Content Management Interoperability Services (CMIS) Version 1.1

Committee Specification 01

12 November 2012

Specification URIs

This version:

http://docs.oasis-open.org/cmis/CMIS/v1.1/cs01/CMIS-v1.1-cs01.pdf (Authoritative)
http://docs.oasis-open.org/cmis/CMIS/v1.1/cs01/CMIS-v1.1-cs01.html

Previous version:

http://docs.oasis-open.org/cmis/CMIS/v1.1/csprd01/CMIS-v1.1-csprd01.pdf (Authoritative)
http://docs.oasis-open.org/cmis/CMIS/v1.1/csprd01/CMIS-v1.1-csprd01.html

Latest version:

http://docs.oasis-open.org/cmis/CMIS/v1.1/CMIS-v1.1.pdf (Authoritative)
http://docs.oasis-open.org/cmis/CMIS/v1.1/CMIS-v1.1.html

Technical Committee:

OASIS Content Management Interoperability Services (CMIS) TC

Chair:

David Choy (david.choy500@gmail.com), Individual

Editors:

Florian Müller (florian.mueller02@sap.com), SAP
Ryan McVeigh (rmcveigh@ziaconsulting.com), Zia Consulting
Jens Hübel (j.huebel@sap.com), SAP

Additional artifacts:

This prose specification is one component of a Work Product which also includes:

- XML schemas, WSDL and Orderly schema:
  http://docs.oasis-open.org/cmis/CMIS/v1.1/cs01/schema/
- XML and JSON examples:
  http://docs.oasis-open.org/cmis/CMIS/v1.1/cs01/examples/
- TeX source files for this prose document:
  http://docs.oasis-open.org/cmis/CMIS/v1.1/cs01/tex/

Related work:

This specification supersedes:

- Content Management Interoperability Services (CMIS) Version 1.0. OASIS Standard:
  http://docs.oasis-open.org/cmis/CMIS/v1.0/cmis-spec-v1.0.html
- Content Management Interoperability Services (CMIS) Version 1.0 Errata 01:
  http://docs.oasis-open.org/cmis/CMIS/v1.0/errata-01/cmis-spec-v1.0-errata-01.html

Declared XML namespaces:

- http://docs.oasis-open.org/ns/cmis/core/200908/
- http://docs.oasis-open.org/ns/cmis/restatom/200908/
- http://docs.oasis-open.org/ns/cmis/messaging/200908/
- http://docs.oasis-open.org/ns/cmis/ws/200908/
- http://docs.oasis-open.org/ns/cmis/link/200908/

Abstract:

The Content Management Interoperability Services (CMIS) standard defines a domain model and Web Services, Restful AtomPub and browser (JSON) bindings 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 OASIS Content Management Interoperability Services (CMIS) TC on the above date. The level of approval is also listed above. Check the "Latest 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).

Citation format:

When referencing this specification the following citation format should be used:

[CMIS-v1.1]

Content Management Interoperability Services (CMIS) Version 1.1. 12 November 2012. OASIS Committee Specification 01. http://docs.oasis-open.org/cmis/CMIS/v1.1/cs01/CMIS-v1.1-cs01.html.

Notices

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 name "OASIS" is a trademark 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/policies-guidelines/trademark for above guidance.

Table of Contents

1 Introduction **12**
- 1.1 Terminology . . . 12
- 1.2 Normative References . . . 12
- 1.3 Non-Normative References . . . 13
- 1.4 Examples . . . 13
- 1.5 Changes for the CMIS 1.1 specification . . . 14
  - 1.5.1 Type Mutability . . . 14
  - 1.5.2 Repository Features . . . 14
  - 1.5.3 Secondary object types . . . 14
  - 1.5.4 Retention and Hold Support . . . 14
  - 1.5.5 Browser Binding . . . 14
  - 1.5.6 New cmis:item Object Type . . . 14
  - 1.5.7 Service bulkUpdateProperties . . . 15
  - 1.5.8 Append to a content stream . . . 15

2 Domain Model **16**
- 2.1 Data Model . . . 16
  - 2.1.1 Repository . . . 16
    - 2.1.1.1 Optional Capabilities . . . 16
    - 2.1.1.2 Implementation Information . . . 19
    - 2.1.1.3 Repository Features . . . 19
  - 2.1.2 Object . . . 20
    - 2.1.2.1 Property . . . 21
  - 2.1.3 Object-Type . . . 23
    - 2.1.3.1 Object-Type Hierarchy and Inheritance . . . 23
    - 2.1.3.2 Object-Type Attributes . . . 24
    - 2.1.3.3 Object-Type Property Definitions . . . 25
  - 2.1.4 Document Object . . . 31
    - 2.1.4.1 Content Stream . . . 31
    - 2.1.4.2 Renditions . . . 31
    - 2.1.4.3 Document Object-Type Definition . . . 33
  - 2.1.5 Folder Object . . . 47
    - 2.1.5.1 File-able Objects . . . 47
    - 2.1.5.2 Folder Hierarchy . . . 48
    - 2.1.5.3 Paths . . . 49
    - 2.1.5.4 Folder Object-Type Definition . . . 50
  - 2.1.6 Relationship Object . . . 57
    - 2.1.6.1 Relationship Object-Type Definition . . . 57
  - 2.1.7 Policy Object . . . 65
    - 2.1.7.1 Policy Object-Type Definition . . . 65
  - 2.1.8 Item Object . . . 72
    - 2.1.8.1 Item Object-Type Definition . . . 72
  - 2.1.9 Secondary Object-Types . . . 78
    - 2.1.9.1 Secondary Type Application . . . 78
    - 2.1.9.2 Secondary Object-Type Definition . . . 78
  - 2.1.10 Object-Type Creation, Modification and Deletion . . . 81
    - 2.1.10.1 General Constraints on Metadata Changes . . . 81
  - 2.1.11 Object-Type Summary . . . 82

2.1.12 Access Control . . . 85
2.1.12.1 ACL, ACE, Principal, and Permission . . . 85
2.1.12.2 CMIS Permissions . . . 85
2.1.12.3 ACL Capabilities . . . 85
2.1.13 Versioning . . . 95
2.1.13.1 Version Series . . . 95
2.1.13.2 Latest Version . . . 95
2.1.13.3 Behavioral constraints on non-Latest Versions . . . 95
2.1.13.4 Major Versions . . . 95
2.1.13.5 Services that modify Version Series . . . 96
2.1.13.6 Versioning Properties on Document Objects . . . 97
2.1.13.7 Document Creation and Initial Versioning State . . . 98
2.1.13.8 Version Specific/Independent membership in Folders . . . 99
2.1.13.9 Version Specific/Independent membership in Relationships . . . 99
2.1.13.10 Versioning visibility in Query Services . . . 99
2.1.14 Query . . . 101
2.1.14.1 Relational View Projection of the CMIS Data Model . . . 101
2.1.14.2 Query Language Definition . . . 103
2.1.14.3 Escaping . . . 112
2.1.15 Change Log . . . 113
2.1.15.1 Completeness of the Change Log . . . 113
2.1.15.2 Change Log Token . . . 113
2.1.15.3 "Latest Change Token" repository information . . . 113
2.1.15.4 Change Event . . . 113
2.1.16 Retentions and Holds . . . 115
2.1.16.1 Repository Managed Retentions . . . 115
2.1.16.2 Client Managed Retentions . . . 118
2.1.16.3 Holds . . . 123
2.2 Services . . . 126
2.2.1 Common Service Elements . . . 126
2.2.1.1 Paging . . . 126
2.2.1.2 Retrieving additional information on objects in CMIS service calls . . . 126
2.2.1.3 Change Tokens . . . 131
2.2.1.4 Exceptions . . . 132
2.2.1.5 ACLs . . . 133
2.2.2 Repository Services . . . 134
2.2.2.1 getRepositories . . . 135
2.2.2.2 getRepositoryInfo . . . 136
2.2.2.3 getTypeChildren . . . 138
2.2.2.4 getTypeDescendants . . . 139
2.2.2.5 getTypeDefinition . . . 140
2.2.2.6 createType . . . 141
2.2.2.7 updateType . . . 142
2.2.2.8 deleteType . . . 143
2.2.3 Navigation Services . . . 144
2.2.3.1 getChildren . . . 145
2.2.3.2 getDescendants . . . 146
2.2.3.3 getFolderTree . . . 148
2.2.3.4 getFolderParent . . . 150
2.2.3.5 getObjectParents . . . 151
2.2.3.6 getCheckedOutDocs . . . 152
2.2.4 Object Services . . . 153
2.2.4.1 createDocument . . . 154
2.2.4.2 createDocumentFromSource . . . 156
2.2.4.3 createFolder . . . 158
2.2.4.4 createRelationship . . . 159
2.2.4.5 createPolicy . . . 160

2.2.4.6 createItem . . . 161
2.2.4.7 getAllowableActions . . . 162
2.2.4.8 getObject . . . 163
2.2.4.9 getProperties . . . 164
2.2.4.10 getObjectByPath . . . 165
2.2.4.11 getContentStream . . . 166
2.2.4.12 getRenditions . . . 167
2.2.4.13 updateProperties . . . 168
2.2.4.14 bulkUpdateProperties . . . 169
2.2.4.15 moveObject . . . 170
2.2.4.16 deleteObject . . . 171
2.2.4.17 deleteTree . . . 172
2.2.4.18 setContentStream . . . 173
2.2.4.19 appendContentStream . . . 174
2.2.4.20 deleteContentStream . . . 175
2.2.5 Multi-filing Services . . . 176
2.2.5.1 addObjectToFolder . . . 177
2.2.5.2 removeObjectFromFolder . . . 178
2.2.6 Discovery Services . . . 179
2.2.6.1 query . . . 180
2.2.6.2 getContentChanges . . . 182
2.2.7 Versioning Services . . . 184
2.2.7.1 checkOut . . . 185
2.2.7.2 cancelCheckOut . . . 186
2.2.7.3 checkIn . . . 187
2.2.7.4 getObjectOfLatestVersion . . . 188
2.2.7.5 getPropertiesOfLatestVersion . . . 189
2.2.7.6 getAllVersions . . . 190
2.2.8 Relationship Services . . . 191
2.2.8.1 getObjectRelationships . . . 192
2.2.9 Policy Services . . . 194
2.2.9.1 applyPolicy . . . 195
2.2.9.2 removePolicy . . . 196
2.2.9.3 getAppliedPolicies . . . 197
2.2.10 ACL Services . . . 198
2.2.10.1 applyACL . . . 199
2.2.10.2 getACL . . . 200

3 AtomPub Binding 201
3.1 Overview . . . 201
3.1.1 Namespaces . . . 201
3.1.2 Authentication . . . 201
3.1.3 Response Formats . . . 201
3.1.4 Optional Arguments . . . 202
3.1.5 Errors and Exceptions . . . 202
3.1.6 Renditions . . . 202
3.1.7 Content Streams . . . 202
3.1.8 Paging of Feeds . . . 202
3.1.9 Services not Exposed . . . 203
3.1.9.1 removePolicy . . . 203
3.2 HTTP . . . 203
3.2.1 HTTP Range . . . 203
3.2.2 HTTP OPTIONS Method . . . 203
3.2.3 HTTP Status Codes . . . 204
3.2.3.1 General CMIS Exceptions . . . 204
3.2.3.2 Notable HTTP Status Codes . . . 204
3.3 Media Types . . . 204

3.3.1 CMIS Atom . . . 205
3.3.2 CMIS Query . . . 205
3.3.3 CMIS Allowable Actions . . . 205
3.3.4 CMIS Tree . . . 206
3.3.5 CMIS ACL . . . 206
3.4 Atom Extensions for CMIS . . . 206
3.4.1 Atom Element Extensions . . . 206
3.4.1.1 AtomPub Workspace . . . 206
3.4.1.2 Atom Feed . . . 207
3.4.1.3 Atom Entry . . . 207
3.4.2 Attributes . . . 208
3.4.2.1 cmisra:id . . . 208
3.4.2.2 cmisra:renditionKind . . . 208
3.4.3 CMIS Link Relations . . . 209
3.4.3.1 Existing Link Relations . . . 209
3.4.3.2 Hierarchy Navigation Internet Draft Link Relations . . . 211
3.4.3.3 Versioning Internet Draft Link Relations . . . 211
3.4.3.4 CMIS Specific Link Relations . . . 211
3.5 Atom Resources . . . 213
3.5.1 Feeds . . . 213
3.5.2 Entries . . . 214
3.5.2.1 Hierarchical Atom Entries . . . 215
3.6 Resources Overview . . . 216
3.7 AtomPub Service Document . . . 218
3.7.1 HTTP GET . . . 218
3.7.1.1 URI Templates . . . 219
3.8 Service Collections . . . 223
3.8.1 Root Folder Collection . . . 223
3.8.2 Query Collection . . . 223
3.8.2.1 HTTP POST . . . 223
3.8.3 Checked Out Collection . . . 224
3.8.3.1 HTTP GET . . . 224
3.8.3.2 HTTP POST . . . 225
3.8.4 Unfiled Collection . . . 225
3.8.4.1 HTTP POST . . . 225
3.8.5 Type Children Collection . . . 226
3.8.5.1 HTTP GET . . . 226
3.8.5.2 HTTP POST . . . 227
3.8.6 Bulk Update Collection . . . 228
3.8.6.1 HTTP POST . . . 228
3.9 Collections . . . 229
3.9.1 Relationships Collection . . . 229
3.9.1.1 HTTP GET . . . 229
3.9.1.2 HTTP POST . . . 230
3.9.2 Folder Children Collection . . . 230
3.9.2.1 HTTP GET . . . 230
3.9.2.2 HTTP POST . . . 231
3.9.2.3 HTTP DELETE . . . 233
3.9.3 Policies Collection . . . 233
3.9.3.1 HTTP GET . . . 233
3.9.3.2 HTTP POST . . . 234
3.9.3.3 HTTP DELETE . . . 234
3.10 Feeds . . . 234
3.10.1 Object Parents Feed . . . 234
3.10.1.1 HTTP GET . . . 235
3.10.2 Changes Feed . . . 235
3.10.2.1 HTTP GET . . . 235

3.10.3 Folder Descendants Feed . . . 236
3.10.3.1 HTTP GET . . . 236
3.10.3.2 HTTP DELETE . . . 237
3.10.4 Folder Tree Feed . . . 238
3.10.4.1 HTTP GET . . . 238
3.10.4.2 HTTP DELETE . . . 239
3.10.5 All Versions Feed . . . 239
3.10.5.1 HTTP GET . . . 239
3.10.6 Type Descendants Feed . . . 240
3.10.6.1 HTTP GET . . . 240
3.11 Resources . . . 241
3.11.1 Type Entry . . . 241
3.11.1.1 HTTP GET . . . 241
3.11.1.2 HTTP PUT . . . 242
3.11.1.3 HTTP DELETE . . . 242
3.11.2 Document Entry . . . 242
3.11.2.1 HTTP GET . . . 242
3.11.2.2 HTTP PUT . . . 244
3.11.2.3 HTTP DELETE . . . 244
3.11.3 PWC Entry . . . 244
3.11.3.1 HTTP GET . . . 244
3.11.3.2 HTTP PUT . . . 245
3.11.3.3 HTTP DELETE . . . 246
3.11.4 Folder Entry . . . 246
3.11.4.1 HTTP GET . . . 246
3.11.4.2 HTTP PUT . . . 247
3.11.4.3 HTTP DELETE . . . 248
3.11.5 Relationship Entry . . . 248
3.11.5.1 HTTP GET . . . 248
3.11.5.2 HTTP PUT . . . 249
3.11.5.3 HTTP DELETE . . . 249
3.11.6 Policy Entry . . . 250
3.11.6.1 HTTP GET . . . 250
3.11.6.2 HTTP PUT . . . 250
3.11.6.3 HTTP DELETE . . . 251
3.11.7 Item Entry . . . 251
3.11.7.1 HTTP GET . . . 251
3.11.7.2 HTTP PUT . . . 252
3.11.7.3 HTTP DELETE . . . 252
3.11.8 Content Stream . . . 253
3.11.8.1 HTTP GET . . . 253
3.11.8.2 HTTP PUT . . . 253
3.11.8.3 HTTP DELETE . . . 254
3.11.9 AllowableActions Resource . . . 254
3.11.9.1 HTTP GET . . . 254
3.11.10ACL Resource . . . 254
3.11.10.1HTTP GET . . . 254
3.11.10.2HTTP PUT . . . 255

4 Web Services Binding 256
4.1 Overview . . . 256
4.1.1 WS-I . . . 256
4.1.2 Authentication . . . 256
4.1.3 Content Transfer . . . 256
4.1.4 Reporting Errors . . . 256
4.2 Web Services Binding Mapping . . . 256
4.3 Additions to the Services section . . . 257

4.3.1 updateProperties and checkIn Semantics . . . 257
4.3.2 Content Ranges . . . 257
4.3.3 Extensions . . . 257
4.3.4 Web Services Specific Structures . . . 257
4.3.4.1 cmisFaultType and cmisFault . . . 257
4.3.4.2 cmisRepositoryEntryType . . . 257
4.3.4.3 cmisTypeContainer . . . 257
4.3.4.4 cmisTypeDefinitionListType . . . 257
4.3.4.5 cmisObjectInFolderType, cmisObjectParentsType and cmisObjectInFolderContainerType . . . 258
4.3.4.6 cmisObjectListType and cmisObjectInFolderListType . . . 258
4.3.4.7 cmisContentStreamType . . . 258
4.3.4.8 cmisACLType . . . 259
4.3.4.9 cmisExtensionType . . . 259

5 Browser Binding 260
5.1 Overview . . . 260
5.2 Common Service Elements . . . 260
5.2.1 Protocol . . . 260
5.2.2 Data Representation . . . 260
5.2.3 Schema . . . 260
5.2.4 Mapping Schema Elements to JSON . . . 260
5.2.5 URL Patterns . . . 261
5.2.6 Multipart Forms . . . 261
5.2.7 Properties in a "value not set" state . . . 261
5.2.8 Callback . . . 261
5.2.9 Authentication . . . 262
5.2.9.1 Basic Authentication for Non-Browser Clients . . . 262
5.2.9.2 Authentication with Tokens for Browser Clients . . . 262
5.2.10 Error Handling and Return Codes . . . 264
5.2.11 Succinct Representation of Properties . . . 265
5.3 URLs . . . 265
5.3.1 Service URL . . . 265
5.3.2 Repository URL . . . 266
5.3.3 Root Folder URL . . . 266
5.3.4 Object URLs . . . 266
5.4 Services . . . 266
5.4.1 Service URL . . . 267
5.4.2 Repository URL . . . 267
5.4.2.1 Selector "repositoryInfo" . . . 267
5.4.2.2 Selector "typeChildren" . . . 268
5.4.2.3 Selector "typeDescendants" . . . 269
5.4.2.4 Selector "typeDefinition" . . . 269
5.4.2.5 Selector "checkedOut" . . . 270
5.4.2.6 Action "createDocument" . . . 271
5.4.2.7 Action "createDocumentFromSource" . . . 271
5.4.2.8 Action "createRelationship" . . . 272
5.4.2.9 Action "createPolicy" . . . 272
5.4.2.10 Action "createItem" . . . 273
5.4.2.11 Action "bulkUpdate" . . . 273
5.4.2.12 Selector "query" . . . 274
5.4.2.13 Action "query" . . . 274
5.4.2.14 Selector "contentChanges" . . . 275
5.4.2.15 Action "createType" . . . 276
5.4.2.16 Action "updateType" . . . 276
5.4.2.17 Action "deleteType" . . . 276
5.4.2.18 Selector "lastResult" . . . 277

5.4.3 Object URL . . . 277
5.4.3.1 Selector "children" . . . 277
5.4.3.2 Selector "descendants" . . . 278
5.4.3.3 Selector "folderTree" . . . 279
5.4.3.4 Selector "parent" . . . 279
5.4.3.5 Selector "parents" . . . 280
5.4.3.6 Selector "checkedout" . . . 281
5.4.3.7 Action "createDocument" . . . 281
5.4.3.8 Action "createDocumentFromSource" . . . 282
5.4.3.9 Action "createFolder" . . . 282
5.4.3.10 Action "createPolicy" . . . 283
5.4.3.11 Action "createItem" . . . 283
5.4.3.12 Selector "allowableActions" . . . 284
5.4.3.13 Selector "object" . . . 284
5.4.3.14 Selector "properties" . . . 285
5.4.3.15 Selector "object" . . . 285
5.4.3.16 Selector "content" . . . 286
5.4.3.17 Selector "renditions" . . . 286
5.4.3.18 Action "update" . . . 287
5.4.3.19 Action "move" . . . 287
5.4.3.20 Action "delete" . . . 288
5.4.3.21 Action "deleteTree" . . . 288
5.4.3.22 Action "setContent" . . . 289
5.4.3.23 Action "appendContent" . . . 289
5.4.3.24 Action "deleteContent" . . . 290
5.4.3.25 Action "addObjectToFolder" . . . 290
5.4.3.26 Action "removeObjectFromFolder" . . . 291
5.4.3.27 Action "checkOut" . . . 291
5.4.3.28 Action "cancelCheckOut" . . . 292
5.4.3.29 Action "checkIn" . . . 292
5.4.3.30 Selector "object" . . . 293
5.4.3.31 Selector "properties" . . . 294
5.4.3.32 Selector "versions" . . . 294
5.4.3.33 Selector "relationships" . . . 295
5.4.3.34 Selector "policies" . . . 295
5.4.3.35 Action "applyPolicy" . . . 296
5.4.3.36 Action "removePolicy" . . . 296
5.4.3.37 Action "applyACL" . . . 297
5.4.3.38 Selector "acl" . . . 297
5.4.4 Use of HTML Forms . . . 297
5.4.4.1 Action . . . 298
5.4.4.2 Structured and Array Parameters . . . 298
5.4.4.3 CMIS Controls . . . 298
5.4.4.4 Access to Form Response Content . . . 308

6 Conformance **312**

A IANA Considerations **314**
A.1 Content-Type Registration . . . 314
A.1.1 CMIS Query . . . 314
A.1.2 CMIS AllowableActions . . . 314
A.1.3 CMIS Tree . . . 315
A.1.4 CMIS Atom . . . 316
A.1.5 CMIS ACL . . . 316

B Schema Language (Orderly) **318**
B.1 Overview . . . 318

B.2 A subset of JSONSchema . . . 318
B.3 A Non-Normative Tutorial . . . 319
B.3.1 Comments and Whitespace . . . 319
B.3.2 Property Names . . . 319
B.3.3 Common Properties . . . 319
B.3.4 String Types . . . 320
B.3.5 Number and Integer types . . . 320
B.3.6 Boolean Types . . . 320
B.3.7 Object Types . . . 320
B.3.8 Array Types . . . 321
B.3.9 Additional properties in arrays and objects . . . 321
B.3.10 Null Types . . . 321
B.3.11 Any types . . . 322
B.3.12 Unions . . . 322
B.3.13 Maps . . . 322
B.3.14 Extensions or Extra Properties . . . 322
B.3.15 ID's . . . 323
B.3.16 References . . . 323
B.3.17 Bases . . . 323
B.3.18 More Complex Examples . . . 323
B.3.19 Cautions . . . 324
B.4 The Normative Grammar . . . 325

C Acknowledgements 328

D Change log 330

1 Introduction

The Content Management Interoperability Services (CMIS) standard defines a domain model and Web Services, Restful AtomPub and browser (JSON) bindings 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

[RFC1867]	E. Nebel, L. Masinter, Form-based File Upload in HTML, http://www.ietf.org/rfc/rfc1867.txt, November 1995
[RFC2045]	N. Freed, N. Borenstein, Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, http://www.ietf.org/rfc/rfc2045.txt, November 1996
[RFC2046]	N. Freed, N. Borenstein, Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, http://www.ietf.org/rfc/rfc2046.txt, November 1996
[RFC2119]	S. Bradner, Key words for use in RFCs to Indicate Requirement Levels, http://www.ietf.org/rfc/rfc2119.txt, March 1997
[RFC2388]	L. Masinter, Returning Values from Forms: multipart/form-data http://www.ietf.org/rfc/rfc2388.txt, August 1998
[RFC2616]	R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, Hypertext Transfer Protocol -- HTTP/1.1, http://www.ietf.org/rfc/rfc2616.txt, June 1999
[RFC2617]	J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, L. Stewart, HTTP Authentication: Basic and Digest Access Authentication, http://www.ietf.org/rfc/rfc2617.txt, June 1999
[RFC2818]	Rescorla, E., HTTP Over TLS, http://www.ietf.org/rfc/rfc2818.txt, May 2000
[RFC3023]	M. Murata, S. St.Laurent, D. Kohn, XML Media Types, http://www.ietf.org/rfc/rfc3023.txt, January 2001
[RFC3986]	T. Berners-Lee, R. Fielding, L. Masinter, Unified Resource Identifier, http://www.ietf.org/rfc/rfc3986.txt, January 2005

[RFC4287]	M. Nottingham, R. Sayre, Atom Syndication Format, http://www.ietf.org/rfc/rfc4287.txt, December 2005
[RFC4288]	N. Freed, J. Klensin, Media Type Specifications and Registration Procedures, http://www.ietf.org/rfc/rfc4288.txt, December 2005
[RFC4627]	D. Crockford, The application/json Media Type for JavaScript Object Notation (JSON), http://www.ietf.org/rfc/rfc4627.txt, July 2006
[RFC4918]	L. Dusseault, HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV), http://www.ietf.org/rfc/rfc4918.txt, June 2007
[RFC5023]	J. Gregorio, B. de hOra, Atom Publishing Protocol, http://www.ietf.org/rfc/rfc5023.txt, October 2007
[RFC6234]	D. Eastlake 3rd, T. Hansen, US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF), http://www.ietf.org/rfc/rfc6234.txt, May 2011
[RFC6266]	J. Reschke, Use of the Content-Disposition Header Field in the Hypertext Transfer Protocol (HTTP), http://www.ietf.org/rfc/rfc6266.txt, June 2011
[XMLSchema]	W3C, XML Schema Part 2: Datatypes Second Edition, http://www.w3.org/TR/xmlschema-2/, 28 October 2004
[SameOriginPolicy]	W3C, Same Origin Policy, http://www.w3.org/Security/wiki/Same_Origin_Policy, January 2010
[ID-Brown]	J. Reschke Editor, A. Brown, G. Clemm, Link Relation Types for Simple Version Navigation between Web Resources, http://tools.ietf.org/id/draft-brown-versioning-link-relations-07.txt, 2010
[ID-WebLinking]	M. Nottingham, Web Linking, http://tools.ietf.org/id/draft-nottingham-http-link-header-07.txt, 2010

1.3 Non-Normative References

1.4 Examples

A set of request and response examples is attached to this specification document. These examples are non-normative and their sole purpose is to illustrate the data structures and bindings that are defined in this specification.

Boxes like the following point to appropriate example files throughout this document. There is usually a request file describing the request sent from a CMIS client to a CMIS repository and a matching response file that contains the content returned from the CMIS repository.

Example:
Request: `atompub/getChildren-request.log`
Response: `atompub/getChildren-response.log`

1.5 Changes for the CMIS 1.1 specification

This section provides a very brief description of each major new CMIS 1.1 feature along with links to the sections of this document for complete descriptions.

1.5.1 Type Mutability

Defines the services and protocol binding extensions that allow CMIS clients to create, modify and delete Type Definitions and Property Definitions for a given repository.

Please see section 2.1.10 Object-Type Creation, Modification and Deletion for a detailed discussion of this feature.

1.5.2 Repository Features

Defines additional schema for the `getRepositoryInfo` service that allows CMIS clients to discover any extensions or additional CMIS based standards supported on each repository.

Please see section 2.1.1.3 Repository Features for a detailed discussion of this feature.

1.5.3 Secondary object types

Defines named sets of properties that can be dynamically added and removed from CMIS objects.

Please see section 2.1.9 Secondary Object-Types for a detailed discussion of this feature.

1.5.4 Retention and Hold Support

Defines secondary types for formally representing Retentions and Holds on CMIS objects. These in turn can be used by the repository to protect objects from being deleted or modified. A Retention describes a period of time that a document must not be deleted, while a Hold marks the document as protected as long as the Hold is applied.

Please see section 2.1.16 Retentions and Holds for a detailed discussion of these features.

1.5.5 Browser Binding

A new optional binding specifically designed to support applications running in a web browser or other client without the need for any additional client libraries. Notable among the differences in this binding are the use of JSON (Java Script Object Notation, [RFC4627]) instead of XML and the exclusive use of HTTP GET and POST for all operations.

Please see section 5 Browser Binding for a detailed discussion of this feature.

1.5.6 New cmis:item Object Type

A new top level data model type that is an extension point for repositories that need to expose any other object types via CMIS that do not fit the model's definition for document, folder, relationship or policy.

Please see section 2.1.8 Item Object for a detailed discussion of this feature.

1.5.7 Service bulkUpdateProperties

A method for supporting bulk property updates on a set of objects within a single service call.

Please see section 2.2.4.14 bulkUpdateProperties for a detailed discussion of this feature.

1.5.8 Append to a content stream

Support for appending to a content stream. Enables clients to break very large uploads of document content into numerous smaller calls.

Please see section 2.2.4.19 appendContentStream for a detailed discussion of this feature.

2 Domain Model

2.1 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 repository. (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 MUST use the CMIS `getRepositories` service to obtain a list of repositories that are available at that endpoint. The repository id MUST uniquely identify an available repository at this service endpoint. Both the repository name and the repository id are opaque to CMIS. Aside from the `getRepositories` service, all other CMIS services are single-repository-scoped, and require a repository id as an input parameter. In other words, except for the `getRepositories` service, multi-repository and inter-repository operations are not supported by CMIS.

2.1.1 Repository

The repository itself is described by the CMIS "Get Repository Information" service. The service output is fully described in section 2.2.2.2 getRepositoryInfo.

2.1.1.1 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 `getRepositoryInfo` service. The following is the list of these optional capabilities. All capabilities are "boolean" (i.e. the repository either supports the capability entirely or not at all) unless otherwise noted.

Navigation Capabilities

capabilityGetDescendants
Ability for an application to enumerate the descendants of a folder via the `getDescendants` service.
See section 2.2.3.2 getDescendants.

capabilityGetFolderTree
Ability for an application to retrieve the folder tree via the `getFolderTree` service.
See section 2.2.3.3 getFolderTree.

capabilityOrderBy

Indicates the ordering capabilities of the repository.
Valid values are:

none Ordering is not supported.

common Only common CMIS properties are supported. See section 2.2.1.2.7 Object Order for the list of properties.

custom Common CMIS properties and custom object-type properties are supported.

See section 2.2.1.2.7 Object Order.

Object Capabilities

capabilityContentStreamUpdatability

Indicates the support a repository has for updating a documents content stream.
Valid values are:

none The content stream may never be updated.

anytime The content stream may be updated any time.

pwconly The content stream may be updated only when checked out. Private Working Copy (PWC) is described in section 2.1.13 Versioning.

See section 2.1.4.1 Content Stream.

capabilityChanges

Indicates what level of changes (if any) the repository exposes via the `getContentChanges` service.
Valid values are:

none The repository does not support the change log feature.

objectidsonly The change log can return only the object ids for changed objects in the repository and an indication of the type of change, not details of the actual change.

properties The change log can return properties and the object id for the changed objects.

all The change log can return the object ids for changed objects in the repository and more information about the actual change.

See section 2.1.15 Change Log.

capabilityRenditions

Indicates whether or not the repository exposes renditions of document or folder objects.
Valid values are:

none The repository does not expose renditions at all.

read Renditions are provided by the repository and readable by the client.

