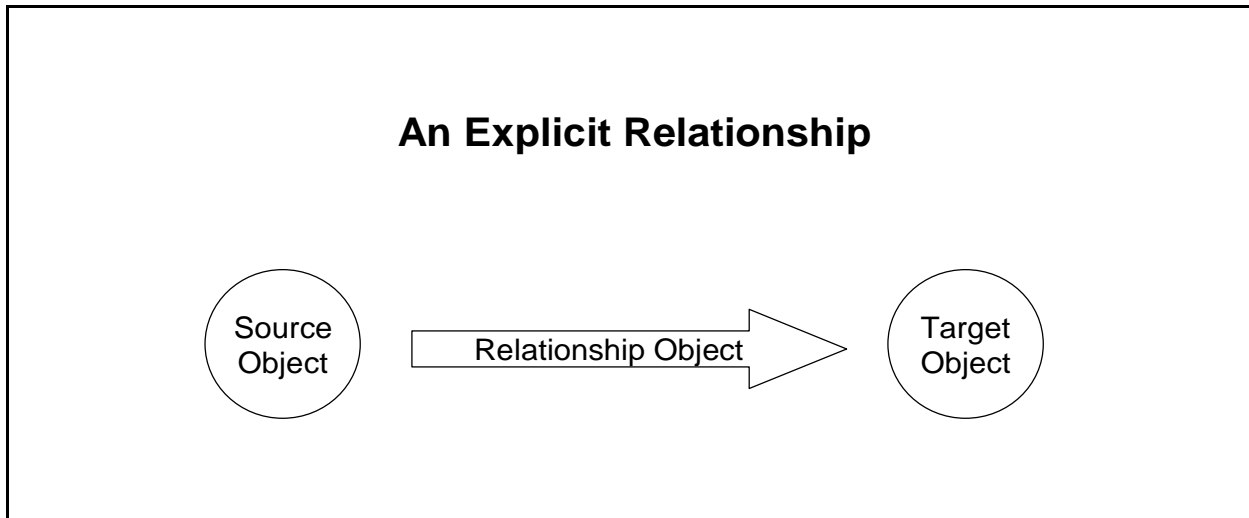
A **Relationship Object** instantiates an explicit, binary, directional, non-invasive, and typed relationship between a **Source Object** and a **Target Object**. The source object and the target object MUST both be an independent objects, such as a document object, a folder object, or a policy object. 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 name, 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-specific or a version-independent binding as described above). This traversal may be selectively 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 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 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.

2.7.1 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:
 - *Document* object type
 - *Folder* object type
 - *Relationship* object type
 -  *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. 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 “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.

2.7.2 Object-Type Attributes

2.7.2.1 Attributes common to ALL Object-Type Definitions

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

Name	Type	Function
------	------	----------

<i>typeId</i>	ID	This attribute uniquely identifies this object type in the repository
<i>queryName</i>	String	Used as a table name in a SQL query. It MAY be in mixed case, but SHALL uniquely identify this object type within the repository case-insensitively, and SHALL conform to the syntax rules for SQL identifiers.
<i>displayName</i>	String	Used for presentation by application
<i>baseType</i>	Enum	A value that indicates whether the base type for this Object-Type is the Document, Folder, Relationship, or Policy base type.
<i>baseTypeQueryName</i>	String	The <i>queryName</i> of the base type for the Object-Type.
<i>parentId</i>	ID	The ID of the Object-Types immediate parent type. It SHALL be “not set” for a base type.
<i>description</i>	String	Description of this object type, such as the nature of content, or its intended use. Used for presentation by application.
<i>creatable</i>	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.
<i>fileable</i>	Boolean	Indicates whether or not objects of this type are file-able .
<i>queryable</i>	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.
<i>includedInSupertypeQuery</i>	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.



controllable

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

Note: In the above table, the “applicable to sub-types of <base type>” is to be read as follows:

- **Yes:** This attribute is applicable to sub-types of this base type, although its value will vary between individual sub-types of the base type.
- **No:** This attribute SHALL always be returned as “not set” for all sub-types of this base-type.
- **TRUE:** This Boolean attribute SHALL always be TRUE for all sub-types of this base type.
- **FALSE:** This Boolean attribute SHALL always be FALSE for all sub-types of this base type.

2.7.2.2 Attributes specific to Document Object-Types

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

Name	Type	Function
<i>versionable</i>	Boolean	Indicates whether or not objects of this type are version-able. (See versioning section.)
<i>contentStreamAllowed</i>	Enum	A value that indicates whether a content-stream MAY, SHALL, or SHALL NOT be included in objects of this type. Values: <ul style="list-style-type: none"> • <i>notAllowed:</i> A content-stream SHALL NOT be included • <i>allowed:</i> A content-stream MAY be included • <i>required:</i> A content-stream SHALL be included (i.e. SHALL be included when the object is created, and SHALL NOT be deleted.)

2.7.2.3 Attributes specific to Relationship Object-Types

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

Name	Type	Function
<i>allowedSourceTypes</i>	String (multi-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.
<i>allowedTargetTypes</i>	String (multi-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.

2.7.3 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.

2.7.3.1 Property Types

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)
- Decimal (xsd:decimal)
- Integer (xsd:integer)
- Boolean (xsd:boolean)
- 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)

2.7.3.2 Attributes common to ALL Object-Type Property Definitions

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

Name	Type	Function
<i>name</i>	string	Identifies this property among all properties of this object type, including inherited properties. This attribute is also used as a column name in a SQL query. It MAY be in mixed case, but SHALL uniquely identify this property case-insensitively, and SHALL conform to the syntax rules for SQL identifiers.
<i>id</i>	string	This attribute contains a system-assigned ID which uniquely identifies this property.
<i>displayName</i>	string	Used for presentation by application.
<i>description</i>	string	This is an optional attribute containing a description of the property

<i>propertyType</i>	Enum	<p>This attribute indicates the type of this property. It MUST be one of the allowed property types. (See the “Property” section.)</p>
<i>cardinality</i>	Enum	<p>Indicates whether the property can have “zero or one” or “one or more” values.</p> <p>Values:</p> <ul style="list-style-type: none"> • 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). <p>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.</p>
<i>updateability</i>	Enum	<p>Indicates under what circumstances the value of this property MAY be updated.</p> <p>Values:</p> <ul style="list-style-type: none"> • 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. <ul style="list-style-type: none"> ○ 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 <i>updateProperties</i> service. • Whencheckedout: The property value SHALL only be update-able using a “private working copy” Object. <ul style="list-style-type: none"> ○ I.e. the update is either made on a “private working copy” object or made using a “check in” service.
<i>Inherited</i>	Boolean	Indicates whether the property is inherited from the parent-type or it is explicitly defined for this object-type.
<i>Required</i>	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.

queryable Boolean

Indicates whether or not the property MAY appear in the WHERE clause of 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 to the 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 <PropertyType 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 specified in the form of <choice name [, value [, index]] <[,choice+]>>. The name is used for presentation purpose. The value will be stored in the property when selected. The index provides guidance for ordering of names 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.

2.7.3.3 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:

Name	Type	Function
------	------	----------

<i>minValue</i>	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.

<i>maxValue</i>	integer	The maximum 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.

2.7.3.4 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:

Name	Type	Function
------	------	----------

<i>precision</i>	Integer	This is the precision in bits supported for values of this property. Valid values for this attribute are:
------------------	---------	---

- 32: 32-bit precision.
- 64: 64-bit precision.

<i>minValue</i>	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.

<i>maxValue</i>	Decimal	The maximum 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.



2.7.3.5 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:

Name	Type	Function
<i>maxLength</i>	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.

2.7.3.6 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:

Name	Type	Function
<i>schemaURI</i>	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.
<i>encoding</i>	string	The encoding that SHALL be used for the property value (e.g. UTF-8, etc.). If an application attempts to set the value of this property to an XML value that is NOT encoded in the format specified, the repository SHALL throw a ConstraintViolation exception.

2.7.4 CMIS Base Object-Type Definitions

This section specifies the Object-Type Definitions for each of the four CMIS base types ([Document](#), [Folder](#), [Relationship](#), [Policy](#)).

2.7.4.1 Document Object-Type Definition

2.7.4.1.1 Document Object-Type 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 table SHALL be followed for the Object-Type definition.

Attribute Name	Value
<i>typeId</i>	<repository-specific>
<i>queryName</i>	Document
<i>displayName</i>	<repository-specific>
<i>baseType</i>	document
<i>baseTypeQueryName</i>	Document
<i>parentId</i>	Not set
<i>description</i>	<repository-specific>
<i>creatable</i>	<repository-specific>
<i>fileable</i>	<ul style="list-style-type: none"> ○ If the repository does NOT support the “un-filing” capability: <ul style="list-style-type: none"> ○ TRUE ○ If the repository does support the “un-filing” capability: <ul style="list-style-type: none"> ○ <repository-specific>, but SHOULD be TRUE
<i>queryable</i>	SHOULD be TRUE
<i>includedInSupertypeQuery</i>	TRUE
<i>Controllable</i>	<repository-specific>
<i>Versionable</i>	<repository-specific>
<i>contentStreamAllowed</i>	<repository-specific>


2.7.4.1.2 Document Object-Type Property Definitions

The Document base Object-Type SHALL have the following property definitions, and MAY include additional property definitions.

Notes:

- The “functional summary” column is purely informational (these values do not actually appear in the Object-Type property definitions).
- The value RS indicates that the value indicates that the value property definition attribute MAY be set to any valid value for the attribute type.

- o The value T indicates the Boolean value SHALL be TRUE.
- o The value F indicates the Boolean value SHALL be FALSE.
- o The value S indicates the value SHALL be “single”.
- o The value M indicates the value SHALL be “multi”.
- o The value RO indicates the value SHALL be “readOnly”
- o The value RW indicates the value SHALL be “readWrite”
- o The value WC indicates the value SHALL be “whenCheckedout”
- o The value NS indicates the value SHALL be “not set”
- o The value NA indicates that this attribute does not exist for a property of the specified propertyType.

Name	Functional summary	Id	displayName	description	propertyType	cardinality	updateability	inherited	required	queryable	orderable
ObjectId	Uniquely identifies the object	RS	RS	RS	ID	S	RO	F	F	RS	RS
Uri	URI for accessing the object.	RS	RS	RS	Uri	S	RO	F	F	RS	RS
ObjectTypeId	ID of the object type for the object.	RS	RS	RS	ID	S	RO	F	F	RS	RS
CreatedBy	User who created the object.	RS	RS	RS	String	S	RO	F	F	RS	RS
CreationDate	DateTime when the object was created.	RS	RS	RS	DateTime	S	RO	F	F	RS	RS
LastModifiedBy	User who last modified the object.	RS	RS	RS	String	S	RO	F	F	RS	RS
LastModificationDate	DateTime when the object was last modified.	RS	RS	RS	DateTime	S	RO	F	F	RS	RS
ChangeToken	Opaque token used for optimistic locking & concurrency checking. (see Change Tokens section)	RS	RS	RS	String	S	RO	F	F	RS	RS
 Name	Name of the object.	RS	RS	RS	String	S	RS	F	RS	RS	RS
IsImmutable	TRUE if the repository SHALL throw an error at any attempt to update or delete the object.	RS	RS	RS	Boolean	S	RO	F	F	RS	RS

isLatestVersion	See versioning section.	RS	RS	RS	Boolean	S	RO	F	F	RS	RS
isMajorVersion	See versioning section.	RS	RS	RS	Boolean	S	RO	F	F	RS	RS
VersionLabel	See versioning section.	RS	RS	RS	String	RS	RO	F	F	RS	RS
VersionSeriesId	See versioning section.	RS	RS	RS	ID	S	RO	F	F	RS	RS
IsVersionSeriesCheckedOut	See versioning section.	RS	RS	RS	Boolean	S	RO	F	F	RS	RS
VersionSeriesCheckedOutBy	See versioning section.	RS	RS	RS	String	S	RO	F	F	RS	RS
VersionSeriesCheckedOutId	See versioning section.	RS	RS	RS	ID	S	RO	F	F	RS	RS
CheckinComment	See versioning section.	RS	RS	RS	String	S	RO	F	F	RS	RS
ContentStreamLength	Length of the content stream (in bytes).	RS	RS	RS	Integer	S	RO	F	F	RS	RS
ContentStreamMimeType	MIME type of the Content Stream	RS	RS	RS	String	S	RO	F	F	RS	RS
ContentStreamFilename	File name of the content stream	RS	RS	RS	String	S	RO	F	F	RS	RS
ContentStreamUri	URI for accessing the object.	RS	RS	RS	Uri	S	RO	F	F	RS	RS

2.7.4.2 Folder Object-Type Definition

2.7.4.2.1 Folder Object-Type Attribute Values

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

(Notes:

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

Attribute Name	Value
<i>typeId</i>	<repository-specific>
<i>queryName</i>	Folder


<i>displayName</i>	<repository-specific>
<i>baseType</i>	folder
<i>baseTypeQueryName</i>	Folder
<i>parentId</i>	Not set
<i>description</i>	<repository-specific>
<i>creatable</i>	<repository-specific>
<i>fileable</i>	TRUE
<i>queryable</i>	SHOULD be TRUE
<i>includedInSupertypeQuery</i>	TRUE
<i>controllable</i>	<repository-specific>

2.7.4.2.2 Folder Object-Type Property Definitions

The Document base Object-Type SHALL have the following property definitions, and MAY include additional property definitions.

Notes:

- The “functional summary” column is purely informational (these values do not actually appear in the Object-Type property definitions).
- The value RS indicates that the value indicates that the value property definition attribute MAY be set to any valid value for the attribute type.
- The value T indicates the Boolean value SHALL be TRUE.
- The value F indicates the Boolean value SHALL be FALSE.
- The value S indicates the value SHALL be “single”.
- The value M indicates the value SHALL be “multi”.
- The value RO indicates the value SHALL be “readOnly”
- The value RW indicates the value SHALL be “readWrite”
- The value WC indicates the value SHALL be “whenCheckedout”
- The value NS indicates the value SHALL be “not set”
- The value NA indicates that this attribute does not exist for a property of the specified propertyType.

Name	Functional summary	Id	displayName	description	propertyType	cardinality	updateability	inherited	required	queryable	orderable
ObjectId	Uniquely identifies the object	RS	RS	RS	ID	S	RO	F	F	RS	RS
Uri	URI for accessing the object.	RS	RS	RS	Uri	S	RO	F	F	RS	RS
ObjectTypeId	ID of the object type for the object.	RS	RS	RS	ID	S	RO	F	F	RS	RS
CreatedBy	User who created the object.	RS	RS	RS	String	S	RO	F	F	RS	RS
CreationDate	DateTime when the object was created.	RS	RS	RS	DateTime	S	RO	F	F	RS	RS
LastModifiedBy	User who last modified the object.	RS	RS	RS	String	S	RO	F	F	RS	RS
LastModificationDate	DateTime when the object was last modified.	RS	RS	RS	DateTime	S	RO	F	F	RS	RS
ChangeToken	Opaque token used for optimistic locking & concurrency checking (see Change Tokens section)	RS	RS	RS	String	S	RO	F	F	RS	RS
 Name	Name of the folder. For the root folder, the value of this field SHALL be "CMIS_Root_Folder".	RS	RS	RS	String	S	RS	F	RS	RS	RS
ParentId	ID of the parent Folder of the folder.	RS	RS	RS	ID	S	RO	F	F	RS	RS
AllowedChildObjectTypes	IDs of the set of Object Types that can be created, moved or filed into this folder.	RS	RS	RS	ID	M	RS	F	F	RS	RS

2.7.4.3 Relationship Object-Type Definition

2.7.4.3.1 Relationship Object-Type 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.

Attribute Name	Value
<i>typeId</i>	<repository-specific>
<i>queryName</i>	Relationship
<i>displayName</i>	<repository-specific>
<i>baseType</i>	Folder
<i>baseTypeQueryName</i>	Folder
<i>parentId</i>	Not set
<i>description</i>	<repository-specific>
<i>creatable</i>	<repository-specific>
<i>fileable</i>	FALSE
<i>queryable</i>	FALSE
<i>includedInSupertypeQuery</i>	TRUE
<i>controllable</i>	<repository-specific>
<i>allowedSourceTypes</i>	<repository-specific>
<i>allowedTargetTypes</i>	<repository-specific>

2.7.4.3.2 Relationship Object-Type Property Definitions

The Relationship base Object-Type SHALL have the following property definitions, and MAY include additional property definitions.

Notes:

- The “functional summary” column is purely informational (these values do not actually appear in the Object-Type property definitions).