See section 2.1.4.2 Renditions.

Filing Capabilities

capabilityMultifiling

Ability for an application to file a document or other file-able object in more than one folder.
See section 2.1.5 Folder Object.

capabilityUnfiling

Ability for an application to leave a document or other file-able object not filed in any folder.
See section 2.1.5 Folder Object.

capabilityVersionSpecificFiling

Ability for an application to file individual versions (i.e., not all versions) of a document in a folder.
See section 2.1.13 Versioning.

Versioning Capabilities

capabilityPWCUpdatable
Ability for an application to update the "Private Working Copy" of a checked-out document.
See section 2.1.13 Versioning.

capabilityPWCSearchable
Ability of the Repository to include the "Private Working Copy" of checked-out documents in query search scope; otherwise PWC's are not searchable.
See section 2.1.13 Versioning.

capabilityAllVersionsSearchable
Ability of the Repository to include all versions of document. If False, typically either the latest or the latest major version will be searchable.
See section 2.1.13 Versioning.

Query Capabilities

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

none No queries of any kind can be fulfilled.

metadataonly Only queries that filter based on object properties can be fulfilled. Specifically, the CONTAINS() predicate function is not supported.

fulltextonly Only queries that filter based on the full-text content of documents can be fulfilled. Specifically, only the CONTAINS() predicate function can be included in the WHERE clause.

bothseparate The repository can fulfill queries that filter EITHER on the full-text content of documents OR on their properties, but NOT if both types of filters are included in the same query.

bothcombined The repository can fulfill queries that filter on both the full-text content of documents and their properties in the same query.

See section 2.1.14 Query.

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

none The repository cannot fulfill any queries that include any JOIN clauses on two primary types. If the Repository supports secondary types, JOINs on secondary types SHOULD be supported, even if the support level is `none`.

inneronly The repository can fulfill queries that include an INNER JOIN clause, but cannot fulfill queries that include other types of JOIN clauses.

innerandouter The repository can fulfill queries that include any type of JOIN clause defined by the CMIS query grammar.

See section 2.1.14 Query.

Type Capabilities

capabilityCreatablePropertyTypes A list of all property data types that can be used by a client to create or update an object-type definition. See sections 2.1.2.1 Property and 2.1.10.1 General Constraints on Metadata Changes.

capabilityNewTypeSettableAttributes
Indicates which object-type attributes can be set by a client when a new object-type is created. This capibility is a set of booleans; one for each of the following attributes:

- id
- localName
- localNamespace
- displayName

- queryName
- description
- creatable
- fileable
- queryable
- fulltextIndexed
- includedInSupertypeQuery
- controllablePolicy
- controllableACL

The repository MUST set the object-type attributes that cannot be set by a client or are not set by a client.
See section 2.1.10 Object-Type Creation, Modification and Deletion.

ACL Capabilities

capabilityACL

Indicates the level of support for ACLs by the repository.

none The repository does not support ACL services.

discover The repository supports discovery of ACLs (`getACL` and other services).

manage The repository supports discovery of ACLs AND applying ACLs (`getACL` and `applyACL` services).

See section 2.1.12 Access Control.

2.1.1.2 Implementation Information

The `getRepositoryInfo` service MUST also return implementation information including vendor name, product name, product version, version of CMIS that it supports, the root folder id (see section 2.1.5.2 Folder Hierarchy), and MAY include other implementation-specific information. The version of CMIS that the repository supports MUST be expressed as a String that matches the specification version. For this version it is the string "1.1".

2.1.1.3 Repository Features

Repositories MAY provide information about additional features that are supported by the repository but that are outside the CMIS specification. This information is returned by the `getRepositoryInfo` service.

Clients that don't understand this information SHOULD ignore it.

The repository MUST provide a unique id for each feature. This id SHOULD take the form of a URI (see [RFC3986]). The repository MAY also provide a version label as well as a human-readable common name and description for each feature.

Furthermore, each feature MAY supply an arbitrary number of key-value pairs. The semantics and rules for these key-value pairs are not defined by CMIS but MAY be constrained by other specifications.

2.1.2 Object

The entities managed by CMIS are modeled as typed *objects*. There are five primary base types of objects: *document objects*, *folder objects*, *relationship objects*, *policy objects*, and *item 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*, *document objects*, *policy objects*, and *item 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 a directional relationship between two objects. The support for relationship objects is optional. The `getTypeChildren` service when asked for the base object-types MUST return the base relationship object-type if the repository supports relationships.
- A *policy object* represents an administrative policy, which may be "applied" to one or more "controllablePolicy" objects. Whether or not an object is controllable is specified in its object-type definition. The support for policy objects is optional. The `getTypeChildren` service when asked for the base object-types MUST return the base policy object-type if the repository supports policies.
- An *item object* represents a generic type of CMIS information asset. Item objects are not versionable and do not have content streams like documents but have properties like all other CMIS objects. The support for item objects is optional. The `getTypeChildren` service when asked for the base object-types MUST return the base item object-type if the repository supports items.

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. Furthermore, object-type management services are provided to create, modify and delete object-types if that is supported by the repository.

Every CMIS object has an opaque and immutable *object id*, which is assigned by the repository when the object is created. An id uniquely identifies an object within a repository regardless of the type of the object. Repositories SHOULD assign ids that are "permanent" -- that is, they remain unchanged during the lifespan of the identified objects, and they are never reused or reassigned after the objects are deleted from the repository.

Every CMIS object has a set of named, but not explicitly ordered, *properties*. (However, a repository SHOULD always return object properties in a consistent order.) Within an object, each property is uniquely identified by its property definition id. The object properties are defined by the object-type.

An object must have one and only one primary object-type, which cannot be changed. An object's primary object-type may be simply called its object-type. The primary object-type of an object classifies the object and defines the properties that the object must have.

An object MAY have zero or more secondary object types applied to it. A secondary type is a named marking that may add extra properties to an object in addition to the properties defined by the object's primary type. That is, applying a secondary type to an object adds the properties defined by this type to the object. Removing a secondary type removes the properties. Secondary object-types can only be defined as subtypes or descendant types of the `cmis:secondary` base type. All other base object types and their descendant types are primary object-types.

Consequently, each instance of a primary object-type corresponds to a distinct object, whereas each instance of a secondary object type does not. Therefore, the "creatable", "fileable", "controllablePolicy", and "controllableACL" object type attributes are not applicable to a secondary object type and must be set to FALSE.

The support for secondary types is optional, and may be discovered via the `getTypeChildren` service. See section 2.1.9 Secondary Object-Types.

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, MUST, or MUST NOT have a content stream. A document

MAY also have one or more *renditions* associated with it. A rendition can be a thumbnail or an alternate representation of the content stream.

Objects MAY have one *Access Control List* (ACL), which controls access to the object. A set of policy objects may also control access to the object. An ACL represents a list of *Access Control Entries* (ACEs). An ACE in turn represents one or more permissions being granted to a principal (a user, group, role, or something similar).

The notion of localization of the objects in the data model is entirely repository specific.

CMIS objects MAY expose additional information, such as vendor-specific workflow data, beyond the attributes described above. In this respect, the data model can be extended as desired. This specification does not standardize such extensions.

2.1.2.1 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.

A property, either single-valued or multi-valued, MAY be in a "not set" state. CMIS does not support "null" property value. If a multi-valued property is not in a "not set" state, its property value MUST be a non-empty list of individual values. Each individual value in the list MUST NOT be in a "not set" state and MUST conform to the property's property-type.

A multi-valued property is either set or not set in its entirety. An individual value of a multi-valued property MUST NOT be in an individual "value not set" state and hold a position in the list of values. An empty list of values MUST 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" (see [XMLSchema]):

string (xsd:string)

boolean (xsd:boolean)

decimal (xsd:decimal)
(see section 2.1.3.3.5 Attributes specific to Decmial Object-Type Property Definitions for attributes specific to Decimal object-type property definitions.)

integer (xsd:integer)
(see section 2.1.3.3.3 Attributes specific to Integer Object-Type Property Definitions for attributes specific to Integer object-type property definitions.)

datetime (xsd:dateTime)
(see section 2.1.3.3.4 Attributes specific to DateTime Object-Type Property Definitions for attributes specific to DateTime object-type property definitions.)

uri (xsd:anyURI)

In addition, the following property-types are also specified by CMIS:

id

html

Individual protocol bindings MAY override or re-specify these property-types.

For single valued String, Id and HTML properties, a repository MAY support the distinction between a set value with an empty string (length = 0), and a "not set" value. In this case an empty value element (e.g.

<cmis:value/>) inside of a property element will indicate a "set but empty" string property. A property element without a <cmis:value/> will indicate a property in a "not set" state. For repositories that do not support this distinction the latter example (absence of the <cmis:value> element) should be used for all cases.

2.1.2.1.1 Id Property

An id property holds a system-generated, read-only identifier, such as an object id, an object-type id, etc. (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. 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 a `constraint` 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 the exception and suggest a remedial action. The content of such messages is outside the scope of CMIS.

2.1.2.1.2 HTML Property

An HTML property holds a document or fragment of Hypertext Markup Language (HTML) content. HTML properties are not guaranteed to be validated in any way. The validation behavior is entirely repository specific.

2.1.2.1.3 Query Names

All properties MUST supply a string `queryName` attribute which is used for query and filter operations on object-types. This is an opaque string with limitations. This string SHOULD NOT contain any characters that negatively interact with the BNF grammar.
The string MUST NOT contain:

- whitespace " "
- comma ","
- double quotes ""
- single quotes "'"
- backslash "\"
- the period "."
- the open "(" or close ")" parenthesis characters

2.1.3 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. If a property not specified in an object's object-type definition is supplied by an application, an exception SHOULD be thrown.

Each object-type is uniquely identified within a repository by a system-assigned and immutable *object-type identifier*, which is of type Id.

A CMIS repository MUST expose exactly one collection of object-types via the "repository" services (`getTypeChildren`, `getTypeDescendants`, `getTypeDefinition`).

While a repository MAY define additional object-types beyond the CMIS base object-types, these object-types MUST 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.

2.1.3.1 Object-Type Hierarchy and Inheritance

Hierarchy and *Inheritance* for object-types are supported by CMIS in the following manner:

- A CMIS repository MUST have these base types:
  - `cmis:document` object-type
  - `cmis:folder` object-type
- A CMIS repository MAY have these base types:
  - `cmis:relationship` object-type
  - `cmis:policy` object-type
  - `cmis:item` object-type
  - `cmis:secondary` object-type
- Additional base types MUST NOT exist. Additional object-types MAY be defined as sub-types or descendant types of these six base 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 attribute values (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 MUST 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 MUST be as if the value of this property is "not set" for all objects of this sub-type.
- The scope of a query on a given object-type is automatically expanded to include all the *descendant types* of the given object-type with the attribute `includedInSuperTypeQuery` equals TRUE. This was added for synthetic types as well as to support different type hierarchies that are not necessarily the same as CMIS. Only the properties of the given object-type, including inherited ones, MUST 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 MUST still appear as a column in the corresponding virtual table in the relational view, but this column MUST contain a "not set" value for all objects of this type. (See section 2.1.14 Query)

2.1.3.2 Object-Type Attributes

2.1.3.2.1 Attributes common to ALL Object-Type Definitions

All *object-type definitions* MUST contain the following *attributes*. Optional attributes MUST be defined but MAY have "not set" values.

id Id

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

localName String

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

localNamespace String (optional)

This attribute allows repositories to represent the internal namespace of the underlying repository's name for the object-type.

queryName String (optional)

Used for query and filter operations on object-types. This is an opaque string with limitations. See 2.1.2.1.3 Query Names for details.

displayName String (optional)

Used for presentation by application.

baseId Enum

A value that indicates whether the base type for this object-type is the document, folder, relationship, policy, item, or secondary base type.

parentId Id

The id of the object-type's immediate parent type. It MUST be "not set" for a base type. Depending on the binding this means it might not exist on the base type object-type definition.

description String (optional)

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

creatable Boolean

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

fileable Boolean

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

queryable Boolean

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

controllablePolicy Boolean

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

controllableACL Boolean

This attribute indicates whether or not objects of this type are controllable by ACL's. Only objects that are controllableACL can have an ACL.

fulltextIndexed Boolean

Indicates whether objects of this type are indexed for full-text search for querying via the CONTAINS() query predicate. If the value of this attribute is TRUE, the full-text index MUST cover the content and MAY cover the metadata.

includedInSupertypeQuery Boolean

Indicates whether this type and its subtypes appear in a query of this type's ancestor types. For example: if `Invoice` is a sub-type of `cmis:document`, if this is TRUE on `Invoice` then for a query on `cmis:document`, 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.

typeMutability.create Boolean (optional)

Indicates whether new child types may be created with this type as the parent.

typeMutability.update Boolean (optional)

Indicates whether clients may make changes to this type per the constraints defined in this specification.

typeMutability.delete Boolean (optional)

Indicates whether clients may delete this type if there are no instances of it in the repository.

2.1.3.3 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, MUST be retrievable through the "get" services, and MAY appear in the SELECT clause of a query.

2.1.3.3.1 Property Types

Property types are defined in section 2.1.2.1 Property.

2.1.3.3.2 Attributes common to ALL Object-Type Property Definitions

All *object-type property definitions* MUST contain the following *attributes*. Optional attributes MUST be defined but MAY have "not set" values.

id Id

This opaque attribute uniquely identifies the property in the repository. If two object-types each contain property definitions with the same id, the basic property definitions (property type, query name, cardinality) MUST be the same. Other attributes MAY be different for each type.

localName String (optional)

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

localNamespace String (optional)

This attribute allows repositories to represent the internal namespace of the underlying repository's name for the property.

queryName String (optional)

Used for query operations on properties. This is an opaque string with limitations. See 2.1.2.1.3 Query Names for details.

displayName String (optional)

Used for presentation by application.

description String (optional)

This is an optional attribute containing a description of the property.

propertyType Enum

This attribute indicates the type of this property. It MUST be one of the allowed property types. (See section 2.1.2.1 Property.)

cardinality Enum

Indicates whether the property can have "zero or one" or "zero or more" values.
Values:
single Property can have zero or one values (if property is not required), or exactly one value (if property is required).
multi Property can have zero or more values (if property is not required), or one or more values (if property is required).
Repositories SHOULD preserve the ordering of values in a multi-valued property. That is, the order in which the values of a multi-valued property are returned in "get" services operations SHOULD be the same as the order in which they were supplied during previous create/update operation.

updatability Enum

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

readonly The value of this property MUST NOT ever be set directly by an application. It is a system property that is either maintained or computed by the repository. The value of a read-only property MAY be indirectly modified by other repository interactions (for example, calling `updateProperties` on an object will change the object's last modified date, even though that property cannot be directly set via an `updateProperties` service call.)

readwrite The property value can be modified using the `updateProperties` service.

whencheckedout The property value MUST only be update-able using a "private working copy" document. That is, the update is either made on a "private working copy" object or made using the `checkIn` service.

oncreate The property value MUST only be update-able during the create operation on that object.

inherited Boolean

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

required Boolean

This attribute is only applicable to non-system properties, i.e. properties whose value is provided by the application.
If TRUE, then the value of this property MUST never be set to the "not set" state when an object of this type is created/updated. If not provided during a create or update operation, the repository MUST provide a value for this property. If a value is not provided, then the default value defined for the property MUST be set. If no default value is provided and no default value is defined, the repository MUST throw a `constraint` exception.
This attribute is not applicable when the "updatability" attribute is "readonly". In that case, "required" SHOULD be set to FALSE.
Note: For CMIS-defined object-types, the value of a system property (such as `cmis:objectId`, `cmis:createdBy`) MUST be set by the repository. However, the property's "required" attribute SHOULD be FALSE because it is read-only to applications.

queryable Boolean

Indicates whether or not the property MAY appear in the WHERE clause of a CMIS query statement.
This attribute MUST have a value of FALSE if the object-type's attribute for "queryable" is set to FALSE.

orderable Boolean

Indicates whether the property can appear in the ORDER BY clause of a CMIS query statement or an `orderBy` parameter of `getChildren` or `getCheckedOutDocs`.
This property MUST be FALSE for any property whose cardinality is "multi".

choices <PropertyChoiceType list> (multi-valued, optional)

Indicates an explicit ordered set of single values allowed for this property.
If the cardinality of the property definition is "single" and the "openChoice" attribute is FALSE, then the property value MUST be at most one of the values listed in this attribute.
If the cardinality of the property definition is "single" and the "openChoice" attribute is TRUE, then the property value MAY be one of the values listed in this attribute.
If the cardinality of the property definition is "multi" and the "openChoice" attribute is FALSE, then the property value MUST be zero, one or more than one of the values listed in this attribute.
If the cardinality of the property definition is "multi" and the "openChoice" attribute is TRUE, then the property value MAY be zero, one, or more than one of the values listed in this attribute.
If this attribute is "not set", then any valid value for this property based on its type may be used.
Each choice includes a displayName and a value. The displayName is used for presentation purpose. The value will be stored in the property when selected.
Choices MAY be hierarchically presented. For example: a value of "choices" for a geographic location would be represented as follows:

- Europe:
  - England
  - France
  - Germany
- North America
  - Canada
  - USA
  - Mexico

openChoice Boolean (optional if `choices` is not set)

This attribute is only applicable to properties that provide a value for the "Choices" attribute.
If FALSE, then the data value for the property MUST 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> (optional)

The value that the repository MUST 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 MUST attempt to store a "not set" property value. If this occurs for a property that is defined to be required, then the creation attempt MUST throw an exception.
The attributes on the default value element are the same as the attributes on the property definition.

2.1.3.3.3 Attributes specific to Integer Object-Type Property Definitions

The following object attributes MUST only apply to property type definitions whose propertyType is "Integer", in addition to the common attributes specified above. A repository MAY provide additional guidance on what values can be accepted. If the following attributes are not present the repository behavior is undefined and it MAY throw an exception if a runtime constraint is encountered.

minValue Integer

The minimum value allowed for this property.
If an application tries to set the value of this property to a value lower than minValue, the repository MUST throw a `constraint` exception.

maxValue Integer

The maximum value allowed for this property.
If an application tries to set the value of this property to a value higher than maxValue, the repository MUST throw a `constraint` exception.

2.1.3.3.4 Attributes specific to DateTime Object-Type Property Definitions

The following object attributes MUST only apply to property type definitions whose propertyType is "DateTime", in addition to the common attributes specified above. A repository MAY provide additional guidance on what values can be accepted. If the following attributes are not present the repository behavior is undefined and it MAY throw an exception if a runtime constraint is encountered.

resolution Enum

This is the resolution supported for values of this property. Valid values for this attribute are:
year Year resolution is persisted. Date and time portion of the value should be ignored.
date Date resolution is persisted. Time portion of the value should be ignored.
time Time resolution is persisted.

2.1.3.3.5 Attributes specific to Decmial Object-Type Property Definitions

The following object attributes MUST only apply to property type definitions whose propertyType is "Decimal", in addition to the common attributes specified above. A repository MAY provide additional guidance on what values can be accepted. If the following attributes are not present the repository behavior is undefined and it MAY throw an exception if a runtime constraint is encountered.

precision Enum

This is the precision in bits supported for values of this property. Valid values for this attribute are:
32 32-bit precision ("single" as specified in IEEE-754-1985).
64 64-bit precision ("double" as specified in IEEE-754-1985).

minValue Decimal

The minimum value allowed for this property.
If an application tries to set the value of this property to a value lower than minValue, the repository MUST throw a `constraint` exception.

maxValue Decimal

The maximum value allowed for this property.
If an application tries to set the value of this property to a value higher than maxValue, the repository MUST throw a `constraint` exception.

2.1.3.3.6 Attributes specific to String Object-Type Property Definitions

The following object attributes MUST only apply to property type definitions whose propertyType is "String", in addition to the common attributes specified above. A repository MAY provide additional guidance on what values can be accepted. If the following attributes are not present the repository behavior is undefined and it MAY throw an exception if a runtime constraint is encountered.

maxLength Integer

The maximum length (in characters) allowed for a value of this property.
If an application attempts to set the value of this property to a string longer than the specified maximum length, the repository MUST throw a `constraint` exception.

2.1.4 Document Object

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

Depending on its object-type definition, a document object may be:

Version-able Can be acted upon via the Versioning Services (for example: `checkOut`, `checkIn`).

File-able Can be filed in zero, one, or more than one folder via the Multi-filing Services.

Query-able Can be located via the Discovery Services (for example: `query`).

Controllable-Policy Can have policies applied to it. (See section 2.1.7 Policy Object.)

Controllable-ACL Can have an ACL applied to it. (See section 2.1.12 Access Control.)

Additionally, whether a document object MUST, MAY or MUST NOT have a content stream is specified in its object-type definition. A document object MAY be associated with zero or more renditions.

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.

2.1.4.1 Content Stream

A content stream is a binary stream. Its maximum length is repository specific. Each content stream has a MIME Media Type, as defined by [RFC2045] and [RFC2046]. A content stream's attributes are represented as properties of the content stream's containing document object. There is no MIME type specific attribute or name directly associated with the content stream outside of the document object.

CMIS provides basic CRUD[1] services for content stream, using the id of a content stream's containing document object for identification. A content stream also has a `contentStreamId` which is used for access to the stream. The `setContentStream` service either creates a new content stream for a document object or replaces an existing content stream. The `appendContentStream` service either creates a new content stream or appends content to an existing content stream. The `getContentStream` service retrieves a content stream. The `deleteContentStream` service deletes a content stream from a document object. In addition, the `createDocument` and `checkIn` services MAY also take a content stream as an optional input. A content stream MUST be specified if required by the object-type definition. These are the only services that operate on content stream. The `getObject` and `query` services, for example, do not return a content stream.

`setContentStream`, `appendContentStream` and `deleteContentStream` services are considered modifications to a content stream's containing document object, and SHOULD therefore change the object's last modification date property upon successful completion.

The ability to set or delete a content stream is controlled by the `capabilityContentStreamUpdatability` capability.

2.1.4.2 Renditions

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

A CMIS repository MAY expose zero or more renditions for a document or folder in addition to a document's content stream. CMIS provides no capability to create or update renditions accessed through the rendition services. Renditions are specific to the version of the document or folder and may differ between document versions. Each rendition consists of a set of rendition attributes and a rendition stream. Rendition attributes are not object properties, and are not queryable. They can be retrieved using the `getRenditions` service.

[1] Create, Read, Update and Delete

A rendition stream can be retrieved using the `getContentStream` service with the rendition's streamId parameter.

2.1.4.2.1 Rendition Attributes

A rendition has the following attributes:

`streamId` Id

Identifies the rendition stream.

`mimeType` String

The MIME type of the rendition stream.

`length` Integer

The length of the rendition stream in bytes.

`title` String (optional)

Human readable information about the rendition.

`kind` String

A categorization String associated with the rendition. See section 2.1.4.2.2 Rendition Kind.

`height` Integer (optional)

Typically used for 'image' renditions (expressed as pixels).
SHOULD be present if kind = `cmis:thumbnail`.

`width` Integer (optional)

Typically used for 'image' renditions (expressed as pixels).
SHOULD be present if kind = `cmis:thumbnail`.

`renditionDocumentId` Id (optional)

If specified, then the rendition can also be accessed as a document object in the CMIS services. If not set, then the rendition can only be accessed via the rendition services. Referential integrity of this id is repository specific.

2.1.4.2.2 Rendition Kind

A rendition may be categorized via its kind. The repository is responsible for assigning kinds to renditions, including custom kinds. A rendition kind does not necessarily identify a single rendition for a given object.

CMIS defines the following kind:

cmis:thumbnail A rendition whose purpose is to provide an image preview of the document without requiring the client to download the full document content stream. Thumbnails are generally reduced fidelity representations.

2.1.4.3 Document Object-Type Definition

This section describes the definition of the document object-type's attribute values and property definitions which must be present on document instance objects. All attributes and property definitions are listed by their id.

2.1.4.3.1 Attributes specific to Document Object-Types

The following object attributes MUST only apply to object-type definitions whose baseId is the `cmis:document` object-type, in addition to the common attributes specified above:

`versionable` Boolean

Indicates whether or not objects of this type are version-able. (See section 2.1.13 Versioning.) If this attribute is set to TRUE, then documents of this type MUST be versionable. If this attribute is set to FALSE, then documents of this type MUST NOT be versionable.

`contentStreamAllowed` Enum

A value that indicates whether a content stream MAY, MUST, or MUST NOT be included in objects of this type.
Values:
notallowed A content stream MUST NOT be included.
allowed A content stream MAY be included.
required A content stream MUST be included (i.e. MUST be included when the object is created, and MUST NOT be deleted).

2.1.4.3.2 Attribute Values

The document object-type MUST have the following attribute values.

Notes:

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

`id`
Value: cmis:document

`localName`
Value: <repository-specific>

`localNamespace`
Value: <repository-specific>

`queryName`
Value: cmis:document

`displayName`
Value: <repository-specific>

baseId
Value: cmis:document

parentId
Value: MUST NOT be set

description
Value: <repository-specific>

creatable
Value: <repository-specific>

fileable
Value: SHOULD be TRUE

queryable
Value: SHOULD be TRUE

controllablePolicy
Value: <repository-specific>

controllableACL
Value: <repository-specific>

includedInSupertypeQuery
Value: <repository-specific>

fulltextIndexed
Value: <repository-specific>

typeMutability.create
Value: <repository-specific>

typeMutability.update
Value: <repository-specific>

typeMutability.delete
Value: <repository-specific>

versionable
Value: <repository-specific>

contentStreamAllowed
Value: <repository-specific>

2.1.4.3.3 Property Definitions

The document base object-type MUST have the following property definitions, and MAY include additional property definitions. Any attributes not specified for the property definition are repository specific. For all property definitions on base types, the query name MUST be the same as the property id. The repository MUST have the following property definitions on the document object-type:

cmis:name Name of the object.

Property Type:	String
Inherited:	FALSE
Required:	TRUE
Cardinality:	single
Updatability:	SHOULD be readwrite or whencheckedout
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	SHOULD be TRUE
Orderable:	SHOULD be TRUE

cmis:description Description of the object.

Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	SHOULD be readwrite or whencheckedout
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	<repository specific>
Orderable:	<repository specific>

If the repository doesn't support object descriptions, the Updatability SHOULD be readonly and the repository SHOULD return a "not set" value for this property.

cmis:objectId Id of the object.

Property Type:	Id
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	TRUE
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:baseTypeId Id of the base object-type for the object.

Property Type:	Id
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	SHOULD be TRUE
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:objectTypeId Id of the object's type.

Property Type:	Id
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	SHOULD be TRUE
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:secondaryObjectTypeIds Ids of the object's secondary types.

Property Type:	Id
Inherited:	FALSE
Required:	FALSE
Cardinality:	multi
Updatability:	readwrite if secondary types are supported, readonly otherwise
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	SHOULD be TRUE
Orderable:	FALSE

If the repository does not support secondary types, the repository MUST return "not set".

cmis:createdBy User who created the object.

Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	TRUE
Orderable:	TRUE

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:creationDate DateTime when the object was created.

Property Type:	DateTime
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	TRUE
Orderable:	TRUE

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:lastModifiedBy User who last modified the object.

Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	TRUE
Orderable:	TRUE

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:lastModificationDate — DateTime when the object was last modified.

Property Type:	DateTime
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	TRUE
Orderable:	TRUE

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:changeToken — Opaque token used for optimistic locking and concurrency checking. (See section 2.2.1.3 Change Tokens.)

Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	FALSE
Orderable:	FALSE

The repository MUST return this property with a non-empty value if the property filter does not exclude it. If the repository does not support change tokens, this property SHOULD not be set.

cmis:isImmutable	Defines if the object can be modified. If TRUE the repository MUST throw an error at any attempt to update or delete the object.
Property Type:	Boolean
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	<repository specific>
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:isLatestVersion	See section 2.1.13 Versioning.
Property Type:	Boolean
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	<repository specific>
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the property filter does not exclude it. Version property values are repository-specific when a document is defined as non-versionable.

cmis:isMajorVersion — See section 2.1.13 Versioning.

Property Type:	Boolean
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	<repository specific>
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the property filter does not exclude it. Version property values are repository-specific when a document is defined as non-versionable.

cmis:isLatestMajorVersion — See section 2.1.13 Versioning.

Property Type:	Boolean
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	<repository specific>
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the property filter does not exclude it. Version property values are repository-specific when a document is defined as non-versionable.

cmis:isPrivateWorkingCopy See section 2.1.13 Versioning.

Property Type:	Boolean
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	SHOULD be TRUE
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the property filter does not exclude it. Version property values are repository-specific when a document is defined as non-versionable.

cmis:versionLabel See section 2.1.13 Versioning.

Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	SHOULD be TRUE
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the property filter does not exclude it. Version property values are repository-specific when a document is defined as non-versionable.

cmis:versionSeriesId See section 2.1.13 Versioning.

Property Type:	Id
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	SHOULD be TRUE
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the property filter does not exclude it. Version property values are repository-specific when a document is defined as non-versionable.

cmis:isVersionSeriesCheckedOut See section 2.1.13 Versioning.

Property Type:	Boolean
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	SHOULD be TRUE
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the property filter does not exclude it. Version property values are repository-specific when a document is defined as non-versionable.

cmis:versionSeriesCheckedOutBy See section 2.1.13 Versioning.

Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	<repository specific>
Orderable:	<repository specific>

The repository SHOULD return this property with a non-empty value if the document is checked out and the property filter does not exclude it. The repository MUST return "not set" if the document is not checked out. Version property values are repository-specific when a document is defined as non-versionable.

cmis:versionSeriesCheckedOutId See section 2.1.13 Versioning.

Property Type:	Id
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	<repository specific>
Orderable:	<repository specific>

The repository SHOULD return this property with a non-empty value if the document is checked out, the PWC is visible to the current user and the property filter does not exclude it. If the PWC is not visible to the current user, the repository SHOULD return "not set". The repository MUST return "not set" if the document is not checked out. Version property values are repository-specific when a document is defined as non-versionable.

cmis:checkinComment	See section 2.1.13 Versioning.
Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	<repository specific>
Orderable:	<repository specific>

Version property values are repository-specific when a document is defined as non-versionable.

cmis:contentStreamLength	Length of the content stream (in bytes). See also section 2.1.4.1 Content Stream.
Property Type:	Integer
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	<repository specific>
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the document has a content stream and the property filter does not exclude it. If the document has no content stream, the repository MUST return "not set".

cmis:contentStreamMimeType — MIME type of the content stream.
See also section 2.1.4.1 Content Stream.

Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	<repository specific>
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the document has a content stream and the property filter does not exclude it. If the document has no content stream, the repository MUST return "not set".

cmis:contentStreamFileName — File name of the content stream.
See also section 2.1.4.1 Content Stream.

Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	<repository specific>
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the document has a content stream and the property filter does not exclude it. If the document has no content stream, the repository MUST return "not set".

cmis:contentStreamId	Id of the content stream. See also section 2.1.4.1 Content Stream.
Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	<repository specific>
Orderable:	<repository specific>

If the document has no content stream, the repository MUST return "not set".

2.1.5 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 specific containment semantics which MUST 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 section 2.1.6 Relationship Object.)

A folder object does not have a content-stream and is not version-able. A folder object MAY be associated with zero or more renditions (see section 2.1.4.2 Renditions).

2.1.5.1 File-able Objects

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

cmis:folder MUST be file-able

cmis:document MAY be file-able

cmis:relationship MUST NOT be file-able

cmis:policy MAY be file-able

cmis:item MAY be file-able

2.1.5.1.1 Document Version Series and Filing

Since document objects are versionable, a document object's membership in a folder 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 `getRepositoryInfo` service.

When the child objects of a folder are retrieved, a specific version of a document MAY be returned. If the repository supports version-specific filing, the specific version filed in that folder is returned. If the repository does not support version-specific filing, the latest version or the latest major version of the document is returned.

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.

2.1.5.1.2 Filing Restrictions by Object-Type

A folder collection's membership MAY be restricted by object-type. Each folder object has a multi-valued `cmis:allowedChildObjectTypeIds` property, which specifies that only objects of these types are allowed to be its children. If this property is "not set", then objects of any file-able type MAY be filed in the folder. It is repository-specific if subtypes of the types listed in the `cmis:allowedChildObjectTypeIds` property MAY be filed in the folder.

Because of these filing constraints, when a new folder object is created, an existing folder object MUST be specified as its parent.

When a non-file-able object is created, a parent folder MUST 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.

2.1.5.2 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. 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 objects. 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.

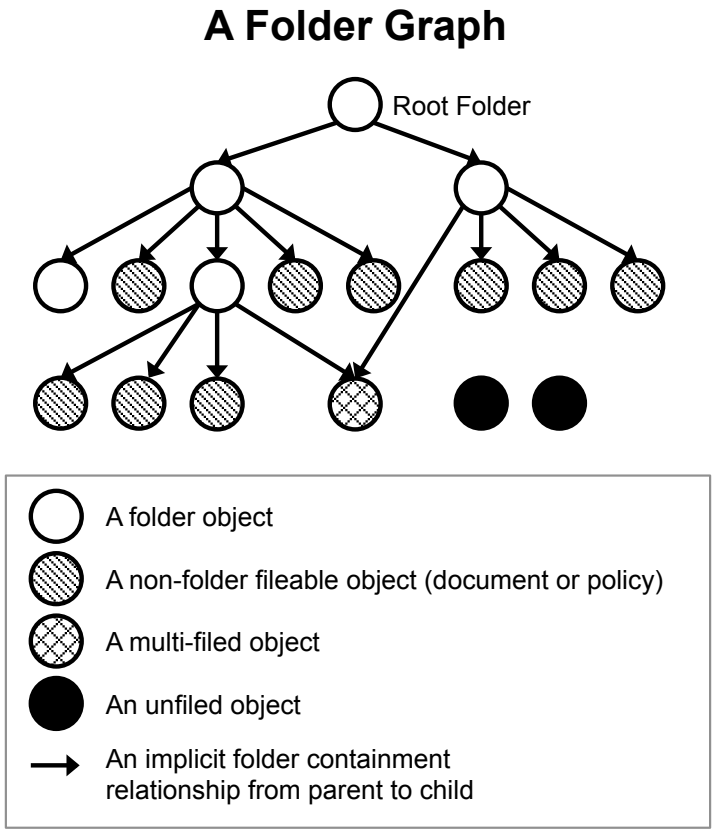

Figure 2.1: Folder Graph

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

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

2.1.5.3 Paths

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

- '/' for the root folder.
- All paths start with the root folder.
- A set of the folder and object path segments separated by '/' in order of closest to the root.
- Folder and object path segments are specified by `pathSegment` tokens which can be retrieved by all services that take an `includePathSegments` parameter (for example `getChildren`).
- A `pathSegment` token MUST not include a '/' character.
  It is repository specific how a repository chooses the value for `pathSegment`. Repositories might choose to use `cmis:name` or content stream filename for `pathSegment` token.
- The `pathSegment` token for each item MUST uniquely identify the item in the folder.

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

A path for an object may be calculated in the following way:

- If the object is the root folder, the path is '/'.
- If the object is a direct child of the root folder, the path is the object's `pathSegment` prefixed by '/'.
- If the object is not a direct child of the root folder, the path is item's parent folder `cmis:path` property appended by '/' and the object's `pathSegment`.

This constructed path may be given as input to the `getObjectByPath` service for object by path retrieval.

The `getObjectParents` service returns `relativePathSegment` tokens. These tokens are the `pathSegment` of the input object relative to the parent folders.

2.1.5.4 Folder Object-Type Definition

This section describes the definition of the folder object-type's attribute values and property definitions which must be present on folder instance objects. All attributes and property definitions are listed by their id.

2.1.5.4.1 Attribute Values

The folder object-type MUST have the following attribute values.

Notes:

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

id
Value: cmis:folder

localName
Value: <repository-specific>

localNamespace
Value: <repository-specific>

queryName
Value: cmis:folder

displayName
Value: <repository-specific>

baseId
Value: cmis:folder

parentId
Value: MUST NOT be set

description
Value: <repository-specific>

creatable
Value: <repository-specific>

fileable
Value: TRUE

queryable
Value: SHOULD be TRUE

controllablePolicy
Value: <repository-specific>

controllableACL
Value: <repository-specific>

includedInSupertypeQuery
Value: <repository-specific>

fulltextIndexed
Value: <repository-specific>

typeMutability.create
Value: <repository-specific>

typeMutability.update
Value: <repository-specific>

typeMutability.delete
Value: <repository-specific>

2.1.5.4.2 Property Definitions

The folder base object-type MUST have the following property definitions, and MAY include additional property definitions. Any attributes not specified for the property definition are repository specific. For all property definitions on base types, the query name MUST be the same as the property id. The repository MUST have the following property definitions on the folder object-type:

cmis:name	Name of the object.
Property Type:	String
Inherited:	FALSE
Required:	TRUE
Cardinality:	single
Updatability:	SHOULD be readwrite
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	SHOULD be TRUE
Orderable:	SHOULD be TRUE

cmis:description Description of the object.

Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	SHOULD be readwrite
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	<repository specific>
Orderable:	<repository specific>

If the repository doesn't support object descriptions, the Updatability SHOULD be readonly and the repository SHOULD return a "not set" value for this property.

cmis:objectId Id of the object.

Property Type:	Id
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	TRUE
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:baseTypeId Id of the base object-type for the object.

Property Type:	Id
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	SHOULD be TRUE
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:objectTypeId Id of the object's type.

Property Type:	Id
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	SHOULD be TRUE
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:secondaryObjectTypeIds Ids of the object's secondary types.

Property Type:	Id
Inherited:	FALSE
Required:	FALSE
Cardinality:	multi
Updatability:	readwrite if secondary types are supported, readonly otherwise
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	SHOULD be TRUE
Orderable:	FALSE

If the repository does not support secondary types, the repository MUST return "not set".

cmis:createdBy User who created the object.

Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	TRUE
Orderable:	TRUE

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:creationDate DateTime when the object was created.

Property Type:	DateTime
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	TRUE
Orderable:	TRUE

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:lastModifiedBy User who last modified the object.

Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	TRUE
Orderable:	TRUE

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:lastModificationDate DateTime when the object was last modified.

Property Type:	DateTime
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	TRUE
Orderable:	TRUE

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:changeToken	Opaque token used for optimistic locking and concurrency checking. (See section 2.2.1.3 Change Tokens.)
Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	FALSE
Orderable:	FALSE

The repository MUST return this property with a non-empty value if the property filter does not exclude it. If the repository does not support change tokens, this property SHOULD not be set.

cmis:parentId	Id of the parent folder of the folder.
Property Type:	Id
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	<repository specific>
Orderable:	FALSE

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:path	The fully qualified path to this folder. See section 2.1.5.3 Paths.
Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	<repository specific>
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:allowedChildObjectTypeIds	Id's of the set of object-types that can be created, moved or filed into this folder. See section 2.1.5.1.2 Filing Restrictions by Object-Type.
Property Type:	Id
Inherited:	FALSE
Required:	FALSE
Cardinality:	multi
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	FALSE
Orderable:	FALSE

2.1.6 Relationship Object

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

If a repository does not support relationship objects, the relationship base object-type SHOULD NOT be returned by a `getTypeChildren` service call.

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 independent objects, such as a document object, a folder object, a policy object, or an item 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 SHOULD NOT modify either the source or the target object. That is, it SHOULD NOT require an update capability (or permission) on either object; SHOULD NOT affect the versioning state of either object; and SHOULD NOT change their "Last Modification Date".

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 repository-specific. Therefore, creating an explicit relationship between two objects MAY impose a constraint on any of the three objects, and removing a relationship or deleting either the source or the target object MAY 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.

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

Additionally, a relationship object-type MAY specify that only objects of a specific object-type 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.

When a relationship object is created, the source object id and the target object id MUST reference valid non-relationship CMIS objects. When a relationship object is retrieved, its source object or target object MAY no longer exist, since referential integrity MAY not be maintained by a repository.

In addition to object CRUD services, a `getObjectRelationships` service 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).

2.1.6.1 Relationship Object-Type Definition

This section describes the definition of the relationship object-type's attribute values and property definitions which must be present on relationship instance objects. All attributes and property definitions are listed by their id.

2.1.6.1.1 Attributes specific to Relationship Object-Types

The following object attributes MUST only apply to object-type definitions whose baseId is the `cmis:relationship` object-type, in addition to the common attributes specified above:

`allowedSourceTypes` Id (multi-valued)

A list of object-type ids, indicating that the source object of a relationship object of this type MUST only be one of the types listed.
If this attribute is "not set", then the source object MAY be of any type.

`allowedTargetTypes` Id (multi-valued)

A list of object-type ids, indicating that the target object of a relationship object of this type MUST only be one of the types listed.
If this attribute is "not set", then the target object MAY be of any type.

2.1.6.1.2 Attribute Values

The relationship object-type MUST have the following attribute values.

Notes:

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

`id`
Value: cmis:relationship

`localName`
Value: <repository-specific>

`localNamespace`
Value: <repository-specific>

`queryName`
Value: cmis:relationship

`displayName`
Value: <repository-specific>

`baseId`
Value: cmis:relationship

`parentId`
Value: MUST NOT be set

`description`
Value: <repository-specific>

`creatable`
Value: <repository-specific>

fileable
Value: FALSE

queryable
Value: <repository-specific>

controllablePolicy
Value: <repository-specific>

controllableACL
Value: <repository-specific>

includedInSupertypeQuery
Value: <repository-specific>

fulltextIndexed
Value: <repository-specific>

typeMutability.create
Value: <repository-specific>

typeMutability.update
Value: <repository-specific>

typeMutability.delete
Value: <repository-specific>

allowedSourceTypes
Value: <repository-specific>

allowedTargetTypes
Value: <repository-specific>

2.1.6.1.3 Property Definitions

The relationship base object-type MUST have the following property definitions, and MAY include additional property definitions. Any attributes not specified for the property definition are repository specific. For all property definitions on base types, the query name MUST be the same as the property id. The repository MUST have the following property definitions on the relationship object-type:

cmis:name	Name of the object.
Property Type:	String
Inherited:	FALSE
Required:	TRUE
Cardinality:	single
Updatability:	SHOULD be readwrite
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	SHOULD be TRUE
Orderable:	SHOULD be TRUE

cmis:description Description of the object.

Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	SHOULD be readwrite
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	<repository specific>
Orderable:	<repository specific>

If the repository doesn't support object descriptions, the Updatability SHOULD be readonly and the repository SHOULD return a "not set" value for this property.

cmis:objectId Id of the object.

Property Type:	Id
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	TRUE
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:baseTypeId Id of the base object-type for the object.

Property Type:	Id
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	SHOULD be TRUE
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:objectTypeId Id of the object's type.

Property Type:	Id
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	SHOULD be TRUE
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:secondaryObjectTypeIds Ids of the object's secondary types.

Property Type:	Id
Inherited:	FALSE
Required:	FALSE
Cardinality:	multi
Updatability:	readwrite if secondary types are supported, readonly otherwise
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	SHOULD be TRUE
Orderable:	FALSE

If the repository does not support secondary types, the repository MUST return "not set".

cmis:createdBy User who created the object.

Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	TRUE
Orderable:	TRUE

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:creationDate DateTime when the object was created.

Property Type:	DateTime
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	TRUE
Orderable:	TRUE

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:lastModifiedBy User who last modified the object.

Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	TRUE
Orderable:	TRUE

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:lastModificationDate DateTime when the object was last modified.

Property Type:	DateTime
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	TRUE
Orderable:	TRUE

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:changeToken	Opaque token used for optimistic locking and concurrency checking. (See section 2.2.1.3 Change Tokens.)
Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	FALSE
Orderable:	FALSE

The repository MUST return this property with a non-empty value if the property filter does not exclude it. If the repository does not support change tokens, this property SHOULD not be set.

cmis:sourceId	Id of the source object of the relationship.
Property Type:	Id
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	<repository specific>
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:targetId	Id of the target object of the relationship.
Property Type:	Id
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	<repository specific>
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

2.1.7 Policy Object

A policy object represents an administrative policy that can be enforced by a repository. CMIS 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. A repository may create subtypes of this base type to support different kinds of administrative policies more specifically. If a repository does not support policy objects, the policy base object-type SHOULD NOT be returned by a `getTypeChildren` service call. This is an extension point for repositories that want to expose other capabilities via CMIS that are not supported directly in CMIS.

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. Indirectly applying policy to an object, e.g. through inheritance, is outside the scope of CMIS.

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 are 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 `applyPolicy` and the `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 `getAppliedPolicies` service may be used to obtain the policy objects that are currently applied to a controllable object.

2.1.7.1 Policy Object-Type Definition

This section describes the definition of the policy object-type's attribute values and property definitions which must be present on policy instance objects. All attributes and property definitions are listed by their id.

2.1.7.1.1 Attribute Values

The policy object-type MUST have the following attribute values.

Notes:

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

id
Value: cmis:policy

localName
Value: <repository-specific>

localNamespace
Value: <repository-specific>

queryName
Value: cmis:policy

displayName
Value: <repository-specific>

baseId
Value: cmis:policy

parentId
Value: MUST NOT be set

description
Value: <repository-specific>

creatable
Value: <repository-specific>

fileable
Value: <repository-specific>

queryable
Value: <repository-specific>

controllablePolicy
Value: <repository-specific>

controllableACL
Value: <repository-specific>

includedInSupertypeQuery
Value: <repository-specific>

fulltextIndexed
Value: <repository-specific>

typeMutability.create
Value: <repository-specific>

typeMutability.update
Value: <repository-specific>

typeMutability.delete
Value: <repository-specific>

2.1.7.1.2 Property Definitions

The policy base object-type MUST have the following property definitions, and MAY include additional property definitions. Any attributes not specified for the property definition are repository specific. For all property definitions on base types, the query name MUST be the same as the property id. The repository MUST have the following property definitions on the policy object-type:

cmis:name	Name of the object.
Property Type:	String
Inherited:	FALSE
Required:	TRUE
Cardinality:	single
Updatability:	SHOULD be readwrite
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	SHOULD be TRUE
Orderable:	SHOULD be TRUE

cmis:description	Description of the object.
Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	SHOULD be readwrite
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	<repository specific>
Orderable:	<repository specific>

If the repository doesn't support object descriptions, the Updatability SHOULD be readonly and the repository SHOULD return a "not set" value for this property.

cmis:objectId Id of the object.

Property Type:	Id
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	TRUE
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:baseTypeId Id of the base object-type for the object.

Property Type:	Id
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	SHOULD be TRUE
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:objectTypeId Id of the object's type.

Property Type:	Id
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	SHOULD be TRUE
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:secondaryObjectTypeIds Ids of the object's secondary types.

Property Type:	Id
Inherited:	FALSE
Required:	FALSE
Cardinality:	multi
Updatability:	readwrite if secondary types are supported, readonly otherwise
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	SHOULD be TRUE
Orderable:	FALSE

If the repository does not support secondary types, the repository MUST return "not set".

cmis:createdBy User who created the object.

Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	TRUE
Orderable:	TRUE

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:creationDate DateTime when the object was created.

Property Type:	DateTime
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	TRUE
Orderable:	TRUE

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:lastModifiedBy — User who last modified the object.

Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	TRUE
Orderable:	TRUE

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:lastModificationDate — DateTime when the object was last modified.

Property Type:	DateTime
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	TRUE
Orderable:	TRUE

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:changeToken — Opaque token used for optimistic locking and concurrency checking.(See section 2.2.1.3 Change Tokens.)

Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	FALSE
Orderable:	FALSE

The repository MUST return this property with a non-empty value if the property filter does not exclude it. If the repository does not support change tokens, this property SHOULD not be set.

cmis:policyText	User-friendly description of the policy.
Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	<repository specific>
Orderable:	<repository specific>

2.1.8 Item Object

The item object is an extension point for repositories that want to expose other object types via CMIS that do not fit the definition for document, folder, relationship or policy. For example an independently persistable collection of properties that was not versionable and did not have content. Another example could be a base identity object for users and groups.

A repository may create subtypes of this base type to support different kinds of generic base objects more specifically. If a repository does not support item objects, the item base object-type SHOULD NOT be returned by a `getTypeChildren` service call. Like the other CMIS objects (folder, policy and relationship), item objects are not versionable and do not have content. Item objects are manipulated with the basic CRUD operations as well as with `query` if the repository has them marked as queryable.

2.1.8.1 Item Object-Type Definition

This section describes the definition of the item object-type's attribute values and property definitions which must be present on item instance objects. All attributes and property definitions are listed by their id.

2.1.8.1.1 Attribute Values

The item object-type MUST have the following attribute values.

Notes:

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

id
Value: cmis:item

localName
Value: <repository-specific>

localNamespace
Value: <repository-specific>

queryName
Value: cmis:item

displayName
Value: <repository-specific>

baseId
Value: cmis:item

parentId
Value: MUST NOT be set

description
Value: <repository-specific>

creatable
Value: <repository-specific>

fileable
Value: <repository-specific>

queryable
Value: <repository-specific>

controllablePolicy
Value: <repository-specific>

controllableACL
Value: <repository-specific>

includedInSupertypeQuery
Value: <repository-specific>

fulltextIndexed
Value: <repository-specific>

typeMutability.create
Value: <repository-specific>

typeMutability.update
Value: <repository-specific>

typeMutability.delete
Value: <repository-specific>

2.1.8.1.2 Property Definitions

The item base object-type MUST have the following property definitions, and MAY include additional property definitions. Any attributes not specified for the property definition are repository specific. For all property definitions on base types, the query name MUST be the same as the property id. The repository MUST have the following property definitions on the item object-type:

cmis:name	Name of the object.
Property Type:	String
Inherited:	FALSE
Required:	TRUE
Cardinality:	single
Updatability:	SHOULD be readwrite
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	SHOULD be TRUE
Orderable:	SHOULD be TRUE

If the repository does not support names for items, it MAY ignore the value of this property when provided by a client. The repository MUST return a name even if the item has no name. It MAY return the object id in this case.

cmis:description Description of the object.

Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	SHOULD be readwrite
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	<repository specific>
Orderable:	<repository specific>

If the repository doesn't support object descriptions, the Updatability SHOULD be readonly and the repository SHOULD return a "not set" value for this property.

cmis:objectId Id of the object.

Property Type:	Id
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	TRUE
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:baseTypeId Id of the base object-type for the object.

Property Type:	Id
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	SHOULD be TRUE
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:objectTypeId Id of the object's type.

Property Type:	Id
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	SHOULD be TRUE
Orderable:	<repository specific>

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:secondaryObjectTypeIds Ids of the object's secondary types.

Property Type:	Id
Inherited:	FALSE
Required:	FALSE
Cardinality:	multi
Updatability:	readwrite if secondary types are supported, readonly otherwise
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	SHOULD be TRUE
Orderable:	FALSE

If the repository does not support secondary types, the repository MUST return "not set".

cmis:createdBy User who created the object.

Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	TRUE
Orderable:	TRUE

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:creationDate DateTime when the object was created.

Property Type:	DateTime
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	TRUE
Orderable:	TRUE

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:lastModifiedBy User who last modified the object.

Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	TRUE
Orderable:	TRUE

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:lastModificationDate DateTime when the object was last modified.

Property Type:	DateTime
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	TRUE
Orderable:	TRUE

The repository MUST return this property with a non-empty value if the property filter does not exclude it.

cmis:changeToken	Opaque token used for optimistic locking and concurrency checking.(See section 2.2.1.3 Change Tokens.)
Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readonly
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	FALSE
Orderable:	FALSE

The repository MUST return this property with a non-empty value if the property filter does not exclude it. If the repository does not support change tokens, this property SHOULD not be set.

2.1.9 Secondary Object-Types

A secondary type defines a set of properties that can be dynamically added to and removed from objects. That is, an object can get and lose additional properties that are not defined by its primary type during its lifetime. Multiple secondary types can be applied to the same object at the same time.

Secondary types can be simple markers without properties. Alternatively, they can contain technical information about an object. For example, a repository might analyze the content of a document, detects a photo and adds a secondary type that adds EXIF data to the document. Applications might want to attach temporary data to an object such the state of the object in a workflow. Secondary types may also change the behaviour of the repository.

The CMIS specification does not define the semantics of secondary types with the exception of secondary types for retentions and holds (see section 2.1.16 Retentions and Holds). CMIS provides a way to apply and remove secondary types to/from an object. Additionally, CMIS provides an optional ability to create, update and remove secondary types.

If a repository does not support secondary types, the secondary type base object-type `cmis:secondary` SHOULD NOT be returned by a `getTypeChildren` service call.

The base object-type does not specify any property definitions and its sole purpose is to be the root type of all other secondary object-types. Repositories MAY provide property definitions on the base type that are then inherited by other secondary object-types.

Secondary types can be applied to and removed from an object at any time. An object MAY have zero or more secondary types assigned to it. When a secondary type is applied, the object provides the properties that are defined by the secondary type. When a secondary type is removed, it loses these properties and its values.

A repository MAY not allow applying or removing certain secondary object-types to certain objects based on rules that are not determined in this specification. The repository SHOULD throw a `constraint` exception if such an operation is not allowed. Secondary object-types CAN NOT be used as primary object-types. That is, when an object is created, its object-type has to be either one of the other base types or an object-type that is derived from the other base types. Hence, a secondary object-type MUST NOT be creatable.

Whether an object is fileable, versionable or controllable is determined by its primary object-type.

2.1.9.1 Secondary Type Application

Secondary types can be applied at creation time by populating the multi-value property `cmis:secondaryObjectTypeIds` with the ids of the secondary types. All properties defined by these secondary types can be set as well.

Secondary types can be added and removed later by changing the `cmis:secondaryObjectTypeIds` property, either through the `updateProperties` service or the `checkIn` service. Adding the id of a secondary type to this multi value property adds the secondary type. Removing the id of a secondary type from this multi value property removes the type and all associated properties and values.

A repository MUST throw a `constraint` exception if a secondary type cannot be added or removed.

Adding a secondary type and providing values for the associated properties of this secondary type MAY be done in the same operation.

2.1.9.2 Secondary Object-Type Definition

This section describes the definition of the secondary object-type's attribute values. All attributes are listed by their id.

2.1.9.2.1 Attribute Values

The secondary object-type MUST have the following attribute values.

Notes:

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

id
Value: cmis:secondary

localName
Value: <repository-specific>

localNamespace
Value: <repository-specific>

queryName
Value: cmis:secondary

displayName
Value: <repository-specific>

baseId
Value: cmis:secondary

parentId
Value: MUST NOT be set

description
Value: <repository-specific>

creatable
Value: FALSE

fileable
Value: FALSE

queryable
Value: <repository-specific>

controllablePolicy
Value: FALSE

controllableACL
Value: FALSE

includedInSupertypeQuery
Value: <repository-specific>

fulltextIndexed
Value: <repository-specific>

Note: This attribute defines if the properties of this secondary type are full-text indexed. It does not make a statement about the content.

2.1.9.2.2 Property Definitions

The secondary base object-type has no properties. Repositories MAY provide custom property definitions.

2.1.10 Object-Type Creation, Modification and Deletion

A repository MAY support the creation, modification and deletion of primary and secondary object-types.

Each object-type definition SHOULD include a set of flags that indicate if the object-type can be used as a parent type or if the object-type can be modified or deleted. Please see section 2.1.3.2.1 Attributes common to ALL Object-Type Definitions for details.

These flags are not to be interpreted as the rights for the current user. These are the rights that would apply to an administrator user or a user that has sufficient rights to modify metadata. For example, a non-administrator would see that an object-type is extendable (the type mutability capabilities create flag is set to TRUE) even though they would not be allowed to actually perform the operation. If a user tries to create, modify or delete a type definition and does not have the required permissions, the repository MUST return a `permissionDenied` error.

A repository MAY also place additional restrictions on these operations where necessary. These restrictions are repository specific.

2.1.10.1 General Constraints on Metadata Changes

The optional capabilities `capabilityNewTypeSettableAttributes` and `capabilityCreatablePropertyTypes` SHOULD indicate which object-type attributes can be set by a client and which properties data types can be used to create or extend an object-type.

Note, that the client CANNOT define whether a new object-type can be used as a parent type, or can be updated or deleted. How the repository determines a given object-type's mutability capabilities is repository specific.

When an object-type is created the client MUST suggest a type id for the new object-type. The repository may do the following with this suggested value:

- Use it exactly as specified.
  e.g. input = invoice : returned value = invoice
- Modify it with the addition of a prefix, suffix or both.
  e.g. input = invoice : returned value = invoice_FAF5D0C5-BBE9
- Return a completely different value.
  e.g. input = invoice : returned value = FAF5D0C5-BBE9-4E47-BB17-C9FE63B4EE20

When a property definition is created the client MUST suggest a property definition id for the new property. The repository may do the following with this suggested value:

- Use it exactly as specified.
  e.g. input = amount : returned value = amount
- Modify it with the addition of a prefix, suffix or both.
  e.g. input = amount: returned value = amount_12AB
- Return a completely different value.
  e.g. input = amount: returned value = 12AB-23CD

When an object-type is created or updated, the repository MUST return the created or updated type definition whereby the order of ALL newly created property definitions MUST match the order of the input. This is so that there will be no ambiguity for clients who need to know which property matches a specific suggested Id value for a new property definition. This special ordering is only required for the return value for `createType` and `updateType`. There is no special ordering of the properties returned for subsequent calls to `getTypeDefinition` for this new or modified type.

When an object-type is updated the following rules MUST be obeyed:

- Inherited properties MUST NOT be modified. This includes constraints of any kind.

- Properties defined by the CMIS specification MUST NOT be modified. This includes constraints of any kind.
- Only leaf types may be modified. That is, if a type already has child types defined then it (and all of its properties and constraints) MUST be considered read only.
- Any added properties marked as "required" MUST have a default value.
- Required properties MAY be changed to optional.
- Optional properties MUST NOT be changed to required.
- Property definitions MUST NOT be removed.
- Property choice constraints MAY be changed in the following manner:
  - 'Open choice' MAY change from FALSE to TRUE.
  - 'Open choice' MUST NOT change from TRUE to FALSE.
  - Choices MAY be added or removed if 'open choice' is TRUE.
  - Choices MUST NOT be removed if 'open choice' is FALSE.
- Validation constraints (min/max length, min/max value, etc.) on existing properties MAY be relaxed but they MUST NOT be further restricted. For example, an integer property value that originally had a minimum constraint of 100 and a maximum constraint of 1000 could change as follows:
  - A new minimum could be changed to 50 but could not be changed to 150.
  - A new maximum could be changed to 1100 but could not be changed to 900.

  This ensures that the new constraints will not leave any existing data out of the permitted constraint range.
- An existing property type's data type and cardinality MUST NOT be changed. For example, an Integer property type MUST NOT be changed to a String.

The execution of the `createType` and `updateType` services MUST not affect the definition of any other types or any other type's current property definitions. For example, any properties on the type being created must not place constraints on other type's properties when/if other properties 'share' property definitions.

An object-type can only be deleted if there are no objects of this type and the object-type has no sub-types. The `deleteType` service MUST return a `constraint` error if an instance of the object-type exists or the object-type is a parent type of another object-type.

2.1.11 Object-Type Summary

The following diagrams illustrate the CMIS object model. Please note that they only reflect the logical model. The CMIS bindings use slightly different data structures.