- The value RS indicates that the value indicates that the value property definition attribute MAY be set to any valid value for the attribute type.
- The value T indicates the Boolean value SHALL be TRUE.
- The value F indicates the Boolean value SHALL be FALSE.
- The value S indicates the value SHALL be “single”.
- The value M indicates the value SHALL be “multi”.
- The value RO indicates the value SHALL be “readOnly”
- The value RW indicates the value SHALL be “readWrite”
- The value WC indicates the value SHALL be “whenCheckedout”
- The value NS indicates the value SHALL be “not set”
- The value NA indicates that this attribute does not exist for a property of the specified propertyType.

Name	Functional summary	Id	displayName	description	propertyType	cardinality	updateability	inherited	required	queryable	orderable
ObjectId	Uniquely identifies the object	RS	RS	RS	ID	S	RO	F	F	RS	RS
Uri	URI for accessing the object.	RS	RS	RS	Uri	S	RO	F	F	RS	RS
ObjectTypeId	ID of the object type for the object.	RS	RS	RS	ID	S	RO	F	F	RS	RS
CreatedBy	User who created the object.	RS	RS	RS	String	S	RO	F	F	RS	RS
CreationDate	DateTime when the object was created.	RS	RS	RS	DateTime	S	RO	F	F	RS	RS
LastModifiedBy	User who last modified the object.	RS	RS	RS	String	S	RO	F	F	RS	RS
LastModificationDate	DateTime when the object was last modified.	RS	RS	RS	DateTime	S	RO	F	F	RS	RS
ChangeToken	Opaque token used for optimistic locking & concurrency checking. (see Change Tokens section)	RS	RS	RS	String	S	RO	F	F	RS	RS
SourceId	ID of the source object of the relationship.	RS	RS	RS	ID	S	RO	F	F	RS	RS

TargetId	ID of the target object of the relationship.	RS	RS	RS	ID	S	RO	F	F	RS	RS
----------	--	----	----	----	----	---	----	---	---	----	----

2.7.4.4 Policy Object-Type Definition

2.7.4.4.1 Policy Object-Type Attribute Values

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

Notes:

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

Attribute Name	Value
<i>typeId</i>	<repository-specific>
<i>queryName</i>	Policy
<i>displayName</i>	<repository-specific>
<i>baseType</i>	Policy
<i>baseTypeQueryName</i>	Policy
<i>parentId</i>	Not set
<i>description</i>	<repository-specific>
<i>creatable</i>	<repository-specific>
<i>fileable</i>	<repository-specific>
<i>queryable</i>	<repository-specific>
<i>includedInSupertypeQuery</i>	TRUE
<i>controllable</i>	<repository-specific>

2.7.4.4.2 Policy Object-Type Property Definitions

The Policy base Object-Type SHALL have the following property definitions, and MAY include additional property definitions.

Notes:

- The “functional summary” column is purely informational (these values do not actually appear in the Object-Type property definitions).
- The value RS indicates that the value indicates that the value property definition attribute MAY be set to any valid value for the attribute type.
- The value T indicates the Boolean value SHALL be TRUE.
- The value F indicates the Boolean value SHALL be FALSE.
- The value S indicates the value SHALL be “single”.
- The value M indicates the value SHALL be “multi”.
- The value RO indicates the value SHALL be “readOnly”
- The value RW indicates the value SHALL be “readWrite”
- The value WC indicates the value SHALL be “whenCheckedout”
- The value NS indicates the value SHALL be “not set”
- The value NA indicates that this attribute does not exist for a property of the specified propertyType.

Name	Functional summary	Id	displayName	description	propertyType	cardinality	updateability	inherited	required	queryable	orderable
ObjectId	Uniquely identifies the object	RS	RS	RS	ID	S	RO	F	F	RS	RS
Uri	URI for accessing the object.	RS	RS	RS	Uri	S	RO	F	F	RS	RS
ObjectTypeId	ID of the object type for the object.	RS	RS	RS	ID	S	RO	F	F	RS	RS
CreatedBy	User who created the object.	RS	RS	RS	String	S	RO	F	F	RS	RS
CreationDate	DateTime when the object was created.	RS	RS	RS	DateTime	S	RO	F	F	RS	RS
LastModifiedBy	User who last modified the object.	RS	RS	RS	String	S	RO	F	F	RS	RS
LastModificationDate	DateTime when the object was last modified.	RS	RS	RS	DateTime	S	RO	F	F	RS	RS
ChangeToken	Opaque token used for optimistic locking & concurrency checking.(see Change Tokens section)	RS	RS	RS	String	S	RO	F	F	RS	RS

PolicyName	Display name for the Policy.	RS	RS	RS	String	S	RS	F	F	RS	RS
PolicyText	User-friendly description of the Policy.	RS	RS	RS	String	S	RS	F	F	RS	RS

2.8 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.

2.8.1 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.

2.8.2 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.

2.8.2.1 Behavioral constraints on non-Latest Versions

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

2.8.3 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.

2.8.4 Services that modify Version Series


2.8.4.1 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 *canCheckOut* 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.

2.8.4.2 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 support the “PWCUpdateable” capability, then the PWC object can only be modified as part of the *checkIn* service call.

2.8.4.3 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.

2.8.4.4 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.

2.8.5 Versioning Properties on Document Objects

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

Name	Type	Usage
<code>isLatestVersion</code>	Boolean	TRUE if the Document object is the <i>Latest Version</i> in its <i>Version Series</i> . FALSE otherwise.
<code>isMajorVersion</code>	Boolean	TRUE if the Document object is the <i>Latest Major Version</i> in its <i>Version Series</i> . FALSE otherwise.
<code>VersionLabel</code>	String	Textual description the position

		of an individual object with respect to the version series. (For example, version 1.0).
VersionSeriesId	ID	ID of the Version Series for this Object.
IsVersionSeriesCheckedOut	Boolean	TRUE if there currently exists a Private Working Copy for this Version Series. FALSE otherwise
VersionSeriesCheckedOutBy	String	If IsVersionSeriesCheckedOut is TRUE: then an identifier for the user who created the Private Working Copy. "Not set" otherwise.
VersionSeriesCheckedOutId	ID	If IsVersionSeriesCheckedOut is TRUE: The Identifier for the Private Working Copy. "Not set" otherwise.

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 LastModificationDate, etc.)

2.8.6 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.

2.8.7 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.

2.8.8 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.

2.8.9 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.

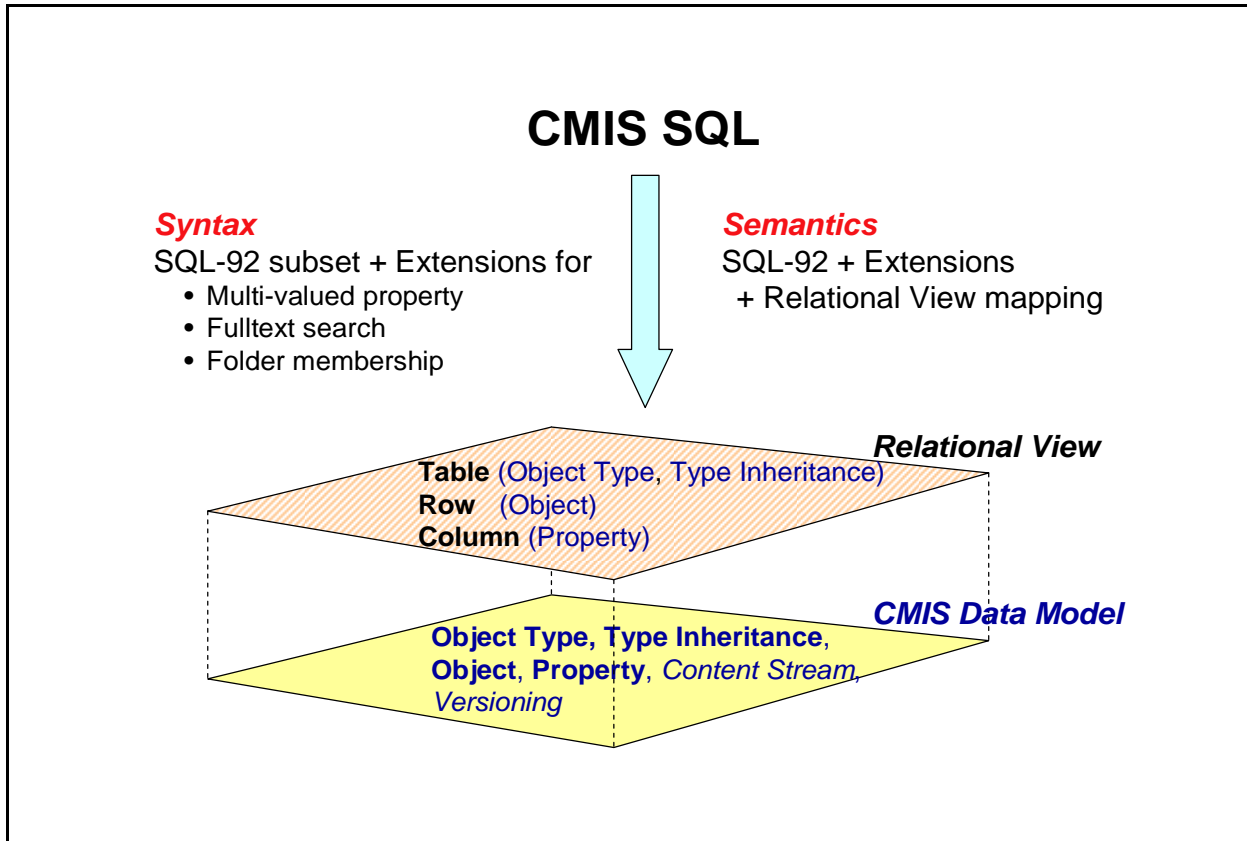
If “PWCSearchable” is TRUE then the Repository SHALL include in the query results ANY Private Working Copy Document Objects that matches the query criteria (subject to other query constraints such as security.)

If “PWCSearchable” is FALSE 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.)

2.9 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.



2.9.1 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 *ObjectTypeQueryName* attribute is used as the table name for the corresponding virtual table, and a property’s *PropertyName* 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.

2.9.1.1 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.

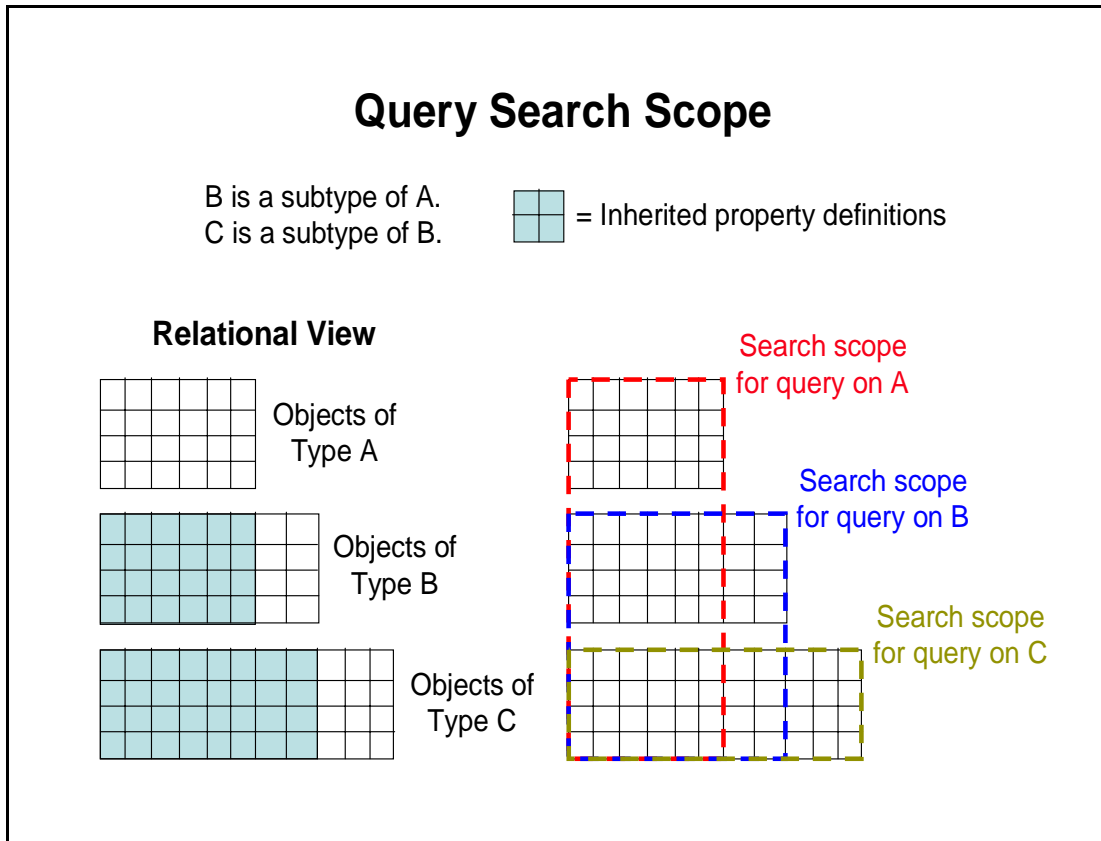
Additionally, an Object-Type definition’s “*includedInSupertypeQuery*” specifies whether objects of that Object-Type SHALL be included in the Virtual Table for any of its ancestor types. If the “*includedInSupertypeQuery*” attribute of the Object-Type is FALSE, then objects of that Object-Type SHALL 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.)



2.9.1.2 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).

2.9.2 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.

2.9.2.1 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 “name” values for properties that are defined as “queryable” in the Object-Type(s) whose Virtual Tables are listed in the FROM clause.

2.9.2.2 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 names of Object-Types whose *queryable* attribute value is TRUE.

2.9.2.2.1 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 “Join”.

- If the Repository’s value for the Join attribute is **NoJoin**, then no JOIN clauses can be used in queries.
- If the Repository’s value for the Join attribute is **InnerOnly**, then only inner JOIN clauses can be used in queries.
- If the Repository’s value for the Join 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.

2.9.2.3 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 “name” values for properties that are defined as “queryable” in the Object-Type(s) whose Virtual Tables are listed in the FROM clause.

2.9.2.3.1 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.

For properties of type "ID", only the "equal" and "not equal" comparison SHALL be allowed in CMIS-SQL queries.

2.9.2.3.2 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.

2.9.2.3.2.1 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.

2.9.2.3.2.2 <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>.

<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.

2.9.2.3.2.3 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.

Query examples:

Example #1:

```
SELECT      *
FROM CAR_REVIEW
```



```
WHERE      ( LOWER(MAKE) = 'buick' ) OR
           ( ANY FEATURES IN ( 'NAVIGATION SYSTEM', 'SATELLITE RADIO', 'MP3' ) )
```