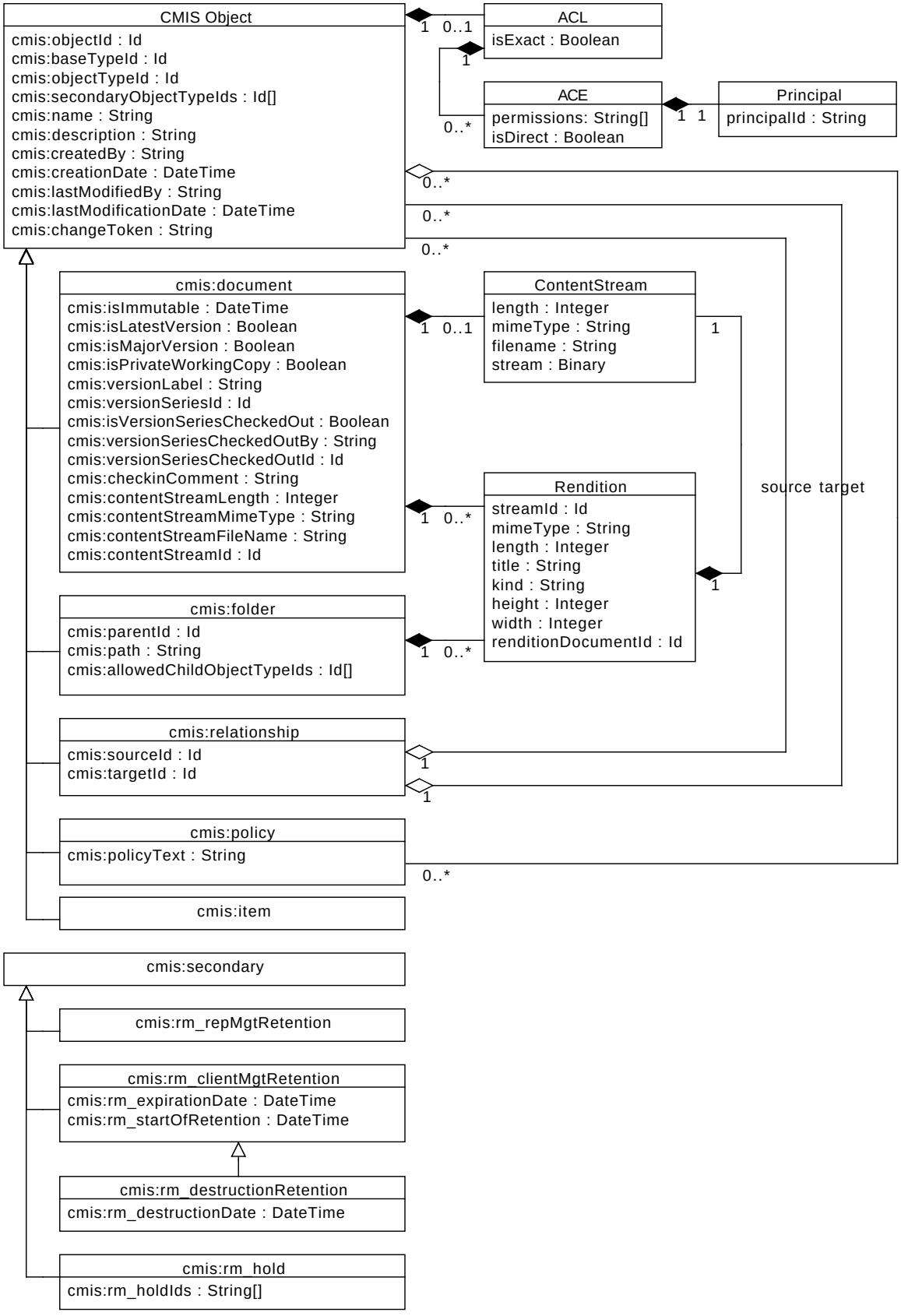

Figure 2.2: CMIS Model

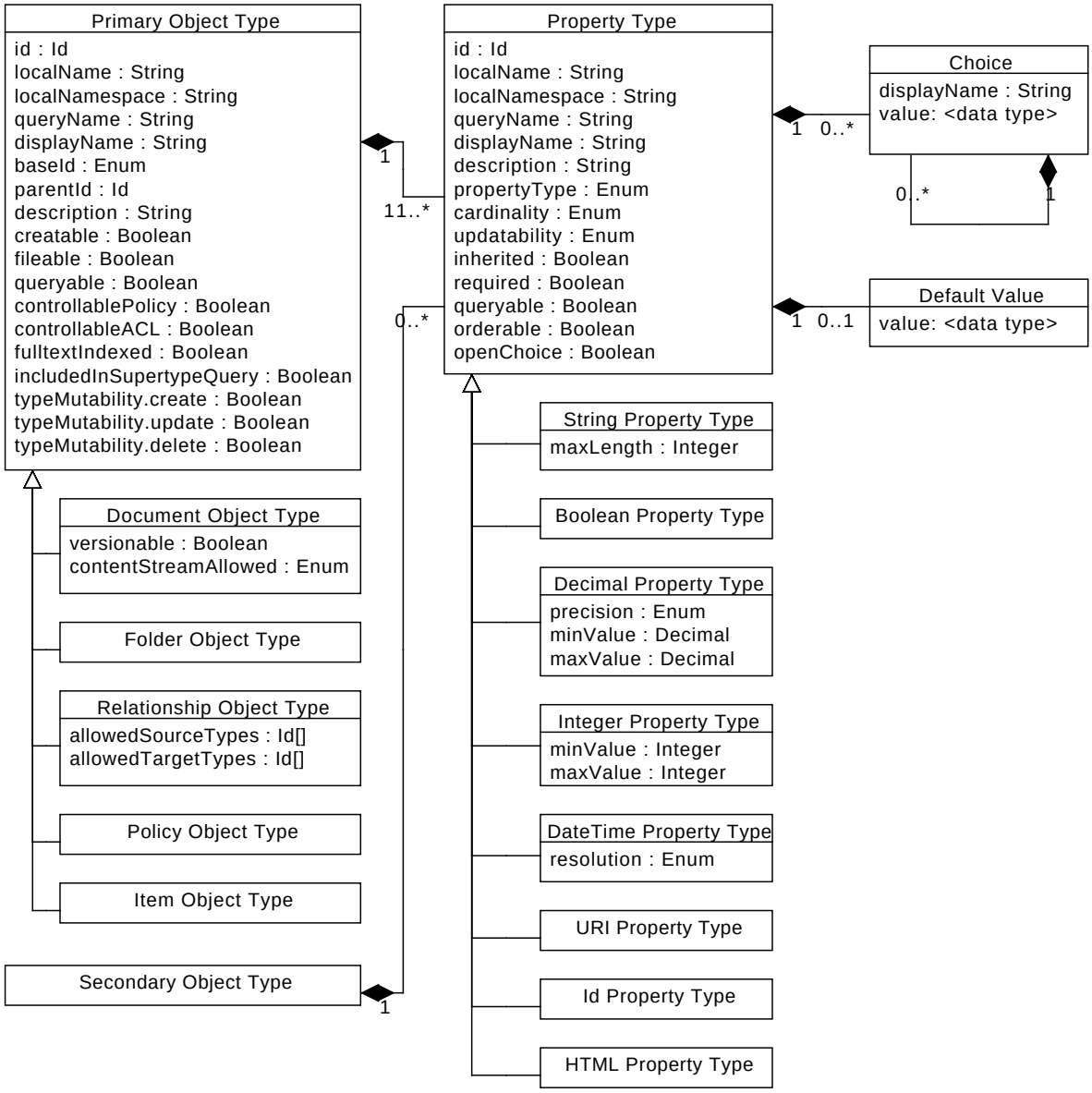

Figure 2.3: CMIS Object Types and Property Types

2.1.12 Access Control

A repository can support either a base set of CMIS-defined permissions and/or its own set of repository specific permissions.

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

2.1.12.1 ACL, ACE, Principal, and Permission

An `Access Control List (ACL)` is a list of `Access Control Entries (ACEs)` and MAY hold zero or more ACEs. If an ACL has no ACEs, the behavior is the same as if the ACL is not set.

An ACE holds:

- A *principal* that represents a user management object, e.g. a user, group, or role. It holds one string with the *principalId*.
- One or more strings with the names of the *permissions*.
- A boolean flag *`direct`* which indicates if TRUE that the ACE is directly assigned to the object. If FALSE, that the ACE is somehow derived or inherited.

2.1.12.2 CMIS Permissions

There are three basic permissions predefined by CMIS:

cmis:read Expresses the "permission to read" properties AND content of an object.

cmis:write Expresses the "permission to write" properties AND content of an object. It MAY include the `cmis:read` permission.

cmis:all SHOULD express all the permissions of a repository. It SHOULD include all other basic CMIS permissions.

How these basic permissions are mapped to the allowable actions is repository specific. However, the actual repository semantics for the basic permissions with regard to allowable actions can be discovered by the mappings parameter returned by the `getRepositoryInfo` service.

Repositories MAY extend this set with repository-specific permissions.

2.1.12.3 ACL Capabilities

Whether a repository supports ACLs at all, may be discovered via `capabilityACL` attribute returned by the `getRepositoryInfo` service (see section 2.1.1.1 Optional Capabilities). If the value of the `capabilityACL` attribute is `none`, ACLs are not supported by the repository.

If the value of the `capabilityACL` attribute is `discover` or `manage`, additional information about the repository's permission model and how ACL modifications are handled are provided by the `getRepositoryInfo` service:

Enum propagation specifies how non-direct ACEs can be handled by the repository using the following values (see section 2.2.10.1 applyACL):

objectonly indicates that the repository is able to apply ACEs to an object without changing the ACLs of other objects.

propagate indicates that the ACEs might be inherited by other objects. `propagate` includes the support for `objectonly`.

repositorydetermined indicates that the repository has its own mechanism of computing how changing an ACL for an object influences the non-direct ACEs of other objects.

<Array> PermissionDefinition repositoryPermissions A list of names and descriptions of the supported permissions.

<Array> PermissionMapping mappings Contains a list of basic CMIS permissions to allowable actions mapping.

2.1.12.3.1 Supported Permissions

The list of permission definitions returned by the `getRepositoryInfo` service lists all the permissions a repository supports. This list also includes the CMIS permissions if supported by the repository.

A PermissionDefinition holds:

String permission The (technical) name of the permission. Permission names MUST be unique within the permission definition list.

String description An optional description of the permission that SHOULD be used as the permission's name to be presented to the user.

2.1.12.3.2 AllowableActions and Permission Mapping

CMIS provides a mechanism called *Allowable Actions* which allows an application to discover the set of service operations that can currently be performed on a particular object by the current user, without having to actually invoke the service.

The set of allowable actions on an object at a point in time are affected not only by CMIS ACLs, but also by other factors such as:

- Constraints inherent in the CMIS Domain Model based on the object's base type or current versioning state.
- Policies or other control mechanisms that are opaque to CMIS.

CMIS defines several services that applications can use at run-time to discover the allowable actions for an object.

If a repository supports ACLs, then the repository MUST provide a mapping table that defines how the permissions supported by the repository interact with the CMIS allowable actions, i.e. which permissions are necessary for a principal to have on one or more objects in order to potentially perform each action, subject to the other constraints on allowable actions mentioned above.

This section defines both the allowable actions as well as how those actions are presented in the permission mapping table.

The permission mapping table contains a set of *key--permissions* pairs:

String key Since several allowable actions require permissions on more than one object, the mapping table is defined in terms of permission "keys". (For example, moving a document from one folder to another may require permissions on the document and each of the folders.) Each key combines the name of the allowable action and the object for which the principal needs the required permission.
For example, the `canMoveObject.Source` key indicates the permissions that the principal must have on the "source folder" to move an object from that folder into another folder.

<Array> String permissions The name of one or more permissions that the principal MUST have. If more than one permission is specified, then the principal MUST be allowed to perform the operation if they have ANY of the listed permissions.

The following list defines all mapping keys, as well as a permissions mapping that repositories SHOULD use. Repositories MAY require additional permissions.

For convenience, the list groups all mapping entries by the underlying allowable actions, and includes descriptive information. For each allowable action the following information is given:

Description The description and name of the service the allowable action enables.

Base Type The base object-types for which the allowable action MAY be TRUE.

Operand The object the permission applies to.

Key The permission mapping key.

Permissions The permission values.

2.1.12.3.2.1 Navigation Services

canGetDescendants

Description:	Can get the descendants of the folder (`getDescendants` and `getFolderTree`).
Base Type:	cmis:folder
Operand:	Folder
Key:	canGetDescendants.Folder
Permission:	cmis:read

canGetChildren

Description:	Can get the children of the folder (`getChildren`).
Base Type:	cmis:folder
Operand:	Folder
Key:	canGetChildren.Folder
Permission:	cmis:read

canGetFolderParent

Description:	Can get the parent folder of the folder (`getFolderParent`).
Base Type:	cmis:folder
Operand:	Folder
Key:	canGetFolderParent.Object
Permission:	cmis:read

canGetObjectParents

Description:	Can get the parent folders of the object (`getObjectParents`).
Base Type:	cmis:document, cmis:policy, cmis:item
Operand:	Object
Key:	canGetParents.Folder
Permission:	cmis:read

2.1.12.3.2.2 Object Services

canCreateDocument

Description:	Can create a cmis:document object in the specified folder (`createDocument`).
Base Type:	cmis:folder
Operand:	Folder
Key:	canCreateDocument.Folder
Permission:	cmis:read

canCreateFolder

Description:	Can create a cmis:folder object as a child of the specified folder (`createFolder`).
Base Type:	cmis:folder
Operand:	Folder
Key:	canCreateFolder.Folder
Permission:	cmis:read

canCreatePolicy

Description:	Can create a cmis:policy object as a child of the specified folder (`createPolicy`).
Base Type:	cmis:folder
Operand:	Folder
Key:	canCreatePolicy.Folder
Permission:	cmis:read

canCreateRelationship

Description:	Can create a relationship object with the object as its source (`createRelationship`).
Base Type:	cmis:document, cmis:folder, cmis:policy, cmis:item
Operand:	Object
Key:	canCreateRelationship.Source
Permission:	cmis:read

canCreateRelationship

Description:	Can create a relationship object with the object as its target (`createRelationship`).
Base Type:	cmis:document, cmis:folder, cmis:policy, cmis:item
Operand:	Object
Key:	canCreateRelationship.Target
Permission:	cmis:read

canGetProperties

Description:	Can read the properties of the specified object (`getProperties`, `getObject` and `getObjectByPath`).
Base Type:	cmis:document, cmis:folder, cmis:relationship, cmis:policy, cmis:item
Operand:	Object
Key:	canGetProperties.Object
Permission:	cmis:read

canUpdateProperties

Description:	Can update the properties of the specified object (`updateProperties`).
Base Type:	cmis:document, cmis:folder, cmis:relationship, cmis:policy, cmis:item
Operand:	Object
Key:	canUpdateProperties.Object
Permission:	cmis:write

canMoveObject

Description:	Can move the specified object (`moveObject`).
Base Type:	cmis:document, cmis:folder, cmis:policy, cmis:item
Operand:	Object
Key:	canMove.Object
Permission:	cmis:write

canMoveObject

Description:	Can move an object into this folder (`moveObject`).
Base Type:	cmis:folder
Operand:	Folder
Key:	canMove.Target
Permission:	cmis:read

canMoveObject

Description:	Can move an object from this folder (`moveObject`).
Base Type:	cmis:folder
Operand:	Folder
Key:	canMove.Source
Permission:	cmis:read

canDeleteObject

Description: Can delete the specified object (deleteObject).
Base Type: cmis:document, cmis:folder, cmis:relationship, cmis:policy, cmis:item
Operand: Object
Key: canDelete.Object
Permission: cmis:write

canGetContentStream

Description: Can get the content stream for the document object (getContentStream).
Base Type: cmis:document
Operand: Object
Key: canViewContent.Object
Permission: cmis:write

canSetContentStream

Description: Can set the content stream for the document object (setContentStream).
Base Type: cmis:document
Operand: Object
Key: canSetContent.Document
Permission: cmis:write

canDeleteContentStream

Description: Can delete the content stream for the Document object (deleteContentStream).
Base Type: cmis:document
Operand: Object
Key: canDeleteContent.Document
Permission: cmis:write

canDeleteTree

Description: Can delete the specified folder and all contained objects (deleteTree).
Base Type: cmis:folder
Operand: Object
Key: canDeleteTree.Folder
Permission: cmis:write

2.1.12.3.2.3 Filing Services

canAddObjectToFolder

Description: Can file the object in a folder (addObjectToFolder).
Base Type: cmis:document, cmis:policy, cmis:item
Operand: Object
Key: canAddToFolder.Object
Permission: cmis:read

canAddObjectToFolder

Description: Can file an object in the specified folder (addObjectToFolder).
Base Type: cmis:document, cmis:policy, cmis:item
Operand: Folder
Key: canAddToFolder.Folder
Permission: cmis:read

canRemoveObjectFromFolder

Description: Can unfile the specified document from a folder (removeObjectFromFolder).
Base Type: cmis:document, cmis:policy, cmis:item
Operand: Object
Key: canRemoveFromFolder.Object
Permission: cmis:read

canRemoveObjectFromFolder

Description: Can unfile an object from the specified folder (removeObjectFromFolder).
Base Type: cmis:document, cmis:policy
Operand: Folder
Key: canRemoveFromFolder.Folder
Permission: cmis:read

2.1.12.3.2.4 Versioning Services

canCheckOut

Description: Can check out the specified document (checkOut).
Base Type: cmis:document
Operand: Object
Key: canCheckout.Document
Permission: cmis:write

canCancelCheckOut

Description: Can cancel the check out the specified PWC (`cancelCheckOut`).
Base Type: cmis:document
Operand: Object
Key: canCancelCheckout.Document
Permission: cmis:write

canCheckIn

Description: Can check in the specified PWC (`checkIn`).
Base Type: cmis:document
Operand: Object
Key: canCheckin.Document
Permission: cmis:write

canGetAllVersions

Description: Can get the version series of the specified document (`getAllVersions`).
Base Type: cmis:document
Operand: Object
Key: canGetAllVersions.VersionSeries
Permission: cmis:read

2.1.12.3.2.5 Relationship Services

canGetObjectRelationships

Description: Can get the relationship in which this object is a source or a target (`getObjectRelationships`).
Base Type: cmis:document, cmis:folder, cmis:policy, cmis:item
Operand: Object
Key: canGetObjectRelationships.Object
Permission: cmis:read

2.1.12.3.2.6 Policy Services

canApplyPolicy

Description: Can apply a policy to the specified object (`applyPolicy`).
Base Type: cmis:document, cmis:folder, cmis:policy, cmis:relationship, cmis:item
Operand: Object
Key: canAddPolicy.Object
Permission: cmis:read

canApplyPolicy

Description:	Can apply the specified policy to an object (`applyPolicy`).
Base Type:	cmis:policy
Operand:	Object
Key:	canAddPolicy.Policy
Permission:	cmis:read

canRemovePolicy

Description:	Can remove a policy from the specified object (`removePolicy`).
Base Type:	cmis:document, cmis:folder, cmis:policy, cmis:relationship, cmis:item
Operand:	Object
Key:	canRemovePolicy.Object
Permission:	cmis:read

canRemovePolicy

Description:	Can remove the specified policy from an object (`removePolicy`).
Base Type:	cmis:policy
Operand:	Object
Key:	canRemovePolicy.Policy
Permission:	cmis:read

canGetAppliedPolicies

Description:	Can get the list of policies applied to the specified object (`getAppliedPolicies`).
Base Type:	cmis:document, cmis:folder, cmis:policy, cmis:relationship, cmis:item
Operand:	Object
Key:	canGetAppliedPolicies.Object
Permission:	cmis:read

2.1.12.3.2.7 ACL Services

canGetACL

Description:	Can get ACL of the specified object (`getACL`).
Base Type:	cmis:document, cmis:folder, cmis:relationship, cmis:policy, cmis:item
Operand:	Object
Key:	canGetACL.Object
Permission:	cmis:read

canApplyACL

Description:	Can apply ACL to this object (`applyACL`).
Base Type:	cmis:document, cmis:folder, cmis:relationship, cmis:policy, cmis:item
Operand:	Object
Key:	canApplyACL.Object
Permission:	cmis:write

2.1.13 Versioning

CMIS supports versioning of document objects. Folder objects, relationship objects, policy objects, and item 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 MUST 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 object id, property values, MAY be acted upon using all CMIS services that act upon document objects, etc.

2.1.13.1 Version Series

A *version series* for a document object is a transitively closed collection of all document objects, other than any Private Working Copy (see section 2.1.13.5.1 Checkout), 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 creation date properties (`cmis:creationDate`).

Additionally, the repository MAY expose a textual *version label* (`cmis: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.

2.1.13.2 Latest Version

The version that has the most recent last modification date (`cmis: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.

2.1.13.3 Behavioral constraints on non-Latest Versions

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

2.1.13.4 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 last modification date (property `cmis:lastModificationDate`) is the *latest major version* of the version series.

(Note that while a version series MUST 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.

2.1.13.5 Services that modify Version Series

2.1.13.5.1 Checkout

A new version of a versionable document object is created when the checkIn service is invoked on the *Private Working Copy (PWC)* of this object. A PWC is created by invoking checkOut on a versionable 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 MUST be as follows:

- A new document object, referred to herein as the *Private Working Copy (PWC)*, is created. The object id of this new document object MUST be unique and MUST NOT be equal to the id of the object on which the checkOut service was invoked.
- The PWC NEED NOT be visible to users who have permissions to view other document objects in the version series.
- The value of the cmis:isPrivateWorkingCopy property MUST be TRUE.
- The PWC is NOT to be considered a version in the version series but inherits the version series id from the document it was created from.
- Therefore, until it is checked in (using the checkIn service), the PWC MUST NOT be considered the latest or latest major version in the version series. That is, the values of the cmis:isLatestVersion and cmis:isLatestMajorVersion properties MUST be FALSE.
- The property values for the PWC SHOULD be identical to the properties of the document object on which the checkOut service was invoked. Certain properties may be different. Properties such as cmis:creationDate most likely will be different. The content stream of the PWC MAY be identical to the content stream of the document object on which the checkOut service was invoked, or MAY be "not set".

After a successful checkOut operation is completed, and until such time when the PWC is deleted (via the cancelCheckOut service) or checked-in (via the checkIn service), the effects on the PWC or on other documents in the version series MUST be as follows:

- The repository MUST 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 cmis:isVersionSeriesCheckedOut property MUST be TRUE.
- The value of the cmis:versionSeriesCheckedOutBy property SHOULD be set to a value indicating which user created the PWC. (The repository MAY still show the "not set" value for this property if, for example, the information is not available or the current user has not sufficient permissions.)
- The value of the cmis:versionSeriesCheckedOutId property SHOULD be set to the object id of the PWC. (The repository MAY still show the "not set" value for this property if the current user has no permissions to see the PWC).
- The repository MAY prevent operations that modify or delete the other documents in the version series.

2.1.13.5.2 Updates to the Private Working Copy

If the repository supports the optional "PWCUpdatable" capability, then the repository MUST allow authorized users to modify the PWC object using the object services (e.g. updateProperties and setContentStream).

If the repository does NOT support the "PWCUpdatable" capability, then the PWC object can only be modified as part of the checkIn service call.

2.1.13.5.3 Discarding Check out

An authorized user MAY discard the check-out using the `cancelCheckOut` service on the PWC object or by using the `deleteObject` service on the PWC object. The effects of discarding a check-out MUST be as follows:

- The PWC Object MUST be deleted.
- For all other documents in the version series:
  - The value of the `cmis:isVersionSeriesCheckedOut` property MUST be FALSE.
  - The value of the `cmis:versionSeriesCheckedOutBy` property MUST be "not set".
  - The value of the `cmis:versionSeriesCheckedOutId` property MUST be "not set".
  - The repository MUST allow authorized users to invoke the `checkOut` service.

2.1.13.5.4 Checkin

An authorized user MAY "check in" the Private Working Copy object via the `checkIn` service.

The `checkIn` service allows users to provide update property values and a content stream for the PWC object.

The effects of the `checkIn` service MUST be as follows for successful checkins:

- The PWC object MUST be updated as specified by the inputs to the `checkIn` service. (Note that for repositories that do NOT support the "PWCUpdatable" property, this is the only way to update the PWC object.)
- The document object resulting from the `checkIn` service MUST be considered the latest version in the version series.
- If the inputs to the checkIn service specified that the PWC MUST be a "major version", then the newly created version MUST be considered the latest major version in the version series.
- If the check-in returns a new `cmis:objectId`, then the PWC object MUST disappear if the `checkIn` call was successful and the new checked in version will use the new specified id.
- For all documents in the version series:
  - The value of the `cmis:isVersionSeriesCheckedOut` property MUST be FALSE.
  - The value of the `cmis:versionSeriesCheckedOutBy` property MUST be "not set".
  - The value of the `cmis:versionSeriesCheckedOutId` property MUST be "not set".
  - The repository MUST allow authorized users to invoke the `checkOut` service.

Note: A repository MAY automatically create new versions of document objects without an explicit invocation of the `checkOut`/`checkIn` services.

2.1.13.6 Versioning Properties on Document Objects

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

`cmis:isPrivateWorkingCopy` Boolean

TRUE if the document object is a Private Working Copy. FALSE otherwise.

cmis:isLatestVersion Boolean

TRUE if the document object is the latest version (most recent last modification date) in its version series. FALSE otherwise. MUST be FALSE for Private Working Copy objects.

cmis:isMajorVersion Boolean

TRUE if the document object is a major version in its version series. FALSE otherwise. MUST be FALSE for Private Working Copy objects.

cmis:isLatestMajorVersion Boolean

TRUE if the document object is the latest major version in its version series. FALSE otherwise. MUST be FALSE for Private Working Copy objects.

cmis:versionLabel String

Textual description the position of an individual object with respect to the version series. (For example, "version 1.0"). MAY be "not set".

cmis:versionSeriesId Id

Id of the version series for this object.

cmis:isVersionSeriesCheckedOut Boolean

TRUE if there currenly exists a Private Working Copy for this version series. FALSE otherwise.

cmis:versionSeriesCheckedOutBy String

If `cmis:isVersionSeriesCheckedOut` is TRUE: An identifier for the user who created the Private Working Copy. "Not set" otherwise.

cmis:versionSeriesCheckedOutId String

If `cmis:isVersionSeriesCheckedOut` is TRUE: The object id for the Private Working Copy. "Not set" otherwise.

cmis:checkinComment String

Textual comment associated with the given version. MAY be "not set".

Note: Changes made via the Versioning Services that affect the values of these properties MUST NOT constitute modifications to the document objects in the version series (e.g. MUST NOT affect the `cmis:lastModificationDate`, etc.).

2.1.13.7 Document Creation and Initial Versioning State

When calling the `createDocument` service or the `createDocumentFromSource` service, a `versioningState` parameter can be used to specify what the versioning state of the newly-created object MUST be.

A repository MAY create new document objects in a "Private Working Copy" state. 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.

If the repository does not support versioning the repository MUST ignore the value of the `versioningState` parameter.

2.1.13.8 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 MUST indicate whether they support version-specific membership in a folder via the "capabilityVersionSpecificFiling" optional capability flag. (See section 2.1.1.1 Optional Capabilities.)

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

- Moving or filing a document object into a folder MUST 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 MUST NOT result in other documents in the version series being moved/filed.

2.1.13.9 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 `getObjectRelationships` service invoked on a document object MUST 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 `getObjectRelationships` service invoked on a document object MUST return the relationship if relationship was source/target is set to the id of the document object on which the service was invoked.

2.1.13.10 Versioning visibility in Query Services

Repositories MAY include non-latest-versions of document objects in results to the `query` service.

Repositories MUST indicate whether they support querying for non-latest-versions via the "capabilityAllVersionsSearchable" optional capability flag. (See section 2.1.1.1 Optional Capabilities.)

If "capabilityAllVersionsSearchable" is TRUE then the repository MUST 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.)

Additionally, repositories MAY include Private Working Copy objects in results to the `query` service. Repositories MUST indicate whether they support querying for Private Working Copy objects via the "capabilityPWCSearchable" optional capability flag.

If "capabilityPWCSearchable" is TRUE then the repository MUST 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 "capabilityPWCSearchable" is FALSE then the repository MUST 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.)

2.1.14 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 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 this query language is defined by the SQL-92 standard, plus the extensions, in conjunction with the model mapping defined by CMIS's relational view.

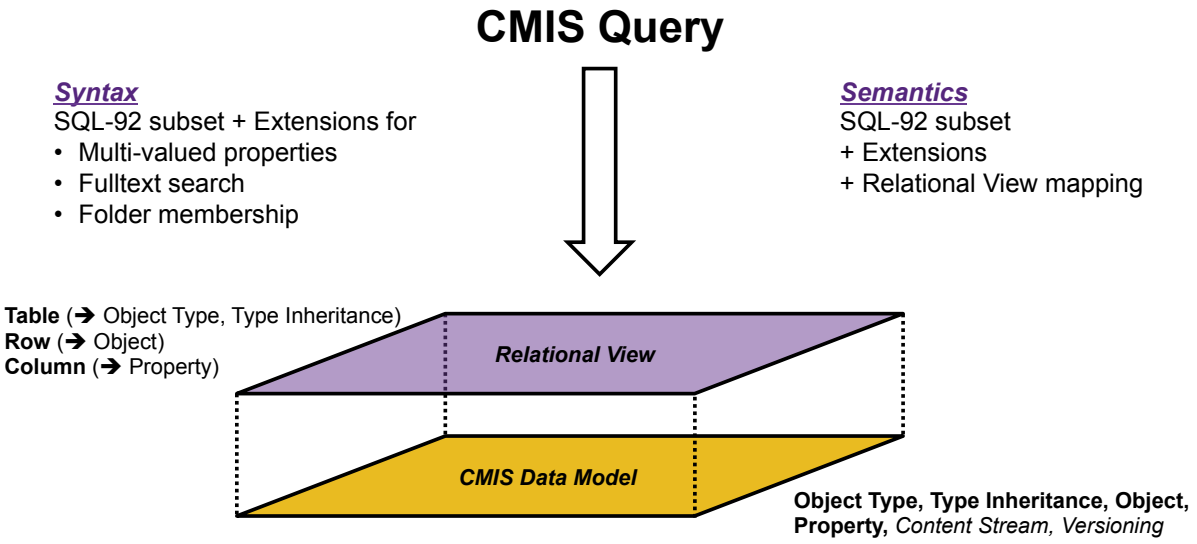

Figure 2.4: CMIS Query

2.1.14.1 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 MUST contain the SQL NULL value. Virtual columns for properties whose value is "not set" MUST contain the SQL NULL value.

An object-type's *queryName* attribute is used as the table name for the corresponding virtual table, and a property's *queryName* attribute is used as the column name for the corresponding table column. Please see the restrictions on queryName in section 2.1.2.1.3 Query Names.

The virtual column for a multi-valued property MUST contain a single list value that includes all values of the property.

2.1.14.1.1 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 MUST be included in the virtual table for any of its ancestor types. If the `includedInSuper-typeQuery` attribute of the object-type is FALSE, then objects of that object-type MUST NOT be included in the virtual table for any of its ancestor types.

In each virtual table, a virtual column is implicitly defined for each property defined in the object-type definition. In addition, a virtual column is also implicitly defined for each property defined on ANY ancestor-type of this object-type but NOT defined in this object-type definition. In addition, the virtual table for a secondary object type has one more virtual column for the `cmis:objectId` property defined by each object's primary type. If a secondary object type does not define any property, then its virtual table will have `cmis:objectId` as the only column, identifying the objects to which the secondary type has been applied. Virtual columns for properties defined on ancestor-types of the object-type but NOT defined (inherited) in the object-type definition MUST contain the SQL NULL value. Virtual columns for properties whose value is "not set" MUST contain the SQL NULL value. The rows of a virtual table corresponding to a queryable primary type represent the objects of that type. The rows of a virtual table corresponding to a queryable secondary type represent objects of various primary types (which may or may not be queryable) that the secondary type is applied to. To query on both an object's primary type properties and its secondary type properties, a SQL JOIN of the respective tables on the `cmis:objectId` column may be performed. Explicit JOIN support, as defined in 2.1.1.1 Optional Capabilities, is not required for a repository to provide join between a primary type and secondary type tables based on `cmis:objectId`.