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

Example #2:

```
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 property.)

2.9.2.3.3 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 optional <qualifier> parameter MAY be used to specify the search scope for the CONTAINS() function, and the Repository MAY ignore the <qualifier> parameter.

The qualifier parameter SHALL be a valid string.

<Text Search Expression>

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

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 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.

2.9.2.3.4 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, which will be mapped to a (pseudo) property name through protocol binding.

2.9.2.3.5 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, then the predicate SHALL apply to all Virtual Tables listed in the FROM clause.

<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>.

2.9.2.3.6 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, then the predicate SHALL apply to all Virtual Tables listed in the FROM clause.

<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>.

2.9.2.4 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 “name” values for properties defined as *orderable* in the Object Type(s) whose Virtual Tables are listed in the FROM clause.

2.9.2.5 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.

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.
- “[**name1**], [**name2**], ...”:
 - Note: The [name] tokens in the above expression SHALL be valid “Name” values as defined in the Object-types for the Objects whose properties are being returned.
 - Note: The
- * : All properties SHALL be returned for all objects.

A repository MAY return additional properties for objects in the method output that are not 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, via the following pattern:

Input Parameters:

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

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. 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 applications 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.

Output Parameters:

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

The set of allowable actions for an object is represented as a set of Boolean values, as follows:

Boolean Name	Base Object Types for which it MAY be TRUE	Allowed Action (method)
canDelete	Document/Folder/Relationship/Policy	Can delete this object (deleteAllVersions)
canUpdateProperties	Document/Folder/Relationship/Policy	Can update the properties of this object (updateProperties)
canGetProperties	Document/Folder/Relationship/Policy	Can read the properties of this object (getProperties)
canGetRelationships	Document/Folder/Policy	Can get the relationship in which this object is a source/target (getRelationships)
canGetParents	Document/Folder/Policy	Can get the parent folders of the object. (getObjectParents)
canGetFolderParent	Folder	Can get the parent/ancestor folder(s) of the folder (getFolderParent)
canGetDescendants	Folder	Can get the descendants of the

		folder (getDescendants)
canMove	Document/Folder/Policy	Can move the object (moveObject)
canDeleteVersion	Document	Can delete this Document object in the Version Series (deleteObject)
canDeleteContent	Document	Can delete the content stream for the Document object (deleteContentStream)
canCheckout	Document	Can check out the Document object (checkOut)
canCancelCheckout	Document	Can cancel the check out the Document object (cancelCheckOut)
canCheckin	Document	Can check in the Document object (checkIn)
canSetContent	Document	Can set the content stream for the Document object (setContentStream)
canGetAllVersions	Document	Can get the version series for the Document object (getAllVersions)
canAddToFolder	Document/Policy	Can file the document in a folder (addObjectToFolder)
canRemoveFromFolder	Document/Policy	Can unfile the document in a folder (removeObjectFromFolder)
canViewContent	Document	Can get the content stream for the Document object (getContentStream)
canAddPolicy	Document/Folder	Can apply a policy to the Object (applyPolicy)
canGetAppliedPolicies	Document/Folder	Can get the list of Policies applied to the Object (getAppliedPolicies)
canRemovePolicy	Document/Folder	Can remove a policy from the Object (removePolicy)
canGetChildren	Folder	Can get the children of the folder (getChildren)

canCreateDocument	Folder	Can create a Document Object in the folder (createDocument)
canCreateFolder	Folder	Can create a Folder Object in the folder (createFolder)
canCreateRelationship	Document/Folder	Can create a Relationship in which this Object is a source/target (createRelationship)
canCreatePolicy	Folder	Can create a Policy Object in the folder (createPolicy)
canDeleteTree	Folder	Can delete the Folder and all of its descendant objects (deleteTree)

3.1.4 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.

3.1.5 Table of Exceptions

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

3.1.5.1 General Exceptions

The following exceptions MAY be returned 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.

Name	Cause
invalidArgumentException	One or more of the input parameters to the service method is missing or invalid.
objectNotFoundException	The service call has specified an object that does not exist in the Repository.
operationNotSupportedException	The service method invoked requires an optional capability not supported by the repository.

permissionDeniedException	The caller of the service method does not have sufficient permissions to perform the operation.
runtimeException	Any other cause not expressible by another CMIS exception.

3.1.5.2 Specific Exceptions

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

The table lists the general intent of the exception, as well as a list of the methods which MAY cause the exception to be thrown.

Name	Intent	Methods that MAY throw this exception
constraintViolationException	The operation violates a Repository- or Object-level constraint defined in the CMIS domain model.	<p>Repository Service:</p> <ul style="list-style-type: none"> • getObjectParents <p>Object Service:</p> <ul style="list-style-type: none"> • createDocument • createFolder • createRelationship • createPolicy • updateProperties • moveObject • deleteObject • setContentStream • deleteContentStream <p>Multi-filing Services:</p> <ul style="list-style-type: none"> • addObjectToFolder <p>Versioning Services:</p> <ul style="list-style-type: none"> • checkOut • cancelCheckOut • checkIn <p>Policy Services:</p> <ul style="list-style-type: none"> • applyPolicy • removePolicy <p>Object Services:</p>
contentAlreadyExistsException	The operation attempts to set the	

filterNotValidException

content stream for a Document that already has a content stream without explicitly specifying the “overwrite” parameter.

The [property filter](#) input to the operation is not valid.

- setContentStream

Repository Service:

- getDescendants
- getChildren
- getFolderParent
- getObjectParents
- getCheckedoutDocs

Object Service:

- getProperties

Versioning Services:

- getPropertiesOfLatestVersion
- getAllVersions

Policy Services:

- getAppliedPolicies



folderNotValidException

The operation is attempting to create an object in an “unfiled” state in a repository that does not support the “Unfiling” [optional capability](#).



notInFolderException storageException

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

Object Services:

- createDocument
- createFolder
- createRelationship
- createPolicy
- updateProperties
- moveObject
- setContentStream
- deleteContentStream

Versioning Services:

- checkout
- checkIn

streamNotSupportedException 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.

Object Services:

- createDocument
- getContentStream
- setContentStream

Versioning Services:

- checkIn



typeNotFoundException

updateConflictException

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

Object Services:

- updateProperties
- moveObject
- deleteObject
- deleteTree
- setContentStream
- deleteContentStream

Versioning Services:

- checkout
- cancelCheckOut
- checkIn

versioningException

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.

Object Services:

- updateProperties
- moveObject
- setContentStream
- deleteContentStream

Versioning Services:

- checkOut
- cancelCheckOut
- checkIn

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.
Inputs	None
Outputs	<ul style="list-style-type: none">• A list of repository information, with the following information for each entry:<ul style="list-style-type: none">○ ID repositoryId: The identifier for the Repository.○ String repositoryName: A display name for the Repository.○ URI repositoryURI: A URI which can be used to access the Repository.
Exception Conditions	<ul style="list-style-type: none">• General exceptions.
Notes	

3.2.2 `getRepositoryInfo`

Description	Returns information about the CMIS repository and the optional capabilities it supports.
Inputs	<ul style="list-style-type: none">• ID repositoryId: The identifier for the Repository.
Outputs	<ul style="list-style-type: none">• ID repositoryId: The identifier for the Repository.<ul style="list-style-type: none">○ 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 “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.• Boolean capabilityMultifiling: TRUE if the repository supports the optional “multifiling” capability.• Boolean capabilityUnfiling: TRUE if the repository supports the

Exception Conditions
Notes

optional “unfiling” capability.

- **Boolean capabilityVersionSpecificFiling:** TRUE if the repository supports the optional “version-specific filing” capability.
- **Boolean capabilityPWCUpdateable:** TRUE if the repository supports the optional “PWCUpdateable” capability.
- **Boolean capabilityPWCSearchable:** TRUE if the repository supports the optional “PWCSearchable” capability.
- **Boolean capabilityAllVersionsSearchable:** TRUE if the repository supports the optional “AllVersionsSearchable” capability.
- **Enum capabilityQuery:** Specifies what level of query support the repository provides. (See [optional capabilities](#) section for list of valid values)
- **Enum capabilityFullText:** Specifies what level of full-text support the repository provides. (See [optional capabilities](#) section for list of valid values)
- **Enum capabilityJoin:** Specifies what types of SQL joins the repository can fulfill in query requests. (See [optional capabilities](#) section for list of valid values)
- **String cmisVersionsSupported:** String that indicates what versions of the CMIS specification the repository can support.
- **XML repositorySpecificInformation:** MAY be used by the Repository to return additional XML.
- [General exceptions.](#)



3.2.3 getTypes

Description
Inputs
Outputs

Returns the list of [Object-Types](#) defined for the Repository.

- **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 Object-Types.
- **(optional) Boolean returnPropertyDefinitions:** 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.
- **(optional) Integer maxItems:** See [Paging](#) section.
- **(optional) Integer skipCount:** See [Paging](#) section.
- **<Array> Object-Types:** The list of [Object-Types](#) defined for the

Exception Conditions	Repository.
Notes	<ul style="list-style-type: none"> • Boolean hasMoreItems: See Paging section. • General exceptions. • If this service is called with an invalid value for typeId, the repository SHALL throw invalidArgumentException.


3.2.4 getTypeDefinition

Description	Gets the definition of the specified Object-Type.
Inputs	<ul style="list-style-type: none"> • string repositoryId: The identifier for the Repository. • string typeId: The typeId of an Object-Type specified in the Repository.
Outputs	<ul style="list-style-type: none"> • <Array> typeAttributeCollection: Type attributes of an object type definition. See the “Object Type” section for a complete list of type attributes. • <Array> propertyDefinition: A list of property definitions. See the “Object Type” section for a complete list of property definitions.
Exception Conditions	<ul style="list-style-type: none"> • General exceptions.
Notes	<ul style="list-style-type: none"> •

3.3 Navigation Services

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

3.3.1 getDescendants

Description	Gets the set of descendant objects contained in the specified folder or any of its child-folders.
Inputs	<ul style="list-style-type: none"> • ID repositoryId: The identifier for the Repository. • ID folderId: The identifier for the folder. • (optional) Enum type: An enumeration specifying that the result set SHALL be restricted to objects with the following baseTypes: <ul style="list-style-type: none"> ○ Documents: Only objects whose baseType is “Document” ○ Folders: Only objects whose baseType is “Folder” ○  Policies: Only objects who baseType is “Policy”. ○ Any (default): Return all objects, regardless of baseType. • (optional) Integer depth: The number of levels of depth in the folder hierarchy from which to return results. Valid values are: <ul style="list-style-type: none"> ○ 1 (default): Return only objects that are children of the folder. ○ <Integer value greater than 1>: Return only objects that are children of the folder and descendants up to <value> levels

Outputs
Exception Conditions
Notes

- deep.
 - **-1**: Return ALL descendant objects at all depth levels in the CMIS hierarchy.
- **(optional) String filter**: See “[Property Filters](#)” section.
- **Boolean includeAllowableActions**: See “[Allowable Actions](#)” section.
- **Boolean includeRelationships**:
- **<Array> Objects**: A list of the descendant objects for the specified folder.
- [General exceptions](#).
- **FilterNotValidException**: The Repository SHALL throw this exception if this property filter input parameter is not valid.
- This method does NOT support paging as defined in the paging section.
- The order in which results are returned SHALL be determined by the repository.
- If the Repository supports the optional “VersionSpecificFiling” capability, then the repository SHALL return the document versions filed in the specified folder or its descendant folders.
 - Otherwise, the latest version of the documents SHALL be returned.

3.3.2 getChildren

Description
Inputs
Outputs
Exception Conditions

- Gets the list of child objects contained in the specified folder.
- **ID repositoryId**: The identifier for the Repository.
 - **ID folderId**: The identifier for the folder.
 - **(optional) Enum type**: An enumeration specifying that the result set SHALL be restricted to objects with the following baseTypes:
 - **Documents**: Only objects whose baseType is “Document”
 - **Folders**: Only objects whose baseType is “Folder”
 - **Policies**: Only objects whose baseType is “Policy”.
 - **Any (default)**: Return all objects, regardless of baseType.
 - **(optional) String filter**: See “[Property Filters](#)” section.
 - **Boolean includeAllowableActions**: See “[Allowable Actions](#)” section.
 - **Boolean includeRelationships**:
 - **(optional) Integer maxItems**: See [Paging](#) section.
 - **(optional) Integer skipCount**: See [Paging](#) section.
 - **<Array> Objects**: A list of the child-objects for the specified folder.
 - **Boolean hasMoreItems**: See [Paging](#) section.
 - [General exceptions](#).
 - **FilterNotValidException**: The Repository SHALL throw this exception

Notes	<p>if this property filter input parameter is not valid.</p> <ul style="list-style-type: none"> • If the Repository supports the optional “VersionSpecificFiling” capability, then the repository SHALL return the document versions filed in the specified folder. <ul style="list-style-type: none"> ◦ Otherwise, the latest version of the documents SHALL be returned.
-------	---

3.3.3 getFolderParent

Description	Gets the parent folder object (and optionally all ancestor folder objects) for the specified folder object.
Inputs	<ul style="list-style-type: none"> • ID repositoryId: The identifier for the Repository. • ID folderId: The identifier for the folder. • (optional) String filter: See “Property Filters” section. • Boolean includeAllowableActions: See “Allowable Actions” section. • Boolean includeRelationships: • Boolean returnToRoot: If TRUE, then the repository SHALL return all folder objects that are ancestors of the specified folder. If FALSE, the repository SHALL return only the parent folder of the specified folder.
Outputs	<ul style="list-style-type: none"> • <Array> Objects: A list of the parent or ancestor folders of the specified folder.
Exception Conditions	<ul style="list-style-type: none"> • General exceptions. • FilterNotValidException: The Repository SHALL throw this exception if this property filter input parameter is not valid.
Notes	<ul style="list-style-type: none"> • If returnToRoot is set to TRUE, then the result list SHALL be ordered in reverse order of ancestry (i.e. where the parent folder of the specified folder is returned first, its parent folder returned second, etc.) • 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 getObjectParents

Description	Gets the parent folder(s) for the specified non-folder, fileable object.
Inputs	<ul style="list-style-type: none"> • ID repositoryId: The identifier for the Repository. • ID objectId: The identifier for the object. • (optional) String filter: See “Property Filters” section. • Boolean includeAllowableActions: See “Allowable Actions” section. • Boolean includeRelationships:
Outputs	<ul style="list-style-type: none"> • <Array> Objects: A list of the parent folder(s) of the specified objects.
Exception Conditions	<ul style="list-style-type: none"> • General exceptions.

Notes

- **ConstraintViolationException:** The Repository SHALL throw this exception if this method is invoked on an object whose Object-Type Definition specifies that it is not fileable.
- **FilterNotValidException:** The Repository SHALL throw this exception if this property filter input parameter is not valid.

3.3.5 getCheckedoutDocs

Description

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

Inputs

- **ID repositoryId:** The identifier for the Repository.
- **(optional) 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.
- **(optional) String filter:** See “[Property Filters](#)” section.
- **Boolean includeAllowableActions:** See “[Allowable Actions](#)” section.



- **Boolean includeRelationships:**
- **(optional) Integer maxItems:** See [Paging](#) section.
- **(optional) Integer skipCount:** See [Paging](#) section.
- **<Array> Objects:** A list of the parent folder(s) of the specified objects.
- **Boolean hasMoreItems:** See [Paging](#) section.
- [General exceptions](#).
- **FilterNotValidException:** The Repository SHALL throw this exception if this property filter input parameter is not valid.

Outputs

Exception Conditions

Notes

•

3.4 Object Services

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

3.4.1 createDocument

Description

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

Inputs

- **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

Outputs

Exception Conditions

Notes

SHALL be the parent folder for the newly-created Document Object.

- This parameter MUST be specified if the Repository does NOT support the optional “unfiling” capability.
- **(Optional) <contentStream> contentStream:** The Content Stream that SHALL be stored for the newly-created Document Object.
- **(Optional) Enum versioningState:** An enumeration specifying what the versioning state of the newly-created object SHALL be. Valid values are:
 - **checkedout:** The document SHALL be created in the checked-out state.
 - **major:** The document SHALL be created as a major version
 - **minor (default):** The document SHALL be created as a minor version.
- **ID objectId:** The ID of the newly-created document.
- **General exceptions.**
- **ConstraintViolationException:** 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 AllowedChildObjectTypes 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.
- **storageException:** See “[specific exceptions](#)” section.
- **streamNotSupportedException:** 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

Inputs

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

- **ID repositoryId:** The identifier for the Repository.
- **ID typeId:** The identifier for the Object-Type of the Folder object being created.

Outputs	<ul style="list-style-type: none"> • <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. • ID objectId: The ID of the newly-created folder.
Exception Conditions	<ul style="list-style-type: none"> • General exceptions. • ConstraintViolationException: The Repository SHALL throw this exception if ANY of the following conditions are met: <ul style="list-style-type: none"> ○ The typeId is not an Object-Type whose baseType is “Folder”. ○ 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 AllowedChildObjectTypes of the parent-folder specified by folderId. • storageException: See “specific exceptions” section.
Notes	

3.4.3 createRelationship

Description	Creates a relationship object of the specified type
Inputs	<ul style="list-style-type: none"> • 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 target object for the newly-created Relationship.
Outputs	<ul style="list-style-type: none"> • ID objectId: The ID of the newly-created relationship.