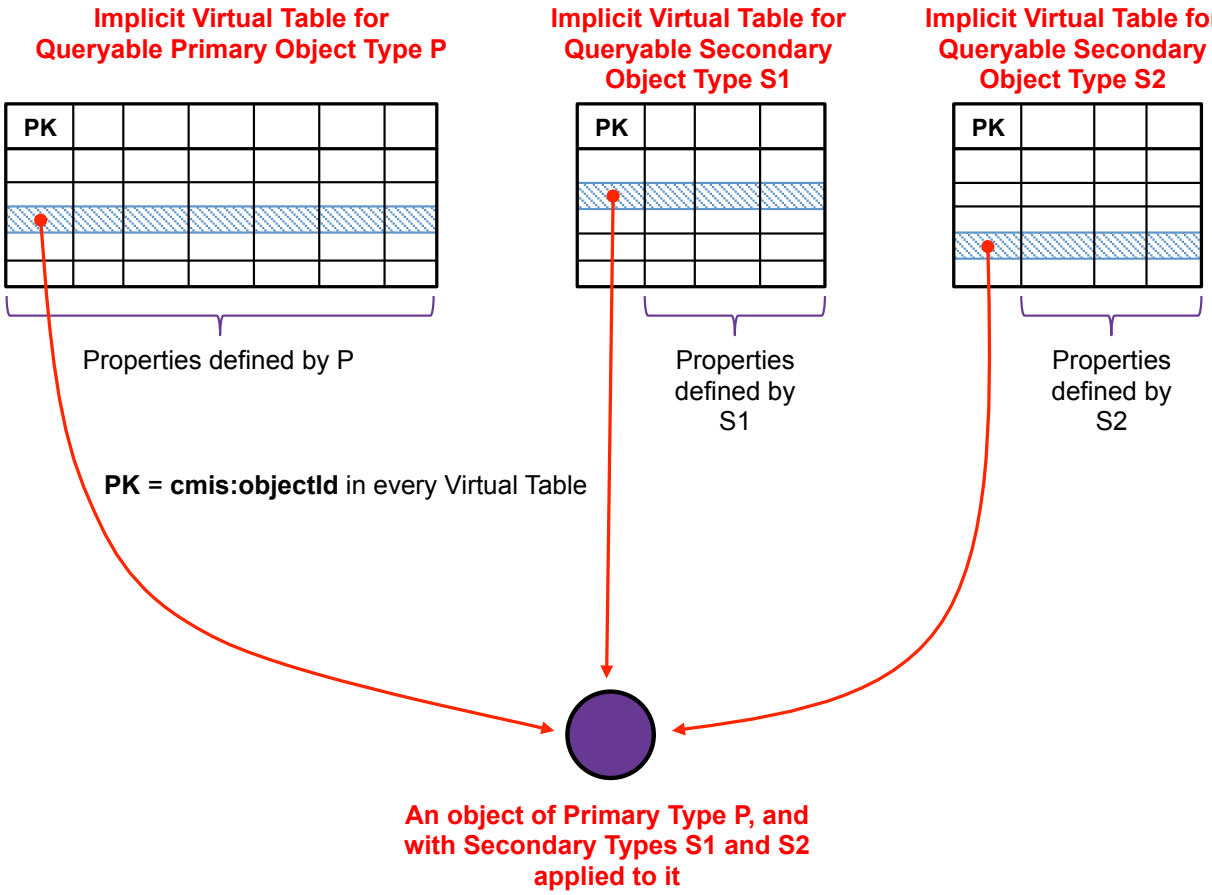

Figure 2.5: Virtual Tables

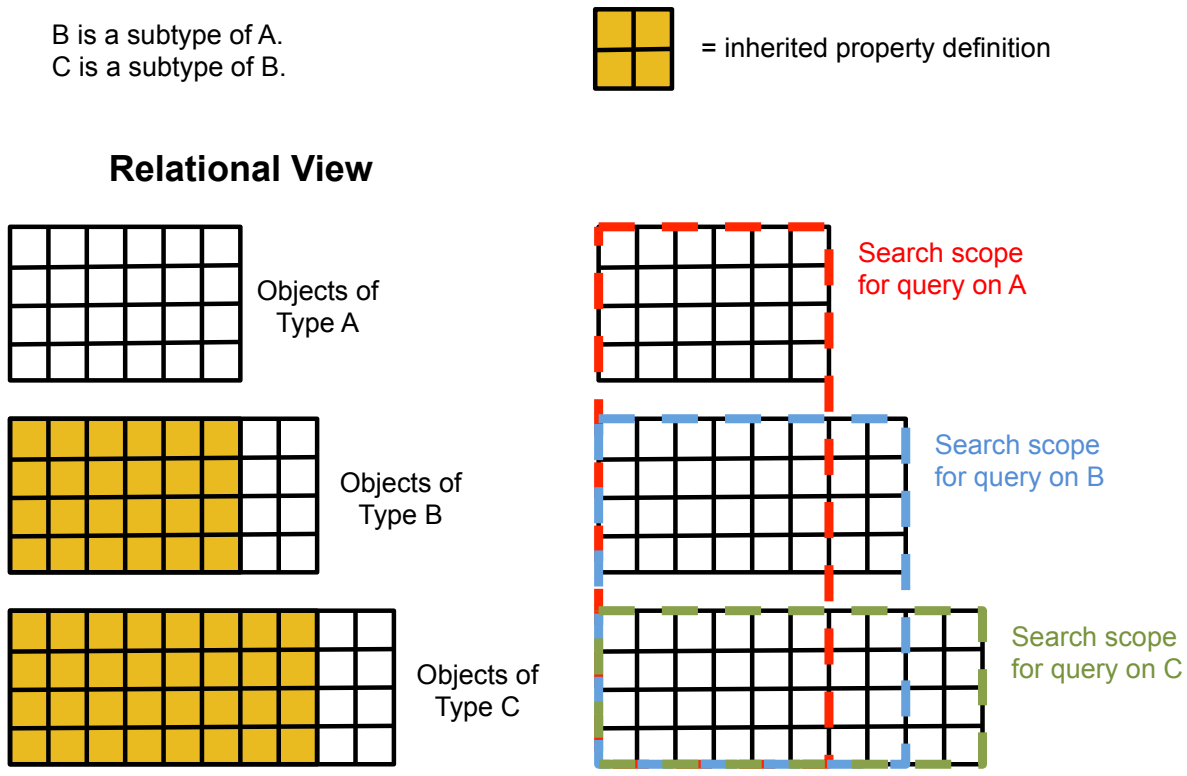

Figure 2.6: Query Search Scope

2.1.14.1.2 Content Streams

Content streams are NOT exposed through this relational view.

2.1.14.1.3 Result Set

When a query is submitted, a set of pseudo CMIS objects will be returned. These pseudo objects are comprised of the properties specified in the select clause of the query statement.

For each property in each object in the result set, the repository MUST include the property definition id as well as either the query name (if no alias is used) or the alias in place of the query name (if an alias is used).

If the select clause of the query statement contains properties from a single type reference then the repository MAY represent these pseudo-objects with additional object information.

2.1.14.2 Query Language Definition

This query languages 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 query is a SQL statement that MUST include the following clauses:

SELECT [virtual columns list] This clause identifies the set of virtual columns that will be included in the query results for each row and optionally their aliases.

FROM [virtual table names] This clause identifies which virtual table(s) the query will run against. Aliases for the object-types are allowed in the BNF grammar.

Additionally, a CMIS query MAY include the following clauses:

WHERE [conditions] This clause identifies the constraints that rows MUST satisfy to be considered a result for the query.

ORDER BY [sort specification] This clause identifies the order in which the result rows MUST be sorted in the result row set.

2.1.14.2.1 BNF Grammar

This BNF grammar is a "subset" of the SQL-92 grammar (ISO/IEC 9075: 1992 – Database Language SQL), except for some production alternatives. Specifically, except for these extensions, the following production rules are derived from the SQL-92 grammar. 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 this query language to cover a larger subset of SQL with minimum conflict. The CMIS extensions are introduced primarily to support multi-valued properties and full-text search, and to test folder membership. Multi-valued properties are handled separately from single-valued properties, using separate non-terminals and separate production rules to prevent the extensions from corrupting SQL-92 semantics.

```
<CMIS 1.1 query statement> ::= <simple table> [ <order by clause> ]
<simple table>   ::= SELECT <select list> <from clause> [ <where clause> ]
<select list>    ::= "*" | <select sublist> [ { "," <select sublist> }... ]
<select sublist> ::= <qualifier> ".*"
  | <value expression> [ [ AS ] <column name> ]
  | <multi-valued-column reference> [ [ AS ] <column name> ]
<value expression> ::= <column reference> | <numeric value function>
<column reference> ::= [ <qualifier> "." ] <column name>
  | [ <qualifier> "." ] <secondary type table name> "." <secondary type column name>
<multi-valued-column reference> ::= [ <qualifier> "." ] <multi-valued-column name>
  | [ <qualifier> "." ] <secondary type table name> "." <secondary type multi-valued-column name>
<numeric value function> ::= SCORE()
<qualifier>   ::= <table name> | <correlation name>
<from clause> ::= FROM <table reference>
<table reference> ::= <table name> [ [ AS ] <correlation name> ] | <joined table>
<joined table> ::= "(" <joined table> ")"
  | <table reference> [ <join type> ] JOIN <table reference> <join specification>
<join type> ::= INNER | LEFT [ OUTER ]
<join specification> ::= ON <column reference> "=" <column reference>
<where clause>   ::= WHERE <search condition>
<search condition> ::= <boolean term> | <search condition> OR <boolean term>
<boolean term>   ::= <boolean factor> | <boolean term> AND <boolean factor>
<boolean factor> ::= [ NOT ] <boolean test>
<boolean test>   ::= <predicate> | "(" <search condition> ")"
<predicate> ::= <comparison predicate> | <in predicate> | <like predicate>
  | <null predicate> | <quantified comparison predicate> | <quantified in predicate>
  | <text search predicate> | <folder predicate>
<comparison predicate> ::= <value expression> <comp op> <literal>
<comp op> ::= "=" | "<>" | "<" | ">" | "<=" | ">="
<literal> ::= <signed numeric literal> | <character string literal>
  | <datetime literal> | <boolean literal>
<in predicate>   ::= <column reference> [ NOT ] IN "(" <in value list> ")"
<in value list>  ::= <literal> [{ "," <literal> }...]
<like predicate> ::= <column reference> [ NOT ] LIKE <character string literal>
<null predicate> ::= { <column reference>
  | <multi-valued-column reference> } IS [ NOT ] NULL
<quantified comparison predicate> ::=
  <literal> "=" ANY <multi-valued-column reference>
<quantified in predicate> ::=
```

```
  ANY <multi-valued-column reference> [ NOT ] IN "(" <in value list> ")"
<text search predicate> ::=
  CONTAINS "(" [ <qualifier>  "," ] <quote> <text search expression> <quote> ")"
<folder predicate> ::= { IN_FOLDER | IN_TREE } "(" [ <qualifier> "," ] <folder id> ")"
<order by clause> ::= ORDER BY <sort specification> [ { "," <sort specification> }... ]
<sort specification> ::= <column reference> [ ASC | DESC ]
<correlation name> ::= <identifier>
<table name>  ::= <identifier> !! This MUST be the name of a primary object-type.
<secondary type table name> ::= <identifier>  !! This MUST be the name of a secondary
    ↪ object-type.
<column name> ::= <identifier> !! This MUST be the name of a single-valued property, or an alias
    ↪ for a scalar output value.
<secondary type column name> ::= <identifier> !! This MUST be the name of a single-valued
    ↪ property for a scalar output value of a secondary type.
<multi-valued-column name> ::= <identifier> !! This MUST be the name of a multi-valued property.
<secondary type multi-valued-column name> ::= <identifier> !! This MUST be the name of a
    ↪ multi-valued property of a secondary type.
<folder id> ::= <character string literal>    !! This MUST be the object identity of a folder
    ↪ object.
<identifier> ::=      !! As defined by queryName attribute.
<signed numeric literal>   ::=  !! As defined by SQL-92 grammar.
<character string literal> ::=  !! As defined by SQL-92 grammar. (i.e. enclosed in single-quotes)

!! This is an independent sub-grammar for full-text search criteria.
!! It is isolatable from the query statement grammar.  (See Escaping)
<text search expression> ::= <conjunct> [ {<space> OR <space> <conjunct>} ... ]
<conjunct> ::= <term> [ {<space> <term>} ... ]
<term>   ::= ['-'] <simple term>
<simple term> ::= <word> | <phrase>
<word> ::= <word element> {<word element>}
<phrase> ::= <double quote> <word> {<space> <word>} <double quote>
<quote symbol> ::= <quote><quote> | <backslash><quote>
<word element> ::= <char> - <space char> - <backslash char> - <quote> - <double quote>
  | <quote symbol>
<space>  ::= <space char> [ {<space char>} ... ]
<space char> ::= ' '
<backslash char> ::= <backslash><backslash>
<char>   ::= !! Any character
<datetime literal> ::= TIMESTAMP <quote> <datetime string> <quote>
<datetime string>  ::= YYYY-MM-DDThh:mm:ss.sss{Z | +hh:mm | -hh:mm}
<boolean literal>  ::= TRUE | FALSE | true | false
<quote> ::= "'" !! Single-quote only, consistent with SQL-92 string literal
<double quote> ::= " !! U+0022
<backslash> ::= \ !! U+005C
```

2.1.14.2.2 SELECT Clause

The SELECT clause MUST 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 MUST include in its result row set all of the columns specified in the SELECT clause.
- * : If this token is specified, then the repository MUST 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 MUST be valid "queryName" values for properties whose virtual tables are listed in the FROM clause. For each "queryName" an alias MAY be defined by adding the string " AS " and the name of the alias to the query name. Alias names MUST comply with the rules for query names. (See section 2.1.2.1.3 Query Names.)

2.1.14.2.3 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 MUST contain only the "queryNames" of object-types whose queryable attribute value is TRUE. For each "queryName" an alias MAY be defined by adding the string " AS " and the name of the alias to the query name. Alias names MUST comply with the rules for query names. (See section 2.1.2.1.3 Query Names.)

2.1.14.2.3.1 Join Support

CMIS repositories MAY support the use of SQL JOIN queries, and MUST indicate their support level using the optional capability attribute `capabilityJoin`.

- If the repository's value for the `capabilityJoin` attribute is `none`, then no JOIN clauses can be used in queries.
- If the repository's value for the `capabilityJoin` attribute is `inneronly`, then only inner JOIN clauses can be used in queries.
- If the repository's value for the `capabilityJoin` attribute is `innerandouter`, then inner and/or outer JOIN clauses can be used in queries.

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

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

2.1.14.2.4 WHERE Clause

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

All column names MUST be valid "queryName" or their aliased values for properties that are defined as "queryable" in the object-type(s) whose virtual tables are listed in the FROM clause.

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

2.1.14.2.4.1 Comparisons permitted in the WHERE clause

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

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

Property Type	Operators supported on Type	Supported type of Literal in comparison
String	=, <>, [NOT] LIKE	String
String (IN)	[NOT] IN	List of Strings
Decimal	=, <>, <, <=, >, >=	Decimal
Decimal (IN)	[NOT] IN	List of Decimal
Integer	=, <>, <, <=, >, >=	Integer
Integer (IN)	[NOT] IN	List of Integer
Boolean	=	Boolean literal
DateTime	=, <>, <[1], <=[1], >[1], >=[1]	DateTime literal
DateTime (IN)	[NOT] IN	List of DateTime literals
ID	=, <>	String
ID (IN)	[NOT] IN	List of strings
URI	=, <>, [NOT] LIKE	String
URI (IN)	[NOT] IN	List of strings

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

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

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

2.1.14.2.4.2 Multi-valued property support (SQL-92 Extension)

The CMIS query language 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.

2.1.14.2.4.3 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.

Quantified comparison predicate

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

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

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

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

[1] Comparison is based on chronological before or after date

Example:

```
SELECT Y.CLAIM_NUM, X.PROPERTY_ADDRESS, Y.DAMAGE_ESTIMATES, Z.BAND
FROM   ( POLICY AS X JOIN CLAIMS AS Y ON X.POLICY_NUM = Y.POLICY_NUM )
       JOIN RISK AS Z ON X.cmis:objectId = Z.cmis:objectId
WHERE  ( 100000 = ANY Y.DAMAGE_ESTIMATES ) AND Z.BAND > 3
```

(Note: DAMAGE_ESTIMATES is a multi-valued Integer property and RISK is a secondary type.)

IN/ANY 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.

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

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

Example 1:

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

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

Example 2:

```
SELECT d.cmis:objectId, d.cmis:name, a.SPECIES
FROM   cmis:document AS d JOIN ANIMAL AS a ON d.cmis:objectId = a.cmis:objectId
WHERE  ANY a.SPECIES IN ('dog', 'cat')
```

(Note: ANIMAL is a secondary type and ANIMAL.SPECIES is a multi-valued String property.)

2.1.14.2.4.4 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 the optional capability attribute `capabilityQuery`.

- If the repository's value for the `capabilityQuery` attribute is `fulltextonly`, then only queries that filter based on the full-text content of documents can be fulfilled. Specifically, only the CONTAINS() predicate function can be included in the WHERE clause.
- If the repository's value for the `capabilityQuery` attribute is `bothseparate`, then the repository can fulfill queries that filter EITHER on the full-text content of documents OR on their properties, but NOT if both types of filters are included in the same query.
- If the repository's value for the `capabilityQuery` attribute is `bothcombined`, then the repository can fulfill queries that filter on both the full-text content of documents and their properties in the same query.

Inputs:

<qualifier> The value of this optional parameter MUST be the name of one of the virtual tables listed in the FROM clause for the query.

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

<text search expression> The <text search expression> parameter MUST be a character string, specifying the full-text search criteria.

- The Text Search Expression may be a set of terms or phrases with an optional '-' to signal negation. A phrase is defined as a word or group of words. A group of words must be surrounded by double quotes to be considered a single phrase.
- Terms may contain wildcards. The wildcard '*' substitutes for zero or more characters. The wildcard '?' substitutes for exactly one character. The characters '%' and '_', which are wildcards in LIKE expressions are not considered wildcards in text serach terms.
- Terms separated by whitespace are AND'ed together.
- Terms separated by "OR" are OR'ed together.
- Implicit "AND" has higher precedence than "OR".
- Within a word or phrase, each (single-)quote must also be escaped by a preceding backslash '\'. Using double single-quotes ('') as a SQL-92 way to escape a literal single-quote (') character SHOULD BE supported as an allowable alternative to the double character '.

Return value:

The predicate returns a Boolean value.

- The predicate MUST return TRUE if the object is considered by the repository as "relevant" with respect to the given <text search expression> parameter.
- The predicate MUST 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 MUST be included in a single query statement. The repository MUST throw an exception if more than one `CONTAINS()` function is found.
- The return value of the `CONTAINS()` function MAY only be included conjunctively (ANDed) with the aggregate of all other predicates, if there is any, in the WHERE clause.

2.1.14.2.4.5 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 MUST be provided for this predicate function.

Return value:

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

- A repository MUST 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 MUST 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.

Constraints:

- The `SCORE()` function MUST only be used in queries that also include a `CONTAINS()` predicate function.
- The `SCORE()` function MUST only be used in the SELECT clause of a query. It MUST NOT be used in the WHERE clause or in the ORDER BY clause.
- 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 query name of `SEARCH_SCORE` is used for the query output, and the property definition id is repository-specific.

2.1.14.2.4.6 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 MUST 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 the query is a join, a server SHOULD throw an exception if the qualifier is not specified.

<folder id> The value of this parameter MUST be the id of a folder object in the repository.

Return value:

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

2.1.14.2.4.7 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 MUST 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 the query is a join, a server SHOULD throw an exception if the qualifier is not specified.

<folder id> The value of this parameter MUST be the id of a folder object in the repository.

Return value:

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

2.1.14.2.5 ORDER BY Clause

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

All column names referenced in this clause MUST be valid "queryName" or their aliased values for properties defined as orderable in the object-type(s) whose virtual tables are listed in the FROM clause.

Only columns in the SELECT clause MAY be in the ORDER BY clause.

Collation rules for the ORDER BY clause are repository specific.

2.1.14.3 Escaping

Character escaping for character strings differs from SQL-92's escaping. A repository MUST support the escaping of certain literal characters in a character string, or in a text search expression, using a backslash character (\) in the following manner. For a <character string literal>, which MUST BE a string enclosed in single-quotes according to the SQL-92 grammar, any occurrence of the single-quote character (') and the escape character (\) in the string MUST BE escaped. This applies to <folder id>, which is a <character string literal>. Furthermore, when a <character string literal> is used in a LIKE predicate, any occurrence of the percent character (%) and the underscore character (_) in the string as a literal MUST BE escaped also. Therefore, within a quoted string in a query:

- The double character \' represents a literal single-quote (') character.
- The double character \\ represents a literal backslash (\) character.
- Within a LIKE string, the double characters \% and _ represent a literal percent (%) character and a literal underscore (_) character respectively.
- Within a CONTAINS text search expression, the double characters * and \? represent a literal asterisk (*) character and a literal question mark (?) character respectively.
- All other instances of a backslash (\) character are errors.

Using double single-quotes ('') as a SQL-92 way to escape a literal single-quote (') character SHOULD BE supported as an allowable alternative to the double character \'.

For a <text search expression>, a second-level character escaping is required so that the <text search expression> sub-grammar is isolatable from the query statement-level grammar. When a text search expression is composed for a query according to the <text search expression> sub-grammar, any occurrence of the following four characters in the expression as a literal character MUST BE escaped: double-quote ("), hyphen (-), single-quote ('), and the escape character (\). Then, before this expression is enclosed in single-quotes and inserted into a CONTAINS() predicate, the query statement-level escaping rules described in the above MUST BE applied. This two-level character escaping allows a query statement parser, using statement-level escaping rules, to correctly extract a <text search expression> as a character string literal independent of the <text search expression> sub-grammar. This extracted <text search expression> can then be correctly interpreted by a full-text search parser independent of the query-statement grammar, using second-level escaping rules. Since the <text search expression> sub-grammar is isolated from the SQL-92 grammar, double single-quotes is not a valid way to escape a literal single-quote character for second-level character escaping.

An <identifier> in a query statement MUST conform to the SQL-92 identifier syntax, and MUST NOT require character escaping.

Example:
A query statement that contains a full-text search for the literal string "John'sPresentation-Version2" may be composed as:

```
SELECT ... FROM ... WHERE ... CONTAINS('John\\\'sPresentation\\-Version2') ...
```

A query parser extracts from this statement the text search expression "John\'sPresentation\-Version2" as a character string literal, and passes it to a text-search parser, which interprets it as a single-word full-text search criteria: John'sPresentation-Version2.

2.1.15 Change Log

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

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

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

2.1.15.1 Completeness of the Change Log

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

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

A repository MAY record events such as filing/unfiling/moving of documents as change events on the documents, their parent folder(s), or both the documents and the parent folders.

2.1.15.2 Change Log Token

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

2.1.15.3 "Latest Change Token" repository information

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

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

2.1.15.4 Change Event

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

A repository that supports the change log capability MUST expose at least the following information for each change object:

Id ObjectId The object id of the object to which the change occurred.

Enum ChangeType An enumeration that indicates the type of the change. Valid values are:

created The object was created.

updated The object was updated.

deleted The object was deleted.

security The access control or security policy for the object were changed.

<Properties> properties Additionally, for events of changeType "updated", the repository MAY optionally include the new values of properties on the object (if any).

Repositories MUST indicate whether they include properties for "updated" change events via the optional `capabilityChanges`.

2.1.16 Retentions and Holds

Retentions and Holds can be used to protect documents from being deleted or modified. A Retention describes a period of time where the document must not be deleted, while a Hold just marks the document as protected as long as the Hold is applied to a document.

This specification defines a basic interface for end user operations. Administrative operations such as managing a file plan or shortening retention periods are out of scope. A repository MAY support settings that require administrative privileges and bend the rules described in the following section. The implications are repository specific.

Retentions and Holds can be applied to documents by applying predefined secondary types for Retentions and Holds. CMIS specifies secondary types for:

- Repository Managed Retentions
- Client Managed Retentions (with a subtype for Destruction Retentions)
- Holds

If a repository does not support one of the predefined types for Retention and Hold management, the corresponding secondary type MUST NOT be returned by a `getTypeChildren` service call.

All secondary types for retention and hold management SHOULD be able to be applied to objects derived from the `cmis:document` base type. Applying such types to other CMIS objects and its behavior is repository specific. A repository MUST throw a `constraint` exception if the operation is not supported.

Retentions and Holds are applied to document versions. How this affects other versions in the version series is repository specfic.

Retentions and Holds protect at least the content of a document from modifications. If this protection also applies to the properties, ACL, policies, relationships, etc. of a document, is repository specific. Clients may use the Allowable Actions to discover what they can do with protected documents.

2.1.16.1 Repository Managed Retentions

Repository Managed Retentions are used in scenarios where the repository is responsible for calculating the concrete expiration date and potential destruction date for a document. As a first step a records manager usually creates a file plan in the repository and assigns rules which are used to calculate the retention period for a specific entry in the file plan. Creating a file plan is out-of-scope for CMIS. It has to be done using the native (user) interfaces of the repository. In order to enable a client to classify documents according to this file plan, the repository exposes the file plan as a secondary type hierarchy. The CMIS client can now apply one of the exposed file plan categories to a document. This process is called classification:

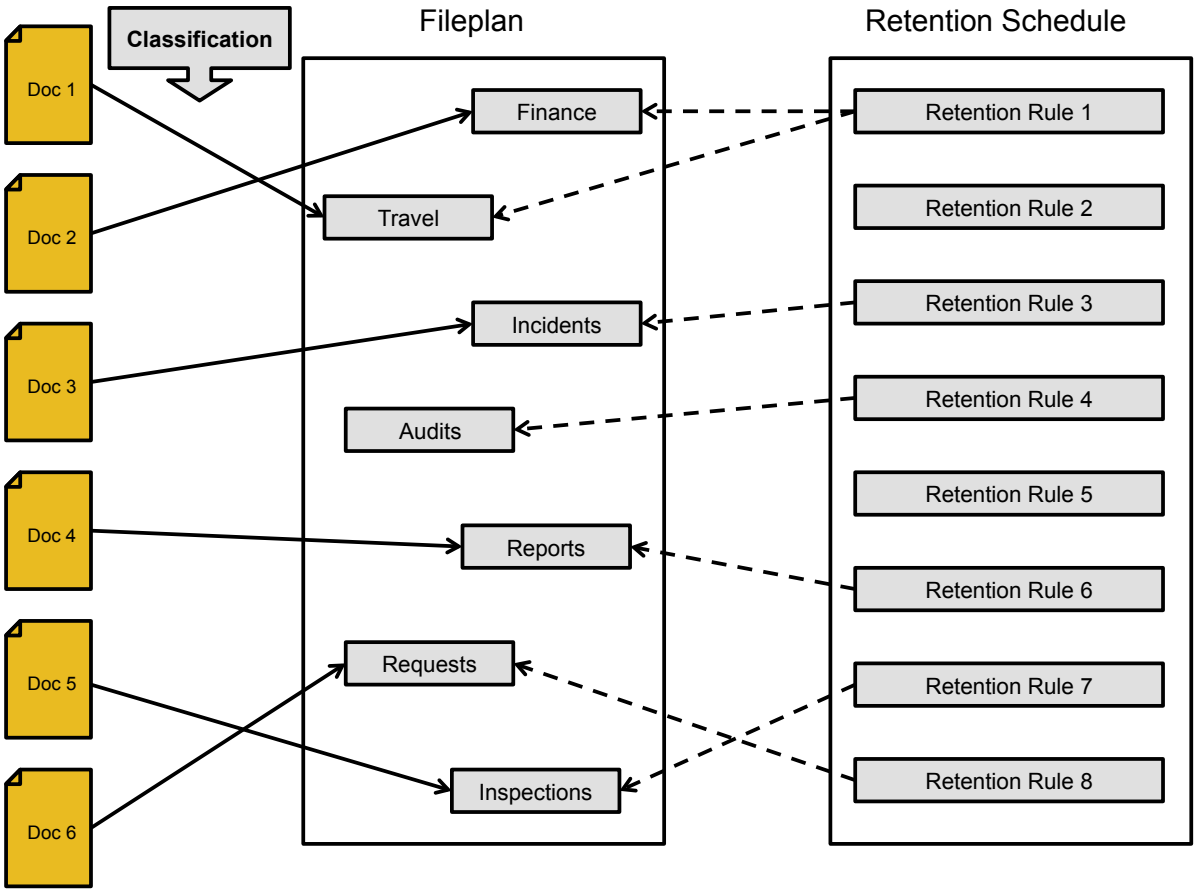

Figure 2.7: Classification

Support for Repository Managed Retentions is optional. A repository that does not support Repository Managed Retentions will not expose a file plan via the secondary type hierarchy. Repositories that support Repository Managed Retentions MUST expose the categories of the file plan as a subtype of the CMIS defined secondary type `cmis:rm_repMgtRetention`. The secondary type `cmis:rm_repMgtRetention` does not require any properties. A repository MAY add repository specific properties. A secondary type hierarchy for Repository Managed Retentions could look like this (white boxes are CMIS defined types, orange boxes are repository specific):

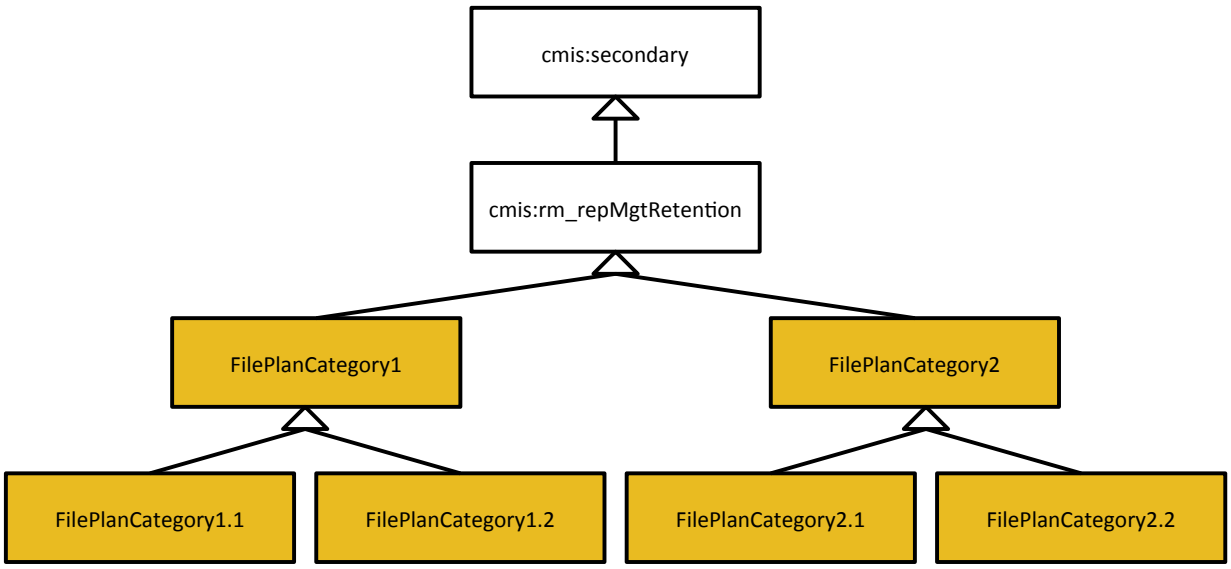

Figure 2.8: Repository Managed Retentions Types

The usage of Repository Managed Retentions allows support of support advanced scenarios where the retention period is not fixed at creation time, but managed more dynamically (e.g. depending on certain property changes like "3 years after setting status to released"). The capabilities that are kind of rules are supported and how they are enforced varies widely between repository implementations. Some may do this automatically, some may require manually triggered batch runs, require an approval or workflow for certain actions etc. This model has minimal requirements for the application but can use much of the functionality that a repository provides.

This specification only defines the classification process, that is applying a Repository Managed Retention to a document. Creating and managing the rules and how rules are mapped to file plan categories is out-of-scope and repository specific. Which set of Repository Managed Retentions can be assigned to which objects is also repository specific.

Whether a user is allowed to apply a Repository Managed Retention is repository specific. If the user has no permission to do so, a `permissionDenied` exception MUST be thrown. In case of others constraints, a `constraint` exception MUST be thrown.

2.1.16.1.1 Repository Managed Retention Type

2.1.16.1.1.1 Attribute Values

The Repository Managed Retention object-type MUST have the following attribute values.

`id`
Value: cmis:rm_repMgtRetention

`localName`
Value: <repository-specific>

`localNamespace`
Value: <repository-specific>

`queryName`
Value: cmis:rm_repMgtRetention

displayName
Value: <repository-specific>

baseId
Value: cmis:secondary

parentId
Value: cmis:secondary

description
Value: <repository-specific>

creatable
Value: FALSE

fileable
Value: FALSE

queryable
Value: SHOULD be TRUE

controllablePolicy
Value: FALSE

controllableACL
Value: FALSE

includedInSupertypeQuery
Value: <repository-specific>

fulltextIndexed
Value: <repository-specific>

typeMutability.create
Value: <repository-specific>

typeMutability.update
Value: <repository-specific>

typeMutability.delete
Value: <repository-specific>

2.1.16.1.1.2 Property Definitions

This type has no properties defined by this specification. A repository MAY add repository specific property definitions.

2.1.16.2 Client Managed Retentions

Client Managed Retentions are used in scenarios where the CMIS client is responsible to calculate the concrete expiration date for a document. This is usually required when documents are related to other objects (like a Business Object in an ERP system) and the documents must get the same retention period than the object where they are related to. In this case a CMIS client can apply a retention period to a document using the Client Managed Retention object-type.

If a repository supports Client Managed Retentions, it exposes the secondary type `cmis:rm_clientMgtRetention` via the secondary type hierarchy. The CMIS defined secondary type `cmis:rm_clientMgtRetention` defines two properties:

cmis:rm_expirationDate contains the date until the document must be preserved.

cmis:rm_startOfRetention contains the date from which the retention time was calculated (for documentation purposes only).