Exception Conditions	<ul style="list-style-type: none"> • General exceptions. • ConstraintViolationException: The Repository SHALL throw this exception if ANY of the following conditions are met: <ul style="list-style-type: none"> ○ 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

Notes

“allowedTargetTypes” specified by the Object-Type definition specified by typeId.

- **storageException:** See “[specific exceptions](#)” section.

3.4.4 createPolicy

Description

Creates a policy object of the specified type

Inputs

- **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.

Outputs

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

Exception Conditions

- **General exceptions.**
- **ConstraintViolationException:** 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 AllowedChildObjectTypes of the parent-folder specified by folderId.

Notes

- **storageException:** See “[specific exceptions](#)” section.

3.4.5 getAllowableActions

Description

Gets the list of allowable actions for an Object (see “[Allowable Actions](#)” section).

Inputs

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

Outputs

- **<Array> AllowableActions:** see “[Allowable Actions](#)” section

Exception Conditions

- **General exceptions.**

Notes

3.4.6 getProperties

Description	Gets a subset of the properties for an Object.
Inputs	<ul style="list-style-type: none">• ID repositoryId: The identifier for the Repository.• ID objectId: The identifier for the object• (optional) Enum returnVersion:• (optional) String Filter: See “Property Filters” section.• Boolean includeAllowableActions: See “Allowable Actions” section.• Boolean includeRelationships:• <Array> Properties: The list of properties for the object.
Outputs	
Exception Conditions	<ul style="list-style-type: none">• General exceptions.• FilterNotValidException: The Repository SHALL throw this exception if this property filter input parameter is not valid.
Notes	

3.4.7 getContentStream

Description	
Inputs	<ul style="list-style-type: none">• ID repositoryId: The identifier for the Repository.• ID objectId: The identifier for the object• <ContentStream> ContentStream: The content stream for the object..
Outputs	
Exception Conditions	<ul style="list-style-type: none">• General exceptions.• streamNotSupportedException: The Repository SHALL throw this exception if the Object-Type definition specified by the objectId parameter’s “contentStreamAllowed” attribute is set to “not allowed”.
Notes	<ul style="list-style-type: none">• Each CMIS protocol binding SHALL provide a way for fetching a sub-range within a content stream, in a manner appropriate to that protocol.

3.4.8 updateProperties

Description	Updates properties of the specified object.
Inputs	<ul style="list-style-type: none">• ID repositoryId: The identifier for the Repository.• ID objectId: The identifier of the object to be updated.• (optional) String changeToken: See “Change Tokens” section.• <Array> properties: The updated property values that SHALL be

Outputs	
Exception Conditions	<ul style="list-style-type: none"> applied to the Object. ID objectId: The ID of the updated object. General exceptions. ConstraintViolationException: The Repository SHALL throw this exception if ANY of the following conditions are met: <ul style="list-style-type: none"> The value of any of the properties violates the min/max/required/length constraints specified in the property definition in the Object-Type. storageException: See “specific exceptions” section. updateConflictException: See “specific exceptions” section. versioningException: The Repository SHALL throw this exception if ANY of the following conditions are met: <ul style="list-style-type: none"> 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 <i>updateability</i> Whencheckedout. Additionally, the repository MAY throw this exception if the object is a non-current Document Version.
Notes	<ul style="list-style-type: none"> 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.

3.4.9 moveObject

Description	Moves the specified file-able object from one folder to another.
Inputs	<ul style="list-style-type: none"> ID repositoryId: The identifier for the Repository. ID objectId: The identifier of the object to be moved. (optional) String changeToken: See “Change Tokens” section. ID targetFolderId: The folder into which the object is to be moved. (optional) ID sourceFolderId: The folder from which the object is to be moved. <ul style="list-style-type: none"> This parameter SHALL be specified if the Repository supports the optional “unfiling” capability.
Outputs	None.
Exception Conditions	<ul style="list-style-type: none"> General exceptions. ConstraintViolationException: The Repository SHALL throw this exception if ANY of the following conditions are met: <ul style="list-style-type: none"> The typeId is NOT in the list of AllowedChildObjectTypes of the parent-folder specified by targetFolderId.

Notes	<ul style="list-style-type: none"> • storageException: See “specific exceptions” section. • updateConflictException: See “specific exceptions” section. • versioningException: The repository MAY throw this exception if the object is a non-current Document Version. <p>None.</p>
-------	--

3.4.10 deleteObject

Description	Deletes the specified object.
Inputs	<ul style="list-style-type: none"> • ID repositoryId: The identifier for the Repository. • ID objectId: The identifier of the object to be deleted. • (optional) String changeToken: See “Change Tokens” section.
Outputs	None.
Exception Conditions	<ul style="list-style-type: none"> • General exceptions. • ConstraintViolationException: The Repository SHALL throw this exception if ANY of the following conditions are met: <ul style="list-style-type: none"> ○ The method is invoked on a Folder object that contains one or more objects. • updateConflictException: See “specific exceptions” section.
Notes	<ul style="list-style-type: none"> • 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.

3.4.11 deleteTree

Description	Deletes the specified folder object and all of its child- and descendant-objects.
Inputs	<ul style="list-style-type: none"> • ID repositoryId: The identifier for the Repository. • ID folderId: The identifier of the object to be deleted. • (optional) String changeToken: See “Change Tokens” section. • (optional) Enum unfileNonfolderObject: An enumeration specifying how the repository SHALL process file-able child- or descendant-objects. Valid values are: <ul style="list-style-type: none"> ○ unfile: Unfile all fileable objects. ○ 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. ○ delete (default): Delete all fileable objects. • (optional) boolean continueOnFailure: If TRUE, then the repository SHALL continue attempting to perform this operation even if deletion of

Outputs	<ul style="list-style-type: none"> • a child- or descendant-object in the specified folder cannot be deleted. <ul style="list-style-type: none"> ◦ If FALSE, then the repository SHALL abort this method when it fails to delete a single child- or descendant-object. • <Array> ID failedToDelete: A list of identifiers of objects in the folder tree that were not deleted.
Exception Conditions	<ul style="list-style-type: none"> • General exceptions. • updateConflictException: See “specific exceptions” section.
Notes	<ul style="list-style-type: none"> • 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.

3.4.12 setContentStream

Description	Sets the content stream for the specified Document object.
Inputs	<ul style="list-style-type: none"> • ID repositoryId: The identifier for the Repository. • ID documentID: The identifier for the Document object. • (optional) String changeToken: See “Change Tokens” section. • Boolean overwriteFlag: If TRUE, then the Repository SHALL replace the existing content stream for the object (if any) with the input contentStream. <ul style="list-style-type: none"> ◦ If FALSE, then the Repository SHALL only set the input contentStream for the object if the object currently does not have a content-stream. • <contentStream> contentStream: The Content Stream that SHALL be stored for the newly-created Document Object.
Outputs	<ul style="list-style-type: none"> • ID documentId: The ID of the document.
Exception Conditions	<ul style="list-style-type: none"> • General exceptions. • ConstraintViolationException: The Repository SHALL throw this exception if ANY of the following conditions are met: <ul style="list-style-type: none"> ◦ The Object’s Object-Type definition “contentStreamAllowed” attribute is set to “<i>notAllowed</i>”. • contentAlreadyExistsException: The Repository SHALL throw this exception if the input parameter overwriteFlag is FALSE and the Object already has a content-stream. • storageException: See “specific exceptions” section. • streamNotSupportedException: The Repository SHALL throw this exception if the Object-Type definition specified by the typeId parameter’s “contentStreamAllowed” attribute is set to “not allowed”. • updateConflictException: See “specific exceptions” section. • versioningException: The repository MAY throw this exception if the object is a non-current Document Version.

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.

3.4.13 deleteContentStream

Description	Deletes the content stream for the specified Document object.
Inputs	<ul style="list-style-type: none">• ID repositoryId: The identifier for the Repository.• ID documentId: The identifier for the Document object.• (optional) String changeToken: See “Change Tokens” section.• ID documentId: The ID of the Document object.
Outputs	
Exception Conditions	<ul style="list-style-type: none">• General exceptions.• ConstraintViolationException: The Repository SHALL throw this exception if ANY of the following conditions are met:<ul style="list-style-type: none">○ The Object's Object-Type definition “contentStreamAllowed” attribute is set to “<i>required</i>”.• storageException: See “specific exceptions” section.• updateConflictException: See “specific exceptions” section.• versioningException: The repository MAY throw this exception if the object is a non-current Document Version.
Notes	<ul style="list-style-type: none">• 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.

3.5 Multi-filing Services

The Multi-filing services (*addObjectToFolder*, *removeObjectFromFolder*) are supported only if the repository supports the multifiling or unfileing [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.
Inputs	<ul style="list-style-type: none">• 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.
Outputs	None.
Exception Conditions	<ul style="list-style-type: none">• General exceptions.