A repository MAY define its own secondary types for Client Managed Retentions with additional properties. Those types MUST be derived from the type `cmis:rm_clientMgtRetention`.

Repositories that support a process to dispose documents after a certain period of time, MAY expose the type `cmis:rm_destructionRetention` which is derived from `cmis:rm_clientMgtRetention`. This type provides an additional property that defines the date when destruction process SHOULD be triggered:

cmis:rm_destructionDate holds the date when the destruction process SHOULD be triggered.

A repository MAY define its own Destruction Retentions. A repository specific Destruction Retention MUST be derived from the type `cmis:rm_destructionRetention`.

The repository MAY round up the dates used for expiration and destruction dates according to its internal capabilities. A secondary type hierarchy for Client Managed Retentions could look like this (white boxes are CMIS defined types, orange boxes are repository specific):

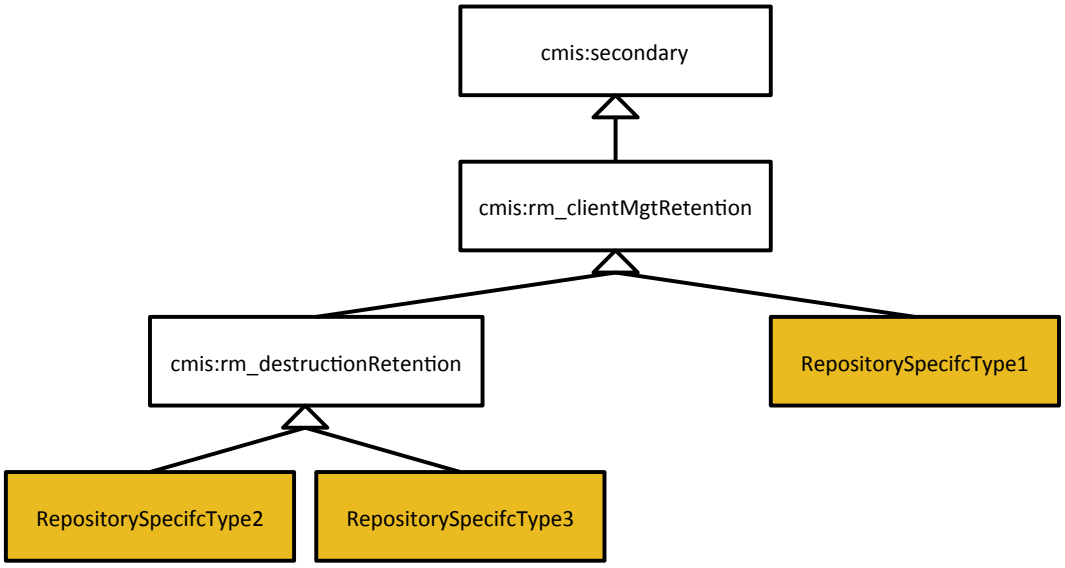

Figure 2.9: Client Managed Retentions Types

2.1.16.2.1 Semantics and Rules to be checked for the Expiration Date Property

The property `cmis:rm_expirationDate` either contains a concrete date or (if not known yet) is in the state "not set". In the first case ("specific expiration date"), the affected object MUST NOT be deletable until the specified date (including the specified date). That does NOT imply that the object is being automatically deleted by the storage system after it expired. In the second case (expiration date "not set"), the affected object MUST NOT be deletable at all. If a new expiration date is applied to an object, the following rules MUST be obeyed:

Assignment rule:

1. A specific expiration date MUST NOT be removable or replaced by an expiration date "not set". The reverse MUST be allowed. That is, it MUST be possible to set a specific expiration date as long as the expiration date is not set.

2. A new expiration date SHALL only be applicable if the expiration date does not lie in the past. Only an expiration date with a current or a future date MUST be assignable to an object.

Prolongation rule:

1. In case an object has already an expiration date assigned, the repository SHALL check whether the new expiration date is equal or greater than the one already assigned. The repository MUST prevent a client from shortening the retention time.
2. Once a Client Managed Retention has been set (with a specific expiration date or expiration date "not set") the Client Managed Retention MUST NOT be removable, even if the expiration date is expired. A violation of any aspect of these rules MUST result in a `constraint` exception. A prolongation of an expiration date MUST succeed regardless of whether the previous expiration date is expired or not.
3. The destruction date, if set, MUST always be the same as the expiration date or greater than the expiration date. When the retention is prolonged, the destruction date may have to be adjusted as well by the client. The repository SHOULD NOT automatically adjust the destruction date.

Whether a user is allowed to apply a Client Managed Retention or Destruction Retention is repository specific. If the user has no permission to do so, a `permissionDenied` exception MUST be thrown. In case of others constraints, a `constraint` exception MUST be thrown.

2.1.16.2.2 Client Managed Retention Type

2.1.16.2.2.1 Attribute Values

The Client Managed Retentions object-type MUST have the following attribute values.

id
Value: cmis:rm_clientMgtRetention

localName
Value: <repository-specific>

localNamespace
Value: <repository-specific>

queryName
Value: cmis:rm_clientMgtRetention

displayName
Value: <repository-specific>

baseId
Value: cmis:secondary

parentId
Value: cmis:secondary

description
Value: <repository-specific>

creatable
Value: FALSE

fileable
Value: FALSE

queryable
Value: SHOULD be TRUE

controllablePolicy
Value: FALSE

controllableACL
Value: FALSE

includedInSupertypeQuery
Value: <repository-specific>

fulltextIndexed
Value: <repository-specific>

typeMutability.create
Value: <repository-specific>

typeMutability.update
Value: <repository-specific>

typeMutability.delete
Value: <repository-specific>

2.1.16.2.2.2 Property Definitions

The Client Managed Retentions object-type MUST have the following property definitions, and MAY include additional property definitions. Any attributes not specified for the property definition are repository specific. The repository MUST have the following property definitions on the Client Managed Retentions object-type:

cmis:rm_expirationDate	Expiration date.
Property Type:	DateTime
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readwrite
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	SHOULD be TRUE
Orderable:	<repository-specific>

cmis:rm_startOfRetention	Start of retention.
Property Type:	DateTime
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	SHOULD be readwrite
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	<repository-specific>
Orderable:	<repository-specific>

2.1.16.2.3 Destruction Retention Type

2.1.16.2.3.1 Attribute Values

The Destruction Retention object-type MUST have the following attribute values.

id
Value: cmis:rm_destructionRetention

localName
Value: <repository-specific>

localNamespace
Value: <repository-specific>

queryName
Value: cmis:rm_destructionRetention

displayName
Value: <repository-specific>

baseId
Value: cmis:secondary

parentId
Value: cmis:rm_clientMgtRetention

description
Value: <repository-specific>

creatable
Value: FALSE

fileable
Value: FALSE

queryable
Value: SHOULD be TRUE

controllablePolicy
Value: FALSE

controllableACL
Value: FALSE

includedInSupertypeQuery
Value: <repository-specific>

fulltextIndexed
Value: <repository-specific>

typeMutability.create
Value: <repository-specific>

typeMutability.update
Value: <repository-specific>

typeMutability.delete
Value: <repository-specific>

2.1.16.2.3.2 Property Definitions

The Destruction Retention object-type MUST have the following property definition, inherits all property defintions from `cmis:rm_clientMgtRetention`, and MAY include additional property definitions. Any attributes not specified for the property definition are repository specific. The repository MUST have the following property definitions on the Destruction Retentions object-type:

cmis:rm_destructionDate	Destruction date.
Property Type:	DateTime
Inherited:	FALSE
Required:	FALSE
Cardinality:	single
Updatability:	readwrite
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	<repository-specific>
Orderable:	<repository-specific>

2.1.16.3 Holds

A Hold assures that a document can be restored to the state it was in when the hold has been applied (usually by protecting the document from being deleted or modified). Support for other objects than documents is repository specific.

If a repository supports holds, it exposes the secondary type `cmis:rm_hold`. This type defines the multi-valued property `cmis:rm_holdIds` which contains a list of identifiers for the affected litigations or audits.

As long as the property `cmis:rm_holdIds` is "not set", the document is not protected by a hold. To protect a document, this property must contain at least one value. The hold type CANNOT be removed from an object as long as the property `cmis:rm_holdIds` contains values.

A repository MAY define its own secondary types for holds with additional properties. Those types MUST be derived from `cmis:rm_hold`.

A secondary type hierarchy for holds could look like this (white boxes are CMIS defined types, orange boxes are repository specific):

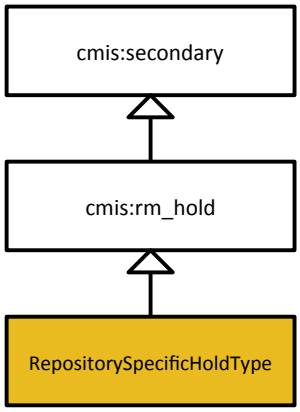

Figure 2.10: Hold Type

Whether a user is allowed to apply a hold is repository-specific. If the user has no permission to do so, a `permissionDenied` exception MUST be thrown. In case of others constraints, a `constraint` exception MUST be thrown.

2.1.16.3.1 Hold Type Definition

2.1.16.3.1.1 Attribute Values

The Hold object-type MUST have the following attribute values.

id
Value: cmis:rm_hold

localName
Value: <repository-specific>

localNamespace
Value: <repository-specific>

queryName
Value: cmis:rm_hold

displayName
Value: <repository-specific>

baseId
Value: cmis:secondary

parentId
Value: cmis:secondary

description
Value: <repository-specific>

creatable
Value: FALSE

fileable
Value: FALSE

queryable
Value: SHOULD be TRUE

controllablePolicy
Value: FALSE

controllableACL
Value: FALSE

includedInSupertypeQuery
Value: <repository-specific>

fulltextIndexed
Value: <repository-specific>

typeMutability.create
Value: <repository-specific>

typeMutability.update
Value: <repository-specific>

typeMutability.delete
Value: <repository-specific>

2.1.16.3.1.2 Property Definitions

The hold object-type MUST have the following property definitions, and MAY include additional property definition. Any attributes not specified for the property definition are repository specific. The repository MUST have the following property definitions on the hold object-type:

cmis:rm_holdIds	Hold Identifiers.
Property Type:	String
Inherited:	FALSE
Required:	FALSE
Cardinality:	multi
Updatability:	SHOULD be readwrite
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	SHOULD be TRUE
Orderable:	FALSE

2.2 Services

The Services section 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 MUST implement all of the methods described in this section or explain why the service is not implemented.

However, the details of how each service and operation is implemented will be described in those protocol binding specifications.

2.2.1 Common Service Elements

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

2.2.1.1 Paging

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

Input Parameters:

Integer `maxItems` (optional) This is the maximum number of items to return in a response. The repository MUST NOT exceed this maximum. Default is repository-specific.

Integer `skipCount` (optional) This is the number of potential results that the repository MUST skip/page over before returning any results. Defaults to 0.

Output Parameters:

Boolean `hasMoreItems` TRUE if the Repository contains additional items after those contained in the response. FALSE otherwise. If TRUE, a request with a larger skipCount or larger maxItems is expected to return additional results (unless the contents of the repository has changed).

Integer `numItems` If the repository knows the total number of items in a result set, the repository SHOULD include the number here. If the repository does not know the number of items in a result set, this parameter SHOULD not be set. The value in the parameter MAY NOT be accurate the next time the client retrieves the result set or the next page in the result set.

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 MUST use the `hasMoreItems` output parameter to indicate if any additional results were not returned.

Repositories MAY return a smaller number of items than the specified value for `maxItems`. A repository SHOULD NOT throw an exception if `maxItems` exceeds the internally supported page size. It SHOULD return a smaller number of items instead.

Each binding will express the above in context and may have different mechanisms for communicating `hasMoreItems` and `numItems`.

2.2.1.2 Retrieving additional information on objects in CMIS service calls

Several CMIS services that return object information have the ability to return dependent object information as part of their response, such as the allowable actions for an object, rendition information, etc.

The CMIS service operations that support returning a result set of objects will include the ability to return the following object information:

- Properties (retrieves a subset instead of additional information)
- Relationships
- Renditions
- ACLs
- AllowableActions

This section describes the input parameter and output pattern for those services. All these input parameters are optional.

2.2.1.2.1 Properties

Description:

All services 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 MUST be returned by the repository in the output of the operation.

Optional Input Parameter:

String **`filter`** Value indicating which properties for objects MUST be returned. This filter is a list of property query names and NOT a list of property ids. The query names of secondary type properties MUST follow the pattern `<secondaryTypeQueryName>.<propertyQueryName>`.
Example: `cmis:name,amount,worflow.stage`

Valid values are:
Not set The set of properties to be returned MUST be determined by the repository.
A comma-delimited list of property definition queryNames The properties listed MUST be returned.
***** All properties MUST be returned for all objects.

If a property is requested by a filter, a property element MUST be returned for that property. A repository MAY return additional properties. If a property filter specifies a property which value is "not set", it MUST be represented as a property element without a value element.

Unknown query names or query names that are not defined for the object-type SHOULD be ignored. For example, if `getChildren` is called with a filter that contains the property `cmis:contentStreamMimeType`, it SHOULD return all non-document objects without this property and SHOULD NOT return an error.

2.2.1.2.2 Relationships

Description:

Used to retrieve the relationships in which the object(s) are participating.

Optional Input Parameter:

Enum **`includeRelationships`** Value indicating what relationships in which the objects returned participate MUST be returned, if any. Values are:
none No relationships MUST be returned. (Default)
source Only relationships in which the objects returned are the source MUST be returned.
target Only relationships in which the objects returned are the target MUST be returned.
both Relationships in which the objects returned are the source or the target MUST be returned.

Output Parameter for each object:

<Array> Relationships A collection of the relationship objects.

2.2.1.2.3 Policies

Description:

Used to retrieve the policies currently applied to the object(s).

Optional Input Parameter:

Boolean **`includePolicyIds`** If TRUE, then the Repository MUST return the Ids of the policies applied to the object. Defaults to FALSE.

Output Parameter for each object:

<Array> Policies A collection of the policy objects.

2.2.1.2.4 Renditions

Description:

Used to retrieve the renditions of the object(s).

Optional Input Parameter:

String **`renditionFilter`** The Repository MUST return the set of renditions whose kind matches this filter. See section below for the filter grammar. Defaults to "cmis:none".

Output Parameter for each object:

<Array> Renditions The set of renditions.

2.2.1.2.4.1 Rendition Filter Grammar

The Rendition Filter grammar is defined as follows:

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

An inclusion pattern allows:

***** Include all associated renditions.

Comma-separated list of Rendition kinds or mimetypes Include only those renditions that match one of the specified kinds or mimetypes.

cmis:none Exclude all associated renditions. (Default)

Examples:

- `*` (include all renditions)
- `cmis:thumbnail` (include only thumbnails)
- `image/*` (include all image renditions)
- `application/pdf,application/x-shockwave-flash` (include web ready renditions)
- `cmis:none` (exclude all renditions)

2.2.1.2.5 ACLs

Description:

Used to retrieve the ACLs for the object(s) described in the service response.

Optional Input Parameter:

Boolean `includeACL` If TRUE, then the repository MUST return the ACLs for each object in the result set. Defaults to FALSE.

Output Parameter for each object:

<Array> ACEs The list of access control entries of the ACL for the object.

If the repository does not support ACLs, it should not return an error if `includeACL` is set to TRUE but ignore this parameter.

2.2.1.2.6 Allowable Actions

Description:

Used to retrieve the allowable actions for the object(s) described in the service response.

Optional Input Parameter:

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

Output Parameter for each object:

<Array> AllowableActions The list of allowable actions for the object.

2.2.1.2.7 Object Order

Description:

Used to define the order of the list of objects returned by `getChildren` and `getCheckedOut-Docs`.
If the optional capability `capabilityOrderBy` is "none" and this parameter is set, the repository SHOULD return an `invalidArgument` error.
If the optional capability `capabilityOrderBy` is "common" and this parameter contains a query name that is not in the set of common properties (see below), the repository SHOULD return an `invalidArgument` error.
If a repository only supports a certain number of `orderBy` properties, it SHOULD ignore all additional properties.
If this parameter contains a query name that is unknown or a query name that belongs to a property that is not queryable, the repository SHOULD ignore it.
The query names of secondary type properties MUST follow this pattern: `<secondaryTypeQueryName>.<propertyQueryName>`

Common CMIS properties:

The following set of properties SHOULD be supported by the repository if the optional capability `capabilityOrderBy` is `common` or `custom`.

- `cmis:name`
- `cmis:objectId`
- `cmis:objectTypeId`
- `cmis:baseTypeId`
- `cmis:createdBy`
- `cmis:creationDate`
- `cmis:lastModifiedBy`
- `cmis:lastModificationDate`
- `cmis:isImmutable`
- `cmis:isPrivateWorkingCopy`
- `cmis:isLatestVersion`
- `cmis:isMajorVersion`
- `cmis:isLatestMajorVersion`
- `cmis:versionLabel`
- `cmis:versionSeriesId`
- `cmis:isVersionSeriesCheckedOut`
- `cmis:versionSeriesCheckedOutBy`
- `cmis:versionSeriesCheckedOutId`
- `cmis:checkinComment`
- `cmis:contentStreamLength`
- `cmis:contentStreamMimeType`
- `cmis:contentStreamFileName`
- `cmis:contentStreamId`
- `cmis:parentId`
- `cmis:allowedChildObjectTypeIds`
- `cmis:path`

Optional Input Parameter:

String `orderBy` A comma-separated list of query names and an optional ascending modifier "ASC" or descending modifier "DESC" for each query name. If the modifier is not stated, "ASC" is assumed.

Example:

```
cmis:baseTypeId,cmis:contentStreamLength DESC,cmis:name
```

2.2.1.3 Change Tokens

The CMIS base object-type definitions include an opaque string `cmis: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 a value for the `cmis:changeToken` property for an object, then all invocations of the "update" methods on that object (`updateProperties`, `bulkUpdateProperties`, `setContentStream`, `appendContentStream`, `deleteContentStream`, etc.) MUST provide the value of the `cmis:changeToken` property as an input parameter, and the repository MUST throw an `updateConflictException` if the value specified for the change token does NOT match the change token value for the object being updated.

2.2.1.4 Exceptions

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

2.2.1.4.1 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.

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

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

objectNotFound
Cause: The service call has specified an object, an object-type or a repository that does not exist.

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

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

2.2.1.4.2 Specific Exceptions

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

For each exception, the general intent is listed.

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

contentAlreadyExists
Intent: The operation attempts to set the content stream for a document that already has a content stream without explicitly specifying the "overwriteFlag" parameter.

filterNotValid
Intent: The property filter or rendition filter input to the operation is not valid. The repository SHOULD NOT throw this expection if the filter syntax is correct but one or more elements in the filter is unknown. Unknown elements SHOULD be ignored.

nameConstraintViolation
Intent: The repository is not able to store the object that the user is creating/updating due to a name constraint violation.

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

streamNotSupported
Intent: The operation is attempting to get or set a content stream for a document whose object-type specifies that a content stream is not allowed for document's of that type.

updateConflict
Intent: The operation is attempting to update an object that is no longer current (as determined by the repository). See also section 2.2.1.3 Change Tokens.

versioning

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

2.2.1.5 ACLs

Those services which allow for the setting of ACLs MAY take the optional macro `cmis:user` which allows the caller to indicate the operation applies to the current authenticated user.

If the repository info provides a value for `principalAnonymous`, this value can be used to in an ACE to specify permissions for anonymous users.

If the repository info provides a value for `principalAnyone`, this value can be used to in an ACE to specify permissions for any authenticated user.

2.2.2 Repository Services

The Repository Services are used to discover information about the repository, including information about the repository and the object-types defined for the repository. Furthermore, it provides operations to create, modify and delete object-type definitions if that is supported by the repository.

2.2.2.1 getRepositories

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

2.2.2.1.1 Inputs

None.

2.2.2.1.2 Outputs

- **Id `repositoryId`:** The identifier for the repository.
- **String `repositoryName`:** A display name for the repository.

2.2.2.1.3 Exceptions Thrown & Conditions

See section 2.2.1.4.1 General Exceptions.

2.2.2.2 getRepositoryInfo

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

2.2.2.2.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.

2.2.2.2.2 Outputs

- **Id `repositoryId`:** The identifier for the repository.
  Note: This MUST be the same identifier as the input to the method.
- **String `repositoryName`:** A display name for the repository.
- **String `repositoryDescription`:** A display description for the repository.
- **String `vendorName`:** A display name for the vendor of the repository's underlying application.
- **String `productName`:** A display name for the repository's underlying application.
- **String `productVersion`:** A display name for the version number of the repository's underlying application.
- **Id `rootFolderId`:** The id of the root folder object for the repository.
- **List<Capabilities> `capabilities`:** The set of values for the repository-optional capabilities specified in section 2.1.1.1 Optional Capabilities.
- **String `latestChangeLogToken`:** The change log token corresponding to the most recent change event for any object in the repository. See section 2.1.15 Change Log.
- **String `cmisVersionSupported`:** A Decimal as String that indicates what version of the CMIS specification this repository supports as specified in section 2.1.1.2 Implementation Information. This value MUST be "1.1".
- **URI `thinClientURI`:** A optional repository-specific URI pointing to the repository's web interface. MAY be not set.
- **Boolean `changesIncomplete`:** Indicates whether or not the repository's change log can return all changes ever made to any object in the repository or only changes made after a particular point in time. Applicable when the repository's optional capability `capabilityChanges` is not `none`.
  - If FALSE, then the change log can return all changes ever made to every object.
  - If TRUE, then the change log includes all changes made since a particular point in time, but not all changes ever made.
- **<Array> Enum `changesOnType`:** Indicates whether changes are available for base types in the repository. Valid values are from enumBaseObjectTypeIds. See section 2.1.15 Change Log.
  - `cmis:document`
  - `cmis:folder`
  - `cmis:policy`
  - `cmis:relationship`
  - `cmis:item`

  Note: The base type `cmis:secondary` MUST NOT be used here. Only primary base types can be in this list.
- **Enum `supportedPermissions`:** Specifies which types of permissions are supported.

basic Indicates that the CMIS basic permissions are supported.

repository Indicates that repository specific permissions are supported.

both Indicates that both CMIS basic permissions and repository specific permissions are supported.

- **Enum `propagation`:** The allowed value(s) for `applyACL`, which control how non-direct ACEs are handled by the repository. See section 2.1.12.3 ACL Capabilities.
- **<Array> Permission `permissions`:** The list of repository-specific permissions the repository supports for managing ACEs. See section 2.1.12 Access Control.
- **<Array> PermissionMapping `mapping`:** The list of mappings for the CMIS basic permissions to allowable actions. See section 2.1.12 Access Control.
- **String `principalAnonymous`:** If set, this field holds the principal who is used for anonymous access. This principal can then be passed to the ACL services to specify what permissions anonymous users should have.
- **String `principalAnyone`:** If set, this field holds the principal who is used to indicate any authenticated user. This principal can then be passed to the ACL services to specify what permissions any authenticated user should have.
- **<Array> RepositoryFeatures `extendedFeatures`:** Optional list of additional repository features. See section 2.1.1.3 Repository Features.

2.2.2.2.3 Exceptions Thrown & Conditions

See section 2.2.1.4.1 General Exceptions.

2.2.2.3 getTypeChildren

Description: Returns the list of object-types defined for the repository that are children of the specified type.

2.2.2.3.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.

Optional:

- **Id `typeId`:** The typeId of an object-type specified in the repository.
  - If specified, then the repository MUST return all of child types of the specified type.
  - If not specified, then the repository MUST return all base object-types.
- **Boolean `includePropertyDefinitions`:** If TRUE, then the repository MUST return the property definitions for each object-type. If FALSE (default), the repository MUST return only the attributes for each object-type.
- **Integer `maxItems`:** See section 2.2.1.1 Paging.
- **Integer `skipCount`:** See section 2.2.1.1 Paging.

2.2.2.3.2 Outputs

- **<Array> Object-Type `types`:** The list of child object-types defined for the given typeId.
- **Boolean `hasMoreItems`:** See section 2.2.1.1 Paging.
- **Integer `numItems`:** See section 2.2.1.1 Paging.

2.2.2.3.3 Exceptions Thrown & Conditions

See section 2.2.1.4.1 General Exceptions.

2.2.2.4 getTypeDescendants

Description: Returns the set of the descendant object-types defined for the Repository under the specified type.

Notes:

- This method does NOT support paging as defined in the 2.2.1.1 Paging section.
- The order in which results are returned is respository-specific.

2.2.2.4.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.

Optional:

- **Id `typeId`:** The typeId of an object-type specified in the repository.
  - If specified, then the repository MUST return all of descendant types of the specified type.
  - If not specified, then the Repository MUST return all types and MUST ignore the value of the depth parameter.
- **Integer `depth`:** The number of levels of depth in the type hierarchy from which to return results. Valid values are:

  1 Return only types that are children of the type. See also `getTypeChildren`.

  <Integer value greater than 1> Return only types that are children of the type and descendants up to <value> levels deep.

  -1 Return ALL descendant types at all depth levels in the CMIS hierarchy.

  The default value is repository specific and SHOULD be at least 2 or -1.
- **Boolean `includePropertyDefinitions`:** If TRUE, then the repository MUST return the property definitions for each object-type. If FALSE (default), the repository MUST return only the attributes for each object-type.

2.2.2.4.2 Outputs

- **<Array> Object-Type `types`:** The hierarchy of object-types defined for the repository.

2.2.2.4.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `invalidArgument` If the service is invoked with "depth = 0" or "depth < -1".

2.2.2.5 getTypeDefinition

Description: Gets the definition of the specified object-type.

2.2.2.5.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `typeId`:** The typeId of an object-type specified in the repository.

2.2.2.5.2 Outputs

- **Object-Type `type`:** Object-type including all property definitions. See section 2.1.3 Object-Type for further details.

2.2.2.5.3 Exceptions Thrown & Conditions

See section 2.2.1.4.1 General Exceptions.

2.2.2.6 createType

Description: Creates a new type definition that is a subtype of an existing specified parent type.

Notes: Only properties that are new to this type (not inherited) are passed to this service.

See section 2.1.10 Object-Type Creation, Modification and Deletion for further details.

2.2.2.6.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Object-Type `type`:** A fully populated type definition including all new property definitions.

2.2.2.6.2 Outputs

- **Object-Type `type`:** The newly created object-type including all property definitions. See sections 2.1.3 Object-Type and 2.1.10 Object-Type Creation, Modification and Deletion for further details.

2.2.2.6.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `invalidArgument` If the specified parent type does not exist or the specified parent type cannot be used as the parent type.
- `constraint` If the type definition violates repository specifc rules.

2.2.2.7 updateType

Description: Updates a type definition.

Notes: If you add an optional property to a type in error. There is no way to remove it/correct it - without deleting the type. See section 2.1.10 Object-Type Creation, Modification and Deletion for further details.

2.2.2.7.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Object-Type `type`:** A type definition object with the property definitions that are to change. Repositories MUST ignore all fields in the type definition except for the type id and the list of properties.

  Properties that are not changing MUST NOT be included, including any inherited property definitions.

  For the properties that are being included, an entire copy of the property definition should be present (with the exception of the choice values – see special note), even values that are not changing.

  Special note about choice values. There are only two types of changes permitted.

  - New choice added to the list.
  - Changing the displayname for an existing choice.

  For any choice that is being added or having its display name changed, both the displayName and value MUST be present.

2.2.2.7.2 Outputs

- **Object-Type `type`:** The updated object-type including all property definitions. See sections 2.1.3 Object-Type and 2.1.10 Object-Type Creation, Modification and Deletion for further details.

2.2.2.7.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `constraint` If the rules listed in section 2.1.10 Object-Type Creation, Modification and Deletion are not obeyed.
- `constraint` If the property definitions violate repository specifc rules.

2.2.2.8 deleteType

Description: Deletes a type definition.

Notes: If there are object instances present of the type being deleted then this operation MUST fail.

See sections 2.1.10 Object-Type Creation, Modification and Deletion for further details.

2.2.2.8.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `typeId`:** The typeId of an object-type specified in the repository.

2.2.2.8.2 Outputs

- None.

2.2.2.8.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `constraint` If objects of this object-type exist in the repository.
- `constraint` If the object-type has a sub-type.

2.2.3 Navigation Services

The Navigation Services are used to traverse the folder hierarchy in a CMIS repository, and to locate documents that are checked out.

2.2.3.1 getChildren

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

Notes: If the repository supports the optional capability `capabilityVersionSpecificFiling`, then the repository MUST return the document versions filed in the specified folder. Otherwise, the latest version or the latest major version of the documents MUST be returned.

2.2.3.1.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `folderId`:** The identifier for the folder.

Optional:

- **Integer `maxItems`:** See section 2.2.1.1 Paging.
- **Integer `skipCount`:** See section 2.2.1.1 Paging.
- **String `orderBy`:** See section 2.2.1.2.7 Object Order.
- **String `filter`:** See section 2.2.1.2.1 Properties.
- **Enum `includeRelationships`:** See section 2.2.1.2.2 Relationships.
- **String `renditionFilter`:** See section 2.2.1.2.4 Renditions.
- **Boolean `includeAllowableActions`:** See section 2.2.1.2.6 Allowable Actions.
- **Boolean `includePathSegment`:** If TRUE, returns a PathSegment for each child object for use in constructing that object's path. Defaults to FALSE. See section 2.1.5.3 Paths.

2.2.3.1.2 Outputs

- **<Array> objects `objects`:** A list of the child objects for the specified folder. Each object result MUST include the following elements if they are requested:

  Properties See section 2.2.1.2.1 Properties.

  Relationships See section 2.2.1.2.2 Relationships.

  Renditions See section 2.2.1.2.4 Renditions.

  AllowableActions See section 2.2.1.2.6 Allowable Actions.

  PathSegment If `includePathSegment` was TRUE. See section 2.1.5.3 Paths.
- **Boolean `hasMoreItems`:** See section 2.2.1.1 Paging.
- **Integer `numItems`:** See section 2.2.1.1 Paging.

2.2.3.1.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `filterNotValid` If the property or rendition filter input parameter is not valid.
- `invalidArgument` If the specified folder is not a folder.

2.2.3.2 getDescendants

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

Notes:

- This operation does NOT support paging as defined in the 2.2.1.1 Paging section.
- The order in which results are returned is respository-specific.
- If the repository supports the optional capability `capabilityVersionSpecificFiling`, then the repository MUST return the document versions filed in the specified folder or its descendant folders. Otherwise, the latest version or latest major version of the documents MUST be returned.
- If the repository supports the optional capability `capabilityMultifiling` and the same document is encountered multiple times in the hierarchy, then the repository MUST return that document each time it is encountered.

2.2.3.2.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `folderId`:** The identifier for the folder.

Optional:

- **Integer `depth`:** The number of levels of depth in the folder hierarchy from which to return results. Valid values are:

  1 Return only objects that are children of the folder. See also `getChildren`.

  <Integer value greater than 1> Return only objects that are children of the folder and descendants up to <value> levels deep.

  -1 Return ALL descendant objects at all depth levels in the CMIS hierarchy.

  The default value is repository specific and SHOULD be at least 2 or -1.
- **String `filter`:** See section 2.2.1.2.1 Properties.
- **Enum `includeRelationships`:** See section 2.2.1.2.2 Relationships.
- **String `renditionFilter`:** See section 2.2.1.2.4 Renditions.
- **Boolean `includeAllowableActions`:** See section 2.2.1.2.6 Allowable Actions.
- **Boolean `includePathSegment`:** If TRUE, returns a PathSegment for each child object for use in constructing that object's path. Defaults to FALSE. See section 2.1.5.3 Paths.

2.2.3.2.2 Outputs

- **<Array> objects `objects`:** A tree of the child objects for the specified folder. Each object result MUST include the following elements if they are requested:

  Properties See section 2.2.1.2.1 Properties.

  Relationships See section 2.2.1.2.2 Relationships.

  Renditions See section 2.2.1.2.4 Renditions.

  AllowableActions See section 2.2.1.2.6 Allowable Actions.

  PathSegment If `includePathSegment` was TRUE. See section 2.1.5.3 Paths.

2.2.3.2.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `filterNotValid` If the property or rendition filter input parameter is not valid.
- `invalidArgument` If the specified folder is not a folder.
- `invalidArgument` If the service is invoked with "depth = 0" or "depth < -1".

2.2.3.3 getFolderTree

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

Notes:

- This operation does NOT support paging as defined in the 2.2.1.1 Paging section.
- The order in which results are returned is respository-specific.

2.2.3.3.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `folderId`:** The identifier for the folder.

Optional:

- **Integer `depth`:** The number of levels of depth in the folder hierarchy from which to return results. Valid values are:

  1 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 deep.

  -1 Return ALL descendant objects at all depth levels in the CMIS hierarchy.

  The default value is repository specific and SHOULD be at least 2 or -1.
- **String `filter`:** See section 2.2.1.2.1 Properties.
- **Enum `includeRelationships`:** See section 2.2.1.2.2 Relationships.
- **String `renditionFilter`:** See section 2.2.1.2.4 Renditions.
- **Boolean `includeAllowableActions`:** See section 2.2.1.2.6 Allowable Actions.
- **Boolean `includePathSegment`:** If TRUE, returns a PathSegment for each child object for use in constructing that object's path. Defaults to FALSE. See section 2.1.5.3 Paths.

2.2.3.3.2 Outputs

- **<Array> objects `objects`:** A tree of the child objects for the specified folder. Each object result MUST include the following elements if they are requested:

  Properties See section 2.2.1.2.1 Properties.

  Relationships See section 2.2.1.2.2 Relationships.

  Renditions See section 2.2.1.2.4 Renditions.

  AllowableActions See section 2.2.1.2.6 Allowable Actions.

  PathSegment If `includePathSegment` was TRUE. See section 2.1.5.3 Paths.

2.2.3.3.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `filterNotValid` If the property or rendition filter input parameter is not valid.
- `invalidArgument` If the specified folder is not a folder.

- `invalidArgument` If the service is invoked with "depth = 0" or "depth < -1".

2.2.3.4 getFolderParent

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

2.2.3.4.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `folderId`:** The identifier for the folder.

Optional:

- **String `filter`:** See section 2.2.1.2.1 Properties.

2.2.3.4.2 Outputs

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

  The repository SHOULD return an object that is equal to the object returned by `getObject` with default parameters.

2.2.3.4.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `filterNotValid` If the property filter input parameter is not valid.
- `invalidArgument` If the specified folder is not a folder.
- `invalidArgument` If the specified folder is the root folder.

2.2.3.5 getObjectParents

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

2.2.3.5.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `objectId`:** The identifier for the object.

Optional:

- **String `filter`:** See section 2.2.1.2.1 Properties.
- **Enum `includeRelationships`:** See section 2.2.1.2.2 Relationships.
- **String `renditionFilter`:** See section 2.2.1.2.4 Renditions.
- **Boolean `includeAllowableActions`:** See section 2.2.1.2.6 Allowable Actions.
- **Boolean `includeRelativePathSegment`:** See section 2.1.5.3 Paths.

2.2.3.5.2 Outputs

- **<Array> objects `objects`:** A list of the parent folder(s) of the specified objects. Empty for the root folder and unfiled objects. Each object result MUST include the following elements if they are requested:

  Properties See section 2.2.1.2.1 Properties.

  Relationships See section 2.2.1.2.2 Relationships.

  Renditions See section 2.2.1.2.4 Renditions.

  AllowableActions See section 2.2.1.2.6 Allowable Actions.

  RelativePathSegment If `includeRelativePathSegment` was TRUE. See section 2.1.5.3 Paths.
- **Boolean `hasMoreItems`:** See section 2.2.1.1 Paging.
- **Integer `numItems`:** See section 2.2.1.1 Paging.

2.2.3.5.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `filterNotValid` If the property or rendition filter input parameter is not valid.

2.2.3.6 getCheckedOutDocs

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

2.2.3.6.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.

Optional:

- **Id `folderId`:** The identifier for the folder.

  If specified, the repository MUST only return checked out documents that are child-objects of the specified folder.

  If not specified, the repository MUST return checked out documents from anywhere in the repository hierarchy.
- **Integer `maxItems`:** See section 2.2.1.1 Paging.
- **Integer `skipCount`:** See section 2.2.1.1 Paging.
- **String `orderBy`:** See section 2.2.1.2.7 Object Order.
- **String `filter`:** See section 2.2.1.2.1 Properties.
- **Enum `includeRelationships`:** See section 2.2.1.2.2 Relationships.
- **String `renditionFilter`:** See section 2.2.1.2.4 Renditions.
- **Boolean `includeAllowableActions`:** See section 2.2.1.2.6 Allowable Actions.

2.2.3.6.2 Outputs

- **<Array> objects `objects`:** A list of checked out documents. Each object result MUST include the following elements if they are requested:

  Properties See section 2.2.1.2.1 Properties.

  Relationships See section 2.2.1.2.2 Relationships.

  Renditions See section 2.2.1.2.4 Renditions.

  AllowableActions See section 2.2.1.2.6 Allowable Actions.
- **Boolean `hasMoreItems`:** See section 2.2.1.1 Paging.
- **Integer `numItems`:** See section 2.2.1.1 Paging.

2.2.3.6.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `filterNotValid` If the property or rendition filter input parameter is not valid.
- `invalidArgument` If a folder is specified but the folder is not a folder.

2.2.4 Object Services

CMIS provides id-based CRUD (Create, Retrieve, Update, Delete) operations on objects in a repository.

2.2.4.1 createDocument

Description: Creates a document object of the specified type (given by the `cmis:objectTypeId` property) in the (optionally) specified location.

2.2.4.1.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **<Array> Property `properties`:** The property values that MUST be applied to the newly-created document object.

Optional:

- **Id `folderId`:** If specified, the identifier for the folder that MUST 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.
- **<contentStream> `contentStream`:** The content stream that MUST be stored for the newly-created document object. The method of passing the contentStream to the server and the encoding mechanism will be specified by each specific binding. MUST be required if the type requires it.
- **Enum `versioningState`:** An enumeration specifying what the versioning state of the newly-created object MUST be. Valid values are:

  none (default, if the object-type is not versionable) The document MUST be created as a non-versionable document.

  checkedout The document MUST be created in the checked-out state. The checked-out document MAY be visible to other users.

  major (default, if the object-type is versionable) The document MUST be created as a major version.

  minor The document MUST be created as a minor version.
- **<Array> Id `policies`:** A list of policy ids that MUST be applied to the newly-created document object.
- **<Array> ACE `addACEs`:** A list of ACEs that MUST be added to the newly-created document object, either using the ACL from folderId if specified, or being applied if no folderId is specified.
- **<Array> ACE `removeACEs`:** A list of ACEs that MUST be removed from the newly-created document object, either using the ACL from folderId if specified, or being ignored if no folderId is specified.

2.2.4.1.2 Outputs

- **Id `objectId`:** The id of the newly-created document.

2.2.4.1.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `constraint` If the `cmis:objectTypeId` property value is not an object-type whose base type is `cmis:document`.
- `constraint` If the `cmis:objectTypeId` property value is NOT in the list of `AllowedChildObjectTypeIds` of the parent-folder specified by folderId.
- `constraint` If the value of any of the properties violates the constraints (min/max/required/length etc.) specified in the property definition in the object-type.

- `constraint` If the `contentStreamAllowed` attribute of the object-type definition specified by the `cmis:objectTypeId` property value is set to "required" and no contentStream input parameter is provided.
- `constraint` If the `versionable` attribute of the object-type definition specified by the `cmis:objectTypeId` property value is set to FALSE and the value for the versioningState input parameter is provided that is something other than `none` or "not set".
- `constraint`If the `versionable` attribute of the object-type definition specified by the `cmis:objectTypeId` property value is set to TRUE and the value for the versioningState input parameter provided is `none`.
- `constraint` If the `controllablePolicy` attribute of the object-type definition specified by the `cmis:objectTypeId` property value is set to FALSE and at least one policy is provided.
- `constraint` If the `controllableACL` attribute of the object-type definition specified by the `cmis:objectTypeId` property value is set to FALSE and at least one ACE is provided.
- `constraint` If any permission referenced in a provided ACE is not supported by the repository. (see also `applyACL`).
- `nameConstraintViolation` If the repository detects a violation with the given `cmis:name` property value, the repository MAY throw this exception or chose a name which does not conflict.
- `storage` See section 2.2.1.4.2 Specific Exceptions.
- `streamNotSupported` See section 2.2.1.4.2 Specific Exceptions.

2.2.4.2 createDocumentFromSource

Description: Creates a document object as a copy of the given source document in the (optionally) specified location.

2.2.4.2.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `sourceId`:** The identifier for the source document.

Optional:

- **<Array> Property `properties`:** The property values that MUST be applied to the object. This list of properties SHOULD only contain properties whose values differ from the source document.
- **Id `folderId`:** If specified, the identifier for the folder that MUST 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.
- **Enum `versioningState`:** An enumeration specifying what the versioning state of the newly-created object MUST be. Valid values are:

  none (default, if the object-type is not versionable) The document MUST be created as a non-versionable document.

  checkedout The document MUST be created in the checked-out state. The checked-out document MAY be visible to other users.

  major (default, if the object-type is versionable) The document MUST be created as a major version.

  minor The document MUST be created as a minor version.
- **<Array> Id `policies`:** A list of policy ids that MUST be applied to the newly-created document object.
- **<Array> ACE `addACEs`:** A list of ACEs that MUST be added to the newly-created document object, either using the ACL from folderId if specified, or being applied if no folderId is specified.
- **<Array> ACE `removeACEs`:** A list of ACEs that MUST be removed from the newly-created document object, either using the ACL from folderId if specified, or being ignored if no folderId is specified.

2.2.4.2.2 Outputs

- **Id `objectId`:** The id of the newly-created document.

2.2.4.2.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `constraint` If the sourceId is not an object whose baseType is `cmis:document`.
- `constraint` If the `cmis:objectTypeId` property value is NOT in the list of `AllowedChildObjectTypeIds` of the parent-folder specified by folderId.
- `constraint` If the value of any of the properties violates the constraints (min/max/required/length etc.) specified in the property definition in the object-type.

- constraint If the versionable attribute of the object-type definition specified by the cmis:objectTypeId property value is set to FALSE and the value for the versioningState input parameter is provided that is something other than none or "not set".
- constraintIf the versionable attribute of the object-type definition specified by the cmis:objectTypeId property value is set to TRUE and the value for the versioningState input parameter provided is none.
- constraint If the controllablePolicy attribute of the object-type definition specified by the cmis:objectTypeId property value is set to FALSE and at least one policy is provided.
- constraint If the controllableACL attribute of the object-type definition specified by the cmis:objectTypeId property value is set to FALSE and at least one ACE is provided.
- constraint If any permission referenced in a provided ACE is not supported by the repository. (see also applyACL).
- nameConstraintViolation If the repository detects a violation with the given cmis:name property value, the repository MAY throw this exception or chose a name which does not conflict.
- storage See section 2.2.1.4.2 Specific Exceptions.
- streamNotSupported See section 2.2.1.4.2 Specific Exceptions.

2.2.4.3 createFolder

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

2.2.4.3.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **<Array> Property `properties`:** The property values that MUST be applied to the newly-created folder object.
- **Id `folderId`:** The identifier for the folder that MUST be the parent folder for the newly-created folder object.

Optional:

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

2.2.4.3.2 Outputs

- **Id `objectId`:** The id of the newly-created folder.

2.2.4.3.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `constraint` If the `cmis:objectTypeId` property value is not an object-type whose base type is `cmis:folder`.
- `constraint` If the `cmis:objectTypeId` property value is NOT in the list of `AllowedChildObjectTypeIds` of the parent-folder specified by folderId.
- `constraint` If the value of any of the properties violates the constraints (min/max/required/length etc.) specified in the property definition in the object-type.
- `constraint` If the `controllablePolicy` attribute of the object-type definition specified by the `cmis:objectTypeId` property value is set to FALSE and at least one policy is provided.
- `constraint` If the `controllableACL` attribute of the object-type definition specified by the `cmis:objectTypeId` property value is set to FALSE and at least one ACE is provided.
- `constraint` If at least one of the specified values for permission in ANY of the ACEs does not match ANY of the permission names returned by the ACL Capabilities in the Repository Info (see section 2.1.12.3.1 Supported Permissions) and is not a CMIS basic permission.
- `nameConstraintViolation` If the repository detects a violation with the given `cmis:name` property value, the repository MAY throw this exception or chose a name which does not conflict.
- `storage` See section 2.2.1.4.2 Specific Exceptions.

2.2.4.4 createRelationship

Description: Creates a relationship object of the specified type.

2.2.4.4.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **<Array> Property `properties`:** The property values that MUST be applied to the newly-created relationship object.

Optional:

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

2.2.4.4.2 Outputs

- **Id `objectId`:** The id of the newly-created relationship.

2.2.4.4.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `constraint` If the `cmis:objectTypeId` property value is not an object-type whose base type is `cmis:relationship`.
- `constraint` If the value of any of the properties violates the constraints (min/max/required/length etc.) specified in the property definition in the object-type.
- `constraint` If the source object's object-type is not in the list of "allowedSourceTypes" specified by the object-type definition specified by `cmis:objectTypeId` property value.
- `constraint` If the target object's object-type is not in the list of "allowedTargetTypes" specified by the object-type definition specified by `cmis:objectTypeId` property value.
- `constraint` If the `controllablePolicy` attribute of the object-type definition specified by the `cmis:objectTypeId` property value is set to FALSE and at least one policy is provided.
- `constraint` If the `controllableACL` attribute of the object-type definition specified by the `cmis:objectTypeId` property value is set to FALSE and at least one ACE is provided.
- `constraint` If at least one of the specified values for permission in ANY of the ACEs does not match ANY of the permission names returned by the ACL Capabilities in the Repository Info (see section 2.1.12.3.1 Supported Permissions) and is not a CMIS basic permission.
- `nameConstraintViolation` If the repository detects a violation with the given `cmis:name` property value, the repository MAY throw this exception or chose a name which does not conflict.
- `storage` See section 2.2.1.4.2 Specific Exceptions.

2.2.4.5 createPolicy

Description: Creates a policy object of the specified type.

2.2.4.5.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **<Array> Property `properties`:** The property values that MUST be applied to the newly-created policy object.

Optional:

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

2.2.4.5.2 Outputs

- **Id `objectId`:** The id of the newly-created policy.

2.2.4.5.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `constraint` If the `cmis:objectTypeId` property value is not an object-type whose base type is `cmis:policy`.
- `constraint` If the value of any of the properties violates the constraints (min/max/required/length etc.) specified in the property definition in the object-type.
- `constraint` If the `cmis:objectTypeId` property value is NOT in the list of `AllowedChildObjectTypeIds` of the parent-folder specified by folderId.
- `constraint` If the `controllablePolicy` attribute of the object-type definition specified by the `cmis:objectTypeId` property value is set to FALSE and at least one policy is provided.
- `constraint` If the `controllableACL` attribute of the object-type definition specified by the `cmis:objectTypeId` property value is set to FALSE and at least one ACE is provided.
- `constraint` If at least one of the specified values for permission in ANY of the ACEs does not match ANY of the permission names returned by the ACL Capabilities in the Repository Info (see section 2.1.12.3.1 Supported Permissions) and is not a CMIS basic permission.
- `nameConstraintViolation` If the repository detects a violation with the given `cmis:name` property value, the repository MAY throw this exception or chose a name which does not conflict.
- `storage` See section 2.2.1.4.2 Specific Exceptions.

2.2.4.6 createItem

Description: Creates an item object of the specified type.

2.2.4.6.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **<Array> Property `properties`:** The property values that MUST be applied to the newly-created item object.

Optional:

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

2.2.4.6.2 Outputs

- **Id `objectId`:** The id of the newly-created item.

2.2.4.6.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `constraint` If the `cmis:objectTypeId` property value is not an object-type whose base type is `cmis:item`.
- `constraint` If the value of any of the properties violates the constraints (min/max/required/length etc.) specified in the property definition in the object-type.
- `constraint` If the `cmis:objectTypeId` property value is NOT in the list of `AllowedChildObjectTypeIds` of the parent-folder specified by folderId.
- `constraint` If the `controllablePolicy` attribute of the object-type definition specified by the `cmis:objectTypeId` property value is set to FALSE and at least one policy is provided.
- `constraint` If the `controllableACL` attribute of the object-type definition specified by the `cmis:objectTypeId` property value is set to FALSE and at least one ACE is provided.
- `constraint` If at least one of the specified values for permission in ANY of the ACEs does not match ANY of the permission names returned by the ACL Capabilities in the Repository Info (see section 2.1.12.3.1 Supported Permissions) and is not a CMIS basic permission.
- `nameConstraintViolation` If the repository detects a violation with the given `cmis:name` property value, the repository MAY throw this exception or chose a name which does not conflict.
- `storage` See section 2.2.1.4.2 Specific Exceptions.

2.2.4.7 getAllowableActions

Description: Gets the list of allowable actions for an object (see section 2.2.1.2.6 Allowable Actions).

2.2.4.7.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `objectId`:** The identifier for the object.

2.2.4.7.2 Outputs

- **<Array> AllowableActions `AllowableActions`:** See section 2.2.1.2.6 Allowable Actions.

2.2.4.7.3 Exceptions Thrown & Conditions

See section 2.2.1.4.1 General Exceptions.

2.2.4.8 getObject

Description: Gets the specified information for the object.

2.2.4.8.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `objectId`:** The identifier for the object.

Optional:

- **String `filter`:** See section 2.2.1.2.1 Properties.
- **Enum `includeRelationships`:** See section 2.2.1.2.2 Relationships.
- **Boolean `includePolicyIds`:** See section 2.2.1.2.3 Policies.
- **String `renditionFilter`:** See section 2.2.1.2.4 Renditions.
- **Boolean `includeACL`:** See section 2.2.1.2.5 ACLs.
- **Boolean `includeAllowableActions`:** See section 2.2.1.2.6 Allowable Actions.

2.2.4.8.2 Outputs

- **<Array> Properties `properties`:** See section 2.2.1.2.1 Properties.
- **<Array> Relationships `relationships`:** See section 2.2.1.2.2 Relationships.
- **<Array> PolicyId `policies`:** See section 2.2.1.2.3 Policies.
- **<Array> Renditions `renditions`:** See section 2.2.1.2.4 Renditions.
- **ACL `acl`:** See section 2.2.1.2.5 ACLs.
- **AllowableActions `allowableActions`:** See section 2.2.1.2.6 Allowable Actions.

2.2.4.8.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `filterNotValid` If the property or rendition filter input parameter is not valid.

2.2.4.9 getProperties

Description: Gets the list of properties for the object.

2.2.4.9.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `objectId`:** The identifier for the object.

Optional:

- **String `filter`:** See section 2.2.1.2.1 Properties.

2.2.4.9.2 Outputs

- **<Array> Properties `properties`:** See section 2.2.1.2.1 Properties.

2.2.4.9.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `filterNotValid` If the property filter input parameter is not valid.

2.2.4.10 getObjectByPath

Description: Gets the specified information for the object.

2.2.4.10.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **String `path`:** The path to the object. See section 2.1.5.3 Paths.

Optional:

- **String `filter`:** See section 2.2.1.2.1 Properties.
- **Enum `includeRelationships`:** See section 2.2.1.2.2 Relationships.
- **Boolean `includePolicyIds`:** See section 2.2.1.2.3 Policies.
- **String `renditionFilter`:** See section 2.2.1.2.4 Renditions.
- **Boolean `includeACL`:** See section 2.2.1.2.5 ACLs.
- **Boolean `includeAllowableActions`:** See section 2.2.1.2.6 Allowable Actions.

2.2.4.10.2 Outputs

- **<Array> Properties `properties`:** See section 2.2.1.2.1 Properties.
- **<Array> Relationships `relationships`:** See section 2.2.1.2.2 Relationships.
- **<Array> PolicyId `policies`:** See section 2.2.1.2.3 Policies.
- **<Array> Renditions `renditions`:** See section 2.2.1.2.4 Renditions.
- **ACL `acl`:** See section 2.2.1.2.5 ACLs.
- **AllowableActions `allowableActions`:** See section 2.2.1.2.6 Allowable Actions.

2.2.4.10.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `filterNotValid` If the property or rendition filter input parameter is not valid.

2.2.4.11 getContentStream

Description: Gets the content stream for the specified document object, or gets a rendition stream for a specified rendition of a document or folder object.

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

2.2.4.11.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `objectId`:** The identifier for the object.

Optional:

- **Id `streamId`:** The identifier for the rendition stream, when used to get a rendition stream. For documents, if not provided then this method returns the content stream. For folders, it MUST be provided.

2.2.4.11.2 Outputs

- **<Stream> ContentStream `contentStream`:** The specified content stream or rendition stream for the object.

2.2.4.11.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `constraint` If the object specified by objectId does NOT have a content stream or rendition stream.

2.2.4.12 getRenditions

Description: Gets the list of associated renditions for the specified object. Only rendition attributes are returned, not rendition stream.

2.2.4.12.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `objectId`:** The identifier for the object.

Optional:

- **String `renditionFilter`:** See section 2.2.1.2.4 Renditions.
- **Integer `maxItems`:** See section 2.2.1.1 Paging.
- **Integer `skipCount`:** See section 2.2.1.1 Paging.

2.2.4.12.2 Outputs

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

2.2.4.12.3 Exceptions Thrown & Conditions

See section 2.2.1.4.1 General Exceptions.

2.2.4.13 updateProperties

Description: Updates properties and secondary types of the specified object.

Notes:

- A repository MAY automatically create new document versions as part of an update properties operation. Therefore, the objectId output NEED NOT be identical to the objectId input.
- Only properties whose values are different than the original value of the object SHOULD be provided.

2.2.4.13.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `objectId`:** The identifier for the object.
- **<Array> Properties `properties`:** The updated property values that MUST be applied to the object.

Optional:

- **String `changeToken`:** See section 2.2.1.3 Change Tokens.

2.2.4.13.2 Outputs

- **Id `objectId`:** The identifier for the object.
- **String `changeToken`:** See section 2.2.1.3 Change Tokens.

2.2.4.13.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `constraint` If the value of any of the properties violates the constraints (min/max/required/length etc.) specified in the property definition in the object-type.
- `nameConstraintViolation` If the repository detects a violation with the given `cmis:name` property value, the repository MAY throw this exception or chose a name which does not conflict.
- `storage` See section 2.2.1.4.2 Specific Exceptions.
- `updateConflict` See section 2.2.1.4.2 Specific Exceptions.
- `versioning` If the object is not checked out and ANY of the properties being updated are defined in their object-type definition to have an attribute value of Updatability whencheckedout.
- `versioning` The repository MAY throw this exception if the object is a non-current document version.

2.2.4.14 bulkUpdateProperties

Description: Updates properties and secondary types of one or more objects.

Notes:

- A repository MAY automatically create new document versions as part of an update properties operation. Therefore, the objectId output NEED NOT be identical to the objectId input.
- Only properties whose values are different than the original value of the object SHOULD be provided.
- This service is not atomic. If the update fails, some objects might have been updated and others might not have been updated.
- This service MUST NOT throw an exception if the update of an object fails. If an update fails, the object id of this particular object MUST be omitted from the result.

2.2.4.14.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **<Array> <Id, String> `objectIdAndChangeToken`:** The identifiers of the objects to be updated and their change tokens. Invalid object ids, for example ids of objects that don't exist, MUST be ignored.

  Change tokens are optional. See section 2.2.1.3 Change Tokens.

Optional:

- **<Array> Properties `properties`:** The updated property values that MUST be applied to the objects.
- **<Array> Id `addSecondaryTypeIds`:** A list of secondary type ids that SHOULD be added to the objects.
- **<Array> Id `removeSecondaryTypeIds`:** A list of secondary type ids that SHOULD be removed from the objects. Secondary type ids in this list that are not attached to an object SHOULD be ignored.

2.2.4.14.2 Outputs

- **<Array> <Id, Id, String> `objectIdAndChangeToken`:** A triple for each updated object composed of:

  1. The original object id. MUST always be set.
  2. The new object id if the update triggered a new version. MUST NOT be set if no new version has been created.
  3. The new change token of the object. MUST be set if the repository supports change tokens.

  Objects that have not been updated MUST NOT be returned. This service does not disclose why updates failed. Clients may call `updateProperties` for each failed object to retrieve individual exceptions.

2.2.4.14.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `invalidArgument` If the input list of object ids is empty.
- `invalidArgument` If secondary type ids are provided that don't exist in the repository.
- `constraint` If the number of objects is too high for the repository.

2.2.4.15 moveObject

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

2.2.4.15.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `objectId`:** The identifier for the object.
- **Id `targetFolderId`:** The folder into which the object is to be moved.
- **Id `sourceFolderId`:** The folder from which the object is to be moved.

2.2.4.15.2 Outputs

- **Id `objectId`:** The identifier for the object. The identifier SHOULD NOT change. If the repository has to change the id, this is the new identifier for the object.

2.2.4.15.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `invalidArgument` If the service is invoked with a missing sourceFolderId or the sourceFolderId doesn't match the specified object's parent folder (or one of the parent folders if the repository supports multifiling.).
- `constraint` If the `cmis:objectTypeId` property value of the given object is NOT in the list of AllowedChildObjectTypeIds of the parent-folder specified by targetFolderId.
- `nameConstraintViolation` If the repository detects a violation with the `cmis:name` property value, the repository MAY throw this exception or chose a name which does not conflict.
- `storage` See section 2.2.1.4.2 Specific Exceptions.
- `updateConflict` See section 2.2.1.4.2 Specific Exceptions.

2.2.4.16 deleteObject

Description: Deletes the specified object.

Notes: If the object is a PWC the checkout is discarded. See section 2.1.13.5.3 Discarding Check out.

2.2.4.16.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `objectId`:** The identifier for the object.

Optional:

- **Boolean `allVersions`:** If TRUE (default), then delete all versions of the document. If FALSE, delete only the document object specified. The repository MUST ignore the value of this parameter when this service is invoke on a non-document object or non-versionable document object.

2.2.4.16.2 Outputs

- None.

2.2.4.16.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `constraint` If the method is invoked on a folder object that contains one or more objects.
- `updateConflict` See section 2.2.1.4.2 Specific Exceptions.

2.2.4.17 deleteTree

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

Notes:

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

2.2.4.17.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `folderId`:** The identifier of the folder to be deleted.

Optional:

- **Boolean `allVersions`:** If TRUE (default), then delete all versions of all documents. If FALSE, delete only the document versions referenced in the tree. The repository MUST ignore the value of this parameter when this service is invoked on any non-document objects or non-versionable document objects.
- **Enum `unfileObjects`:** An enumeration specifying how the repository MUST process file-able child- or descendant-objects. Valid values are:

  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.

- **Boolean `continueOnFailure`:** If TRUE, then the repository SHOULD continue attempting to perform this operation even if deletion of a child- or descendant-object in the specified folder cannot be deleted.

  If FALSE (default), then the repository SHOULD abort this method when it fails to delete a single child object or descendant object.

2.2.4.17.2 Outputs

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

2.2.4.17.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `constraint` If the method is invoked on a non-folder object.
- `updateConflict` See section 2.2.1.4.2 Specific Exceptions.

2.2.4.18 setContentStream

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

Notes: A repository MAY automatically create new document versions as part of this service operations. Therefore, the objectId output NEED NOT be identical to the objectId input.

2.2.4.18.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `objectId`:** The identifier for the document object.
- **ContentStream `contentStream`:** The content stream.

Optional:

- **Boolean `overwriteFlag`:** If TRUE (default), then the repository MUST replace the existing content stream for the object (if any) with the input contentStream.

  If FALSE, then the repository MUST only set the input contentStream for the object if the object currently does not have a content stream.
- **String `changeToken`:** See section 2.2.1.3 Change Tokens.

2.2.4.18.2 Outputs

- **Id `objectId`:** The identifier for the object.
- **String `changeToken`:** See section 2.2.1.3 Change Tokens.

2.2.4.18.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `contentAlreadyExists` If the input parameter overwriteFlag is FALSE and the object already has a content stream.
- `storage` See section 2.2.1.4.2 Specific Exceptions.
- `streamNotSupported` See section 2.2.1.4.2 Specific Exceptions.
- `updateConflict` See section 2.2.1.4.2 Specific Exceptions.
- `versioning` The repository MAY throw this exception if the object is a non-current document version.

2.2.4.19 appendContentStream

Description: Appends to the content stream for the specified document object.

Notes:

- A repository MAY automatically create new document versions as part of this service method. Therefore, the objectId output NEED NOT be identical to the objectId input.
- The document may or may not have a content stream prior to calling this service. If there is no content stream, this service has the effect of setting the content stream with the value of the input `contentStream`.
- This service is intended to be used by a single client. It should support the upload of very huge content streams. The behavior is repository specific if multiple clients call this service in succession or in parallel for the same document.

2.2.4.19.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `objectId`:** The identifier for the document object.
- **ContentStream `contentStream`:** The content stream that should be appended to the existing content.

Optional:

- **Boolean `isLastChunk`:** If TRUE, then this is the last chunk of the content and the client does not intend to send another chunk. If FALSE (default), then the repository should except another chunk from the client.

  Clients SHOULD always set this parameter but repositories SHOULD be prepared that clients don't provide it.

  Repositories may use this flag to trigger some sort of content processing. For example, only if `isLastChunk` is TRUE the repsoitory could generate renditions of the content.
- **String `changeToken`:** See section 2.2.1.3 Change Tokens.

2.2.4.19.2 Outputs

- **Id `objectId`:** The identifier for the object.
- **String `changeToken`:** See section 2.2.1.3 Change Tokens.

2.2.4.19.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `constraint` If the client sends more then one "last chunk". If a repository can accept multiple chunks that are marked as "last chunk", it MAY not throw this exception.
- `storage` See section 2.2.1.4.2 Specific Exceptions.
- `streamNotSupported` See section 2.2.1.4.2 Specific Exceptions.
- `updateConflict` See section 2.2.1.4.2 Specific Exceptions.
- `versioning` The repository MAY throw this exception if the object is a non-current document version.

2.2.4.20 deleteContentStream

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

Notes: A repository MAY automatically create new document versions as part of this service method. Therefore, the obejctId output NEED NOT be identical to the objectId input.

2.2.4.20.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `objectId`:** The identifier for the document object.

Optional:

- **String `changeToken`:** See section 2.2.1.3 Change Tokens.

2.2.4.20.2 Outputs

- **Id `objectId`:** The identifier for the object.
- **String `changeToken`:** See section 2.2.1.3 Change Tokens.

2.2.4.20.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `constraint` If the object's object-type definition "contentStreamAllowed" attribute is set to "required".
- `storage` See section 2.2.1.4.2 Specific Exceptions.
- `updateConflict` See section 2.2.1.4.2 Specific Exceptions.
- `versioning` The repository MAY throw this exception if the object is a non-current document version.