Notes	<ul style="list-style-type: none"> • ConstraintViolationException: The Repository SHALL throw this exception if ANY of the following conditions are met: <ul style="list-style-type: none"> ○ The typeId is NOT in the list of AllowedChildObjectTypes of the parent-folder specified by folderId.
	None.

3.5.2 removeObjectFromFolder

Description	Removes an existing fileable non-folder object from a folder.
Inputs	<ul style="list-style-type: none"> • 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. <ul style="list-style-type: none"> ○ If no value is specified, then the Repository SHALL remove the object from all folders in which it is currently filed.
Outputs	None.
Exception Conditions	<ul style="list-style-type: none"> • General exceptions.
Notes	None.

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.
Inputs	<ul style="list-style-type: none"> • ID repositoryId: The identifier for the Repository. • String statement: CMIS-SQL query to be executed. (See “Query” section.) • Boolean SearchAllVersions: • (optional) Integer maxItems: See Paging section. • (optional) Integer skipCount: See Paging section. • Boolean includeAllowableActions: See “Allowable Actions” section.
Outputs	<ul style="list-style-type: none"> • <Array> Object QueryResults: The set of results for the query. (See “Query” section.) • Boolean hasMoreItems: See Paging section.
Exception Conditions	<ul style="list-style-type: none"> • General exceptions.

Notes

None.

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.
Inputs	<ul style="list-style-type: none">• ID repositoryId: The identifier for the Repository.• ID documentId: The identifier of the object.• (optional) String changeToken: See “Change Tokens” section.
Outputs	<ul style="list-style-type: none">• 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.<ul style="list-style-type: none">◦ FALSE if the content-stream of the Private Working Copy is “not set”.
Exception Conditions	<ul style="list-style-type: none">• General exceptions.• ConstraintViolationException: The Repository SHALL throw this exception if ANY of the following conditions are met:<ul style="list-style-type: none">◦ The Document’s Object-Type definition’s <i>versionable</i> attribute is FALSE.• storageException: See “specific exceptions” section.• updateConflictException: See “specific exceptions” section.• versioningException: The repository MAY throw this exception if the object is a non-current Document Version.
Notes	None.

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.
Inputs	<ul style="list-style-type: none">• ID repositoryId: The identifier for the Repository.• ID documentId: The identifier of the object.• (optional) String changeToken: See “Change Tokens” section.
Outputs	None.

Exception Conditions	<ul style="list-style-type: none"> • General exceptions. • ConstraintViolationException: The Repository SHALL throw this exception if ANY of the following conditions are met: <ul style="list-style-type: none"> ◦ The Document's Object-Type definition's <i>versionable</i> attribute is FALSE. • updateConflictException: See "specific exceptions" section. • versioningException: The repository MAY throw this exception if the object is a non-current Document Version.
Notes	None.

3.7.3 checkIn

Description	Checks-in the Private Working Copy object.
Inputs	<ul style="list-style-type: none"> • ID repositoryId: The identifier for the Repository. • ID documentId: The identifier of the object. • (optional) String changeToken: See "Change Tokens" section. • Boolean Major: TRUE if the checked-in Document Object SHALL be a major version. <ul style="list-style-type: none"> ◦ FALSE if the checked-in Document Object SHALL NOT be a major version. • (Optional) <Array> properties: The property values that SHALL be applied to the checked-in Document Object. • (Optional) <contentStream> contentStream: The Content Stream that SHALL be stored for the checked-in Document Object. • (Optional) String checkinComment: See "Versioning Properties on Document Objects" section.
Outputs	<ul style="list-style-type: none"> • ID documentId: The ID of the checked-in document.
Exception Conditions	<ul style="list-style-type: none"> • General exceptions. • ConstraintViolationException: The Repository SHALL throw this exception if ANY of the following conditions are met: <ul style="list-style-type: none"> ◦ The Document's Object-Type definition's <i>versionable</i> attribute is FALSE. • storageException: See "specific exceptions" section. • streamNotSupportedException: 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. • updateConflictException: See "specific exceptions" section. • versioningException: The repository MAY throw this exception if the object is a non-current Document Version.

Notes

For repositories that do NOT support the [optional “PWCUpdateable” capability](#), the *properties* and *contentStream* input parameters to this method is the ONLY way to set those values for a Private Working Copy object.

3.7.4 getPropertiesOfLatestVersion

Description

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

Inputs

- **ID repositoryId:** The identifier for the Repository.
- **ID versionSeriesId:** The identifier for the Version Series.
- **Boolean MajorVersion:** If TRUE, then the Repository SHALL return the properties for the latest major version object in the Version Series.
 - If FALSE, the Repository SHALL return the properties for the latest (major or non-major) version object in the Version Series.
- **(optional) String Filter:** See “[Property Filters](#)” section.
- **<Array> Properties:** The list of properties for the object.

Outputs

Exception Conditions

- [General exceptions](#).
- **FilterNotValidException:** The Repository SHALL throw this exception if this property filter input parameter is not valid.

Notes

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

3.7.5 getAllVersions

Description

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

Inputs

- **ID repositoryId:** The identifier for the Repository.
- **ID versionSeriesId:** The identifier for the Version Series.
- **(optional) String filter:** See “[Property Filters](#)” section.
- **Boolean includeAllowableActions:** See “[Allowable Actions](#)” section.
- **Boolean includeRelationships:**

Outputs

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

Exception Conditions

- [General exceptions](#).
- **FilterNotValidException:** The Repository SHALL throw this exception if this property filter input parameter is not valid.

Notes

- If the input parameter MajorVersion is TRUE and the Version Series

contains no major versions, then the Repository SHALL throw `objectNotFoundException`.

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

3.7.6 deleteAllVersions

Description	Deletes all Document Objects in the specified Version Series, including the Private Working Copy.
Inputs	<ul style="list-style-type: none">• ID repositoryId: The identifier for the Repository.• ID versionSeriesId: The identifier of the Version Series to be deleted.
Outputs	None.
Exception Conditions	<ul style="list-style-type: none">• General exceptions.
Notes	None.

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 the subset of relationships associated with an independent object.
Inputs	<ul style="list-style-type: none">• ID repositoryId: The identifier for the Repository.• ID objectId: The identifier of the object.• Enum direction: 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:<ul style="list-style-type: none">○ source: 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.○ both: The Repository SHALL return relationship objects where the specified object is either the source or the target object.• (optional) 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.)<ul style="list-style-type: none">○ If not specified, then the repository SHALL return Relationship objects of all types.• (optional): Boolean includeSubRelationshipTypes: If TRUE, then the Repository SHALL return all relationships whose Object-Types are descendant-types of <i>typeId</i>.

Outputs	<ul style="list-style-type: none"> ○ If FALSE (default), then the Repository SHALL only return relationships whose Object-Type is <i>typed</i>.
Exception Conditions	<ul style="list-style-type: none"> • (optional) String Filter: See “Property Filters” section. • Boolean includeAllowableActions: See “Allowable Actions” section. • (optional) Integer maxItems: See Paging section. • (optional) Integer skipCount: See Paging section. • <Array> Objects: A list of the relationship objects. • Boolean hasMoreItems: See Paging section.
Notes	None.

3.9 Policy Services

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

3.9.1 applyPolicy

Description	Applies a specified policy to an object.
Inputs	<ul style="list-style-type: none"> • ID repositoryId: The identifier for the Repository. • ID policyId: The identifier for the Policy to be applied. • ID objectId: The identifier of the object.
Outputs	None.
Exception Conditions	<ul style="list-style-type: none"> • General exceptions. • ConstraintViolationException: The Repository SHALL throw this exception if ANY of the following conditions are met: <ul style="list-style-type: none"> ○ The specified object’s Object-Type definition’s attribute for <i>controllable</i> is FALSE.
Notes	None.

3.9.2 removePolicy

Description	Removes a specified policy from an object.
-------------	--

Inputs	<ul style="list-style-type: none"> • ID repositoryId: The identifier for the Repository. • ID policyId: The identifier for the Policy to be removed. • ID objectId: The identifier of the object.
Outputs	None.
Exception Conditions	<ul style="list-style-type: none"> • General exceptions. • ConstraintViolationException: The Repository SHALL throw this exception if ANY of the following conditions are met: <ul style="list-style-type: none"> ○ The specified object's Object-Type definition's attribute for <i>controllable</i> is FALSE.
Notes	None.

3.9.3 getAppliedPolicies

Description	Gets the list of policies currently applied to the specified object.
Inputs	<ul style="list-style-type: none"> • ID repositoryId: The identifier for the Repository. • ID objectId: The identifier of the object. • (optional) String filter: See "Property Filters" section.
Outputs	<Array> Objects: A list of Policy Objects.
Exception Conditions	<ul style="list-style-type: none"> • General exceptions. • FilterNotValidException: The Repository SHALL throw this exception if this property filter input parameter is not valid.
Notes	None.

 **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
0.6	3/8/2009	Ethan Gur-esh	<ul style="list-style-type: none">• Re-wrote spec text into normative style.• Re-factored spec sections for clarity.• Fixed several spec bugs tracked in JIRA.