2.2.5 Multi-filing Services

The Multi-filing services are supported only if the repository supports the multifiling or unfiling optional capabilities (`capabilityMultifiling`). 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.

2.2.5.1 addObjectToFolder

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

2.2.5.1.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `objectId`:** The identifier for the object.
- **Id `folderId`:** The folder into which the object is to be filed.

Optional:

- **Boolean `allVersions`:** Add all versions of the object to the folder if the repository supports version-specific filing. Defaults to TRUE.

2.2.5.1.2 Outputs

- None.

2.2.5.1.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `constraint` If the `cmis:objectTypeId` property value of the given object is NOT in the list of AllowedChildObjectTypeIds of the parent-folder specified by folderId.

2.2.5.2 removeObjectFromFolder

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

2.2.5.2.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `objectId`:** The identifier for the object.

Optional:

- **Id `folderId`:** The folder from which the object is to be removed.

  If no value is specified, then the repository MUST remove the object from all folders in which it is currently filed.

2.2.5.2.2 Outputs

- None.

2.2.5.2.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `constraint` If the repository does not support unfiled objects (`capabilityUnfiling` = false) and this operation would unfile the object from the last folder.

2.2.6 Discovery Services

The Discovery Services are used to search for query-able objects within the repository.

2.2.6.1 query

Description: Executes a CMIS query statement against the contents of the repository.

2.2.6.1.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **String `statement`:** CMIS query to be executed. See section 2.1.14 Query.
  Note: The AtomPub and the Browser Binding also use the name `q`.

Optional:

- **Boolean `searchAllVersions`:** If TRUE, then the repository MUST include latest and non-latest versions of document objects in the query search scope.

  If FALSE (default), then the repository MUST only include latest versions of documents in the query search scope.

  If the repository does not support the optional `capabilityAllVersionsSearchable` capability, then this parameter value MUST be set to FALSE.
- **Enum `includeRelationships`:** See section 2.2.1.2.2 Relationships.

  For query statements where the SELECT clause contains properties from only one virtual table reference (i.e. referenced object-type), any value for this enum may be used. If the SELECT clause contains properties from more than one table, then the value of this parameter MUST be `none`. Defaults to `none`.
- **String `renditionFilter`:** See section 2.2.1.2.4 Renditions.

  If the SELECT clause contains properties from more than one table, then the value of this parameter MUST not be set.
- **Boolean `includeAllowableActions`:** See section 2.2.1.2.6 Allowable Actions.

  For query statements where the SELECT clause contains properties from only one virtual table reference (i.e. referenced object-type), any value for this parameter may be used. If the SELECT clause contains properties from more than one table, then the value of this parameter MUST be "FALSE". Defaults to FALSE.
- **Integer `maxItems`:** See section 2.2.1.1 Paging.
- **Integer `skipCount`:** See section 2.2.1.1 Paging.

2.2.6.1.2 Outputs

- **<Array> Object `queryResults`:** The set of results for the query. (See section 2.2.1.2.2 Relationships.)

  Each object result MUST include the following elements if they are requested:

  Relationships See section 2.2.1.2.2 Relationships.

  Renditions See section 2.2.1.2.4 Renditions.

  AllowableActions See section 2.2.1.2.6 Allowable Actions.
- **Boolean `hasMoreItems`:** See section 2.2.1.1 Paging.
- **Integer `numItems`:** See section 2.2.1.1 Paging.

2.2.6.1.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `invalidArgument` If the select clause includes properties from more than a single type reference and if `includeRelationships` is something other than "none" or `includeAllowableActions` is specified as TRUE.

2.2.6.2 getContentChanges

Description: Gets a list of content changes. This service is intended to be used by search crawlers or other applications that need to efficiently understand what has changed in the repository. See section 2.1.15 Change Log.

Notes:

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

2.2.6.2.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.

Optional:

- **String `changeLogToken`:** If specified, then the repository MUST return the change event corresponding to the value of the specified change log token as the first result in the output.

  If not specified, then the repository MUST return the first change event recorded in the change log.

- **Boolean `includeProperties`:** If TRUE, then the repository MUST include the updated property values for "updated" change events if the repository supports returning property values as specified by `capbilityChanges`.

  If FALSE (default), then the repository MUST NOT include the updated property values for "updated" change events. The single exception to this is that the property `cmis:objectId` MUST always be included.

- **Boolean `includePolicyIds`:** If TRUE, then the repository MUST include the ids of the policies applied to the object referenced in each change event, if the change event modified the set of policies applied to the object.

  If FALSE (default), then the repository MUST not include policy information.

- **Boolean `includeACL`:** See section 2.2.1.2.5 ACLs.

- **Integer `maxItems`:** See section 2.2.1.1 Paging.

2.2.6.2.2 Outputs

- **<Array> ChangeEvents `changeEvents`:** A collection of CMIS objects that MUST include the information as specified in 2.1.15.4 Change Event. Each result MUST include the following elements if they are requested:

  policyIds The ids of policies applied to the object referenced in the change event.

  ACL The ACL applied to the object reference in the change event.

- **String `latestChangeLogToken`:** The change log token corresponding to the last change event in changeEvents.

- **Boolean `hasMoreItems`:** See section 2.2.1.1 Paging.

- **Integer `numItems`:** See section 2.2.1.1 Paging.

2.2.6.2.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `invalidArgument` if the event corresponding to the change log token provided as an input parameter is no longer available in the change log. (E.g. because the change log was truncated).

2.2.7 Versioning Services

The Versioning services are used to navigate or update a document version series. See section 2.1.13 Versioning.

2.2.7.1 checkOut

Description: Create a private working copy (PWC) of the document. See section 2.1.13.5.1 Checkout.

2.2.7.1.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `objectId`:** The identifier for the document version object that should be checked out.

2.2.7.1.2 Outputs

- **Id `objectId`:** The identifier for the "Private Working Copy" document.
- **Boolean `contentCopied`:** TRUE if the content stream of the Private Working Copy is a copy of the contentStream of the document that was checked out.

  FALSE if the content stream of the Private Working Copy is "not set".

2.2.7.1.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `constraint` if the document's object-type definition's `versionable` attribute is FALSE.
- `storage` See section 2.2.1.4.2 Specific Exceptions.
- `updateConflict` See section 2.2.1.4.2 Specific Exceptions.
- `versioning` The repository MAY throw this exception if the object is a non-current document version.

2.2.7.2 cancelCheckOut

Description: Reverses the effect of a check-out (`checkOut`). Removes the Private Working Copy of the checked-out document, allowing other documents in the version series to be checked out again. If the private working copy has been created by `createDocument`, `cancelCheckOut` MUST delete the created document. See section 2.1.13.5.3 Discarding Check out.

2.2.7.2.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `objectId`:** The identifier of the Private Working Copy.

2.2.7.2.2 Outputs

- None.

2.2.7.2.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `constraint` if the document's object-type definition's `versionable` attribute is FALSE.
- `updateConflict` See section 2.2.1.4.2 Specific Exceptions.
- `versioning` The repository MAY throw this exception if the object is a non-current document version.

2.2.7.3 checkIn

Description: Checks-in the Private Working Copy document. See section 2.1.13.5.4 Checkin.

Notes:

- For repositories that do NOT support the optional `capabilityPWCUpdatable` capability, the properties and contentStream input parameters MUST be provided on the `checkIn` service for updates to happen as part of checkIn.
- Each CMIS protocol binding MUST specify whether the checkin service MUST always include all updatable properties, or only those properties whose values are different than the original value of the object.

2.2.7.3.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `objectId`:** The identifier for the Private Working Copy.

Optional:

- **Boolean `major`:** TRUE (default) if the checked-in document object MUST be a major version.

  FALSE if the checked-in document object MUST NOT be a major version but a minor version.
- **<Array> Property `properties`:** The property values that MUST be applied to the checked-in document object.
- **<contentStream> `contentStream`:** The content stream that MUST be stored for the checked-in document object. The method of passing the contentStream to the server and the encoding mechanism will be specified by each specific binding. MUST be required if the type requires it.
- **String `checkinComment`:** See section 2.1.13.6 Versioning Properties on Document Objects.
- **<Array> Id `policies`:** A list of policy ids that MUST be applied to the newly-created document object.
- **<Array> ACE `addACEs`:** A list of ACEs that MUST be added to the newly-created document object.
- **<Array> ACE `removeACEs`:** A list of ACEs that MUST be removed from the newly-created document object.

2.2.7.3.2 Outputs

- **Id `objectId`:** The id of the checked-in document.

2.2.7.3.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `constraint` if the object is not a Private Working Copy.
- `storage` See section 2.2.1.4.2 Specific Exceptions.
- `updateConflict` See section 2.2.1.4.2 Specific Exceptions.
- `streamNotSupported` See section 2.2.1.4.2 Specific Exceptions.

2.2.7.4 getObjectOfLatestVersion

Description: Get the latest document object in the version series.

2.2.7.4.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `versionSeriesId`:** The identifier for the version series.

Optional:

- **Boolean `major`:** If TRUE, then the repository MUST return the properties for the latest major version object in the version series.

  If FALSE (default), the repository MUST return the properties for the latest (major or non-major) version object in the version series.
- **String `filter`:** See section 2.2.1.2.1 Properties.
- **Enum `includeRelationships`:** See section 2.2.1.2.2 Relationships.
- **Boolean `includePolicyIds`:** See section 2.2.1.2.3 Policies.
- **String `renditionFilter`:** See section 2.2.1.2.4 Renditions.
- **Boolean `includeACL`:** See section 2.2.1.2.5 ACLs.
- **Boolean `includeAllowableActions`:** See section 2.2.1.2.6 Allowable Actions.

2.2.7.4.2 Outputs

- **<Array> Properties `properties`:** See section 2.2.1.2.1 Properties.
- **<Array> Relationships `relationships`:** See section 2.2.1.2.2 Relationships.
- **<Array> PolicyId `policies`:** See section 2.2.1.2.3 Policies.
- **<Array> Renditions `renditions`:** See section 2.2.1.2.4 Renditions.
- **ACL `acl`:** See section 2.2.1.2.5 ACLs.
- **AllowableActions `allowableActions`:** See section 2.2.1.2.6 Allowable Actions.

2.2.7.4.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `filterNotValid` If the property or rendition filter input parameter is not valid.
- `objectNotFound` If the input parameter major is TRUE and the version series contains no major versions.

2.2.7.5 getPropertiesOfLatestVersion

Description: Get a subset of the properties for the latest document object in the version series.

2.2.7.5.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `versionSeriesId`:** The identifier for the version series.

Optional:

- **Boolean `major`:** If TRUE, then the repository MUST return the properties for the latest major version object in the version series.

  If FALSE (default), the repository MUST return the properties for the latest (major or non-major) version object in the version series.
- **String `filter`:** See section 2.2.1.2.1 Properties.

2.2.7.5.2 Outputs

- **<Array> Properties `properties`:** See section 2.2.1.2.1 Properties.

2.2.7.5.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `filterNotValid` If the property filter input parameter is not valid.
- `objectNotFound` If the input parameter major is TRUE and the version series contains no major versions.

2.2.7.6 getAllVersions

Description: Returns the list of all document objects in the specified version series, sorted by `cmis:creationDate` descending.

Notes: If a Private Working Copy exists for the version series and the caller has permissions to access it, then it MUST be returned as the first object in the result list.

2.2.7.6.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `versionSeriesId`:** The identifier for the version series.

Optional:

- **String `filter`:** See section 2.2.1.2.1 Properties.
- **Boolean `includeAllowableActions`:** See section 2.2.1.2.6 Allowable Actions.

2.2.7.6.2 Outputs

- **<Array> ObjectResults `objects`:** A list of document objects in the specified version series. Each object result MUST include the following elements if they are requested:

  Properties See section 2.2.1.2.1 Properties.

  AllowableActions See section 2.2.1.2.6 Allowable Actions.

2.2.7.6.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `filterNotValid` If the property filter input parameter is not valid.

2.2.8 Relationship Services

The Relationship Services are used to retrieve the dependent relationship objects associated with an independent object.

2.2.8.1 getObjectRelationships

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

2.2.8.1.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `objectId`:** The identifier of the object.

Optional:

- **Boolean `includeSubRelationshipTypes`:** If TRUE, then the repository MUST return all relationships whose object-types are descendant-types of the object-type specified by the `typeId` parameter value as well as relationships of the specified type.

  If FALSE (default), then the repository MUST only return relationships whose object-types is equivalent to the object-type specified by the `typeId` parameter value.

  If the `typeId` input is not specified, then this input MUST be ignored.
- **Enum `relationshipDirection`:** An enumeration specifying whether the repository MUST return relationships where the specified object is the source of the relationship, the target of the relationship, or both. Valid values are:

  source (default) The repository MUST return only relationship objects where the specified object is the source object.

  target The repository MUST return only relationship objects where the specified object is the target object.

  either The repository MUST return relationship objects where the specified object is either the source or the target object.
- **Id `typeId`:** If specified, then the repository MUST return only relationships whose object-type is of the type specified. See also parameter `includeSubRelationshipTypes`.

  If not specified, then the repository MUST return relationship objects of all types.
- **Integer `maxItems`:** See section 2.2.1.1 Paging.
- **Integer `skipCount`:** See section 2.2.1.1 Paging.
- **String `filter`:** See section 2.2.1.2.1 Properties.
- **Boolean `includeAllowableActions`:** See section 2.2.1.2.6 Allowable Actions.

2.2.8.1.2 Outputs

- **<Array> Object `objects`:** A list of the relationship objects. Each object result MUST include the following elements if they are requested:

  Properties See section 2.2.1.2.1 Properties.

  AllowableActions See section 2.2.1.2.6 Allowable Actions.
- **Boolean `hasMoreItems`:** See section 2.2.1.1 Paging.
- **Integer `numItems`:** See section 2.2.1.1 Paging.

2.2.8.1.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `filterNotValid` If the property or rendition filter input parameter is not valid.

2.2.9 Policy Services

The Policy Services are used to apply or remove a policy object to a `controllablePolicy` object.

2.2.9.1 applyPolicy

Description: Applies a specified policy to an object.

2.2.9.1.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `policyId`:** The identifier for the policy to be applied.
- **Id `objectId`:** The identifier of the object.

2.2.9.1.2 Outputs

- None.

2.2.9.1.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `constraint` The repository MUST throw this exception if the specified object's object-type definition's attribute for `controllablePolicy` is FALSE.

2.2.9.2 removePolicy

Description: Removes a specified policy from an object.

2.2.9.2.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `policyId`:** The identifier for the policy to be removed.
- **Id `objectId`:** The identifier of the object.

2.2.9.2.2 Outputs

- None.

2.2.9.2.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.

2.2.9.3 getAppliedPolicies

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

2.2.9.3.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `objectId`:** The identifier of the object.

Optional:

- **String `filter`:** See section 2.2.1.2.1 Properties.

2.2.9.3.2 Outputs

- **<Array> Object `objects`:** A list of the policy objects.

2.2.9.3.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `filterNotValid` If the property filter input parameter is not valid.

2.2.10 ACL Services

The ACL Services are used to discover and manage Access Control Lists.

2.2.10.1 applyACL

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

Notes: This service MUST be supported by the repository, if the optional capability `capabilityACL` is `manage`.

How ACEs are added or removed to or from the object is repository specific – with respect to the `ACLPropagation` parameter.

Some ACEs that make up an object's ACL may not be set directly on the object, but determined in other ways, such as inheritance. A repository MAY merge the ACEs provided with the ACEs of the ACL already applied to the object (i.e. the ACEs provided MAY not be completely added or removed from the effective ACL for the object).

2.2.10.1.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `objectId`:** The identifier of the object.

Optional:

- **<Array> AccessControlEntryType `addACEs`:** The ACEs to be added.
- **<Array> AccessControlEntryType `removeACEs`:** The ACEs to be removed.
- **Enum `ACLPropagation`:** Specifies how ACEs should be handled. Valid values are:

  objectonly ACEs must be applied without changing the ACLs of other objects.

  propagate ACEs must be applied by propagate the changes to all "inheriting" objects.

  repositorydetermined (default) Indicates that the client leaves the behavior to the repository.

2.2.10.1.2 Outputs

- **<Array> AccessControlEntryType `acl`:** The list of access control entries of the ACL for the object after the new ACEs have been applied. The repository MAY return an empty list if the user has no permissions to access the ACL of this object anymore after calling this service.
- **Boolean `exact`:** An indicator that the ACL returned fully describes the permission for this object. That is, there are no other security constraints applied to this object. Not provided defaults to FALSE.

2.2.10.1.3 Exceptions Thrown & Conditions

- See section 2.2.1.4.1 General Exceptions.
- `constraint` If the specified object's object-type definition's attribute for `controllableACL` is FALSE.
- `constraint` If the value for `ACLPropagation` does not match the values as returned by the ACL Capabilities in the Repository Info. (See section 2.1.12.3 ACL Capabilities.)
- `constraint` If at least one of the specified values for permission in ANY of the ACEs does not match ANY of the permission names returned by the ACL Capabilities in the Repository Info (see section 2.1.12.3.1 Supported Permissions) and is not a CMIS basic permission.

2.2.10.2 getACL

Description: Get the ACL currently applied to the specified object.

Notes: This service MUST be supported by the repository, if the optional capability `capabilityACL` is `discover` or `manage`. A client MUST NOT assume that the returned ACEs can be applied via `applyACL`.

2.2.10.2.1 Inputs

Required:

- **Id `repositoryId`:** The identifier for the repository.
- **Id `objectId`:** The identifier of the object.

Optional:

- **Boolean `onlyBasicPermissions`:** See section 2.1.12 Access Control. The repository SHOULD make a best effort to fully express the native security applied to the object.
  - TRUE (default) indicates that the client requests that the returned ACL be expressed using only the CMIS basic permissions.
  - FALSE indicates that the server may respond using either solely CMIS basic permissions, or repository specific permissions or some combination of both.

2.2.10.2.2 Outputs

- **<Array> AccessControlEntryType `acl`:** The list of access control entries of the ACL for the object. The repository MAY return an empty list if the user has no permissions to access the ACL of this object.
- **Boolean `exact`:** An indicator that the ACL returned fully describes the permission for this object. That is, there are no other security constraints applied to this object. Not provided defaults to FALSE.

2.2.10.2.3 Exceptions Thrown & Conditions

See section 2.2.1.4.1 General Exceptions.

3 AtomPub Binding

3.1 Overview

This binding is based upon the Atom ([RFC4287]) and Atom Publishing Protocol ([RFC5023]). Implementations of CMIS MUST be compliant with [RFC4287] and [RFC5023].

In this binding, the client interacts with the repository by acquiring the service document. The client will request the service document by the URI provided by the vendor. The client will then choose a CMIS collection, and then start accessing the repository by following the references in the returned documents.

This binding consists of a service document specifying at least CMIS service collections, Atom collections, feeds and entry documents. CMIS extends the Atom and AtomPub documents utilizing the Atom and AtomPub extension mechanism. CMIS also leverages link tags to specify additional resources related to the requested resource.

When requesting a resource, optional parameters may be specified to change default behavior via query parameters.

3.1.1 Namespaces

This specification uses the following namespaces and prefixes when referring to XML or XML schema elements in the text or examples:

- CMIS-Core: `http://docs.oasis-open.org/ns/cmis/core/200908/`
  Prefix: `cmis`
- CMIS-RestAtom: `http://docs.oasis-open.org/ns/cmis/restatom/200908/`
  Prefix: `cmisra`
- Atom : `http://www.w3.org/2005/Atom`
  Prefix: `atom`
- AtomPub: `http://www.w3.org/2007/app`
  Prefix: `app`

3.1.2 Authentication

Authentication SHOULD be handled by the transport protocol. Please see AtomPub [RFC5023] section 14.

If the provided credentials are incorrect or unknown or entirely missing, a repository MAY return the HTTP status code 403 (Forbidden) instead of the HTTP status code 401 (Unauthorized). This may prevent attacks against the Browser Binding. See also section 5.2.9 Authentication.

3.1.3 Response Formats

The client can specify, in HTTP the Accept header, which formats are acceptable to the client. With this mechanism the client can choose which response format the CMIS implementation should respond with. The CMIS compliant implementation MUST support the appropriate Media Types specified in this document.

3.1.4 Optional Arguments

The binding supports adding optional parameters to CMIS resources to modify the default behavior. CMIS implementations MUST support arguments being specified as HTTP query string parameters.

Names and valid values for HTTP query string parameters are as described in the appropriate CMIS Service descriptions (see section 2.2 Services). Valid values of enumeration types are also represented in the CMIS Core XML Schema.

3.1.5 Errors and Exceptions

Exceptions MUST be mapped to the appropriate HTTP status code.

Repositories SHOULD provide sufficient information in the body of the HTTP response for a user to determine corrective action.

See section 3.2.3 HTTP Status Codes for more information.

3.1.6 Renditions

Each rendition included in a CMIS AtomPub response is represented as an Atom link with a relationship alternate.

The following attributes SHOULD be included on the link element:

href URI to the rendition content stream

type The Media Type of the rendition

cmisra:renditionKind The Rendition Kind for the rendition

The following attributes MAY be included:

title The filename (or name property if object) of rendition

length The length of the rendition in bytes

3.1.7 Content Streams

The content stream for a document SHOULD be referenced by the content `src` attribute as well as the `edit-media` link relation. A CMIS Repository MAY use different URIs for both content `src` attribute and the `edit-media` link relation for the same content stream.

The following attributes SHOULD be included on the link element:

href URI to the content stream

type The Media Type of the content stream

3.1.8 Paging of Feeds

For paging, please see the AtomPub RFC. CMIS leverages first, next, previous, and last link relations to express paging.

If the repository can include the number of items (`numItems` in CMIS Domain Model) in a feed, then the repository SHOULD include the `cmisra:numItems` extension element in the feed.

3.1.9 Services not Exposed

The following services are not exposed in this binding:

- `getRenditions`: This is exposed as part of `getObject`.
- `getProperties`: This is exposed as part of `getObject`.
- `getPropertiesOfLatestVersion`: This is exposed as part of `getObjectOfLatestVersion`.
- `createDocumentFromSource`: This is not exposed in this binding except as the client saving the resource and resubmitting it without the `cmis:objectId`.
- Adding and removing ACEs on Create or CheckIn operations: This is not possible with the AtomPub binding. The Create or CheckIn operation must be performed first. Then the dependent resource, ACL, must be retrieved and updated.
- `setContentStream`: This does not return the new object id and change token as specified by the domain model. This is not possible without introducing a new HTTP header.
- `deleteContentStream`: This does not return the new object id and change token as specified by the domain model. This is not possible without introducing a new HTTP header.
- `checkOut`: This does not return whether or not content was copied. This is not possible without introducing a new HTTP header. Clients may check content related properties such as the `cmis:contentStreamLength` property if the Private Working Copy has a content stream.
- `deleteTree`: This does not return which objects have not been deleted. If the deletion of a folder fails, the client can call `getChildren` or `getDescendants` to determine which objects could not been deleted.

3.1.9.1 removePolicy

This service is exposed from the domain model in the AtomPub binding. However, it is not as straightforward. To remove a policy from an object, one must do:

- Get the object.
- Fetch the policies collection of the object.
- Walk through the feed and find the policy object where `cmis:objectId` == policy id to remove.
- Get the self link of this policy object.
- Perform a DELETE on this URL.

This is also the only case in the AtomPub Binding where an URI in a collection (policies) is specific to that collection.

3.2 HTTP

3.2.1 HTTP Range

Repositories MAY support HTTP Range requests on content streams.

3.2.2 HTTP OPTIONS Method

The repository MAY support the HTTP OPTIONS method on all the resources defined in this specification. If the repository supports OPTIONS, then the repository MUST at least return the HTTP methods specified for that resource in the Allow header.

3.2.3 HTTP Status Codes

Please see the HTTP specification for more information on the HTTP status codes. These are provided as guidance from the HTTP specification. If any conflict arises, the HTTP specification is authoritative.

3.2.3.1 General CMIS Exceptions

The following listing defines the HTTP status codes that repositories MUST return for the various common exceptions defined in CMIS Domain Model.

CMIS Services Exception	HTTP Status Code
General Exceptions	
`invalidArgument`	400
`notSupported`	405
`objectNotFound`	404
`permissionDenied`	403
`runtime`	500
Specific Exceptions	
`constraint`	409
`contentAlreadyExists`	409
`filterNotValid`	400
`nameConstraintViolation`	409
`storage`	500
`streamNotSupported`	403
`updateConflict`	409
`versioning`	409

3.2.3.2 Notable HTTP Status Codes

415 Unsupported Media Type
When a document is POST'ed to a collection that does not support the media type of the document, this status code MUST be returned

422 Unprocessable Entity (Defined in [RFC4918] Section 11.2)
When a request has been POST'ed but cannot be processed, this status code MUST be returned.

Please see [RFC2616] Section 10 for more information.

3.3 Media Types

CMIS introduces new media types for:

- a CMIS Query document (`application/cmisquery+xml`)
- a CMIS AllowableActions document (`application/cmisallowableactions+xml`)
- an Atom document (Entry or Feed) with any CMIS Markup (`application/cmisatom+xml`)
- an Atom Feed document with CMIS Hierarchy extensions (`application/cmistree+xml`)
- a CMIS ACL document (`application/cmisacl+xml`)

In addition to those media types specified by CMIS, CMIS also leverages these media types:

- AtomPub Service (`application/atomsvc+xml`)
- Atom Entry (`application/atom+xml;type=entry`)
- Atom Feed (`application/atom+xml;type=feed`)

3.3.1 CMIS Atom

Media Type:	`application/cmisatom+xml`
Starting tag:	`atom:feed` or `atom:entry`
Type Parameters:	`type`: the semantics of the type parameter MUST be the same as the media type parameter for Atom documents.

This allows clients to differentiate between repositories that require Atom media type with CMIS extensions (`application/cmisatom+xml`) for creation and repositories that allow generic Atom media type without CMIS extensions (`application/atom+xml`).

This is only used for CMIS repositories to advertise what media types are accepted for adding to a collection (e.g., creating resources in a collection). As such CMIS does not require specifying whether an Atom feed has CMIS markup. It is included to be consistent with the Atom media type.

All feeds and entries from a CMIS repository MUST utilize the Atom media type for exposing Atom resources. Please see the individual resources for more information on the media type. This provides the interoperability with Atom clients.

Example:
Request: `atompub/getObject-request.log`
Response: `atompub/getObject-response.log`

3.3.2 CMIS Query

Media Type:	`application/cmisquery+xml`
Starting tag:	`cmis:query`

This document contains the representation of a query to be executed in a CMIS repository.

Example:
Request: `atompub/doQuery-request.log`
Response: `atompub/doQuery-response.log`

3.3.3 CMIS Allowable Actions

Media Type:	`application/cmisallowableactions+xml`
Starting tag:	`cmis:allowableActions`

This document contains the representation of the allowable actions the user may perform on the referenced object.

Example:
Request: `atompub/getAllowableActions-request.log`
Response: `atompub/getAllowableActions-response.log`

3.3.4 CMIS Tree

Media Type:	`application/cmistree+xml`
Starting tag:	`atom:feed`

This document is an Atom feed (`application/atom+xml;type=feed`) with CMIS markup to nest a hierarchy.

Please see section 3.5.2.1 Hierarchical Atom Entries.

Example:
Request: `atompub/getDescendants-request.log`
Response: `atompub/getDescendants-response.log`

Note: This media type is used on links with relation `down` (see section 3.4.3.2 Hierarchy Navigation Internet Draft Link Relations). When the individual resources are returned by the CMIS repository they will use the Atom media type (`application/atom+xml`)

3.3.5 CMIS ACL

Media Type:	`application/cmisacl+xml`
Starting tag:	`cmis:acl`

This document specifies an Access Control List based on the schema in CMIS Domain Model.

Example:
Request: `atompub/getAcl-request.log`
Response: `atompub/getAcl-response.log`

3.4 Atom Extensions for CMIS

3.4.1 Atom Element Extensions

3.4.1.1 AtomPub Workspace

3.4.1.1.1 cmisra:collectionType

This element is included inside the `app:collection` element. This specifies the CMIS collection type.

3.4.1.1.2 cmisra:repositoryInfo

This element is included inside the `app:workspace` element. This specifies information about the CMIS repository.

3.4.1.1.3 cmis:uritemplate

This element is included inside the `app:workspace` element. This specifies information about URI templates

3.4.1.2 Atom Feed

3.4.1.2.1 cmisra:numItems

This element is included inside the `atom:feed` element. This specifies the number of items in the feed.

3.4.1.3 Atom Entry

3.4.1.3.1 cmisra:children

This element is included inside the `atom:entry` element. This includes the children of the Atom entry. This element MUST include an `atom:feed` element.

3.4.1.3.2 cmisra:object

This element is included inside the `atom:entry` element for CMIS document, folder, relationship, policy, and item objects. This specifies the CMIS object information for the Atom entry.

3.4.1.3.3 cmisra:pathSegment

This element is included inside the `atom:entry` element. This specifies the pathSegment for this object in the folder representing the feed.

3.4.1.3.4 cmisra:relativePathSegment

This element is included inside the `atom:entry` element. This specifies the relative pathSegment for the object in that particular folder. This MUST be used only inside an object parents feed.

3.4.1.3.5 cmisra:type

This element is included inside the `atom:entry` element for CMIS Type Definitions. This specifies the type definition the Atom entry represents.

3.4.1.3.6 cmisra:content

This element specifies the content of the `atom:entry` element. The content is base64 encoded in the `base64` element. The elements of a `cmisra:content` element are:

cmisra:mediaType This contains the media type of the content as described by [RFC4288].

cmisra:base64 This contains the base64 content of the file

This element MUST take precedence over `atom:content` on submission of an Atom entry to a repository.

A repository MUST use the `atom:content` element to return back to the client the content of the document.

Note: This is required when the client has an XML document stored that might not be well formed and thus would not be able to be included inside `atom:content` element.

3.4.1.3.7 cmisra:bulkUpdate

This element is included inside the `atom:entry` element. It specifics bulk update data.

See the `bulkUpdateProperties` service for details.

3.4.2 Attributes

These attributes are in the CMIS RestAtom namespace (`cmisra`).

3.4.2.1 cmisra:id

This attribute is used on the `atom:link` element to specify the CMIS id of the resource. This attribute SHOULD be on all link relations that point to a CMIS object.

This attribute MAY also be on `cmisra:type`. The value of the attribute on `cmis:type` MUST be the same as the type definition id.

Example

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<atom:link xmlns:cmis="http://docs.oasis-open.org/ns/cmis/core/200908/"
  xmlns:cmism="http://docs.oasis-open.org/ns/cmis/messaging/200908/"
  xmlns:atom="http://www.w3.org/2005/Atom"
  xmlns:app="http://www.w3.org/2007/app"
  xmlns:cmisra="http://docs.oasis-open.org/ns/cmis/restatom/200908/"
  type="application/atom+xml;type=feed"
  rel="down"
  href="http://example.com/rep1/children/e170da7d-d322-472d-b1eb-67bdb1ec18ca/1"
  cmisra:id="e170da7d-d322-472d-b1eb-67bdb1ec18ca"/>
```

3.4.2.2 cmisra:renditionKind

This attribute is used on the `atom:link` element with relation alternate to specify the renditionKind of the resource. This attribute SHOULD be on all link elements with relation alternate that are a CMIS rendition.

Example

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<atom:link xmlns:cmis="http://docs.oasis-open.org/ns/cmis/core/200908/"
  xmlns:cmism="http://docs.oasis-open.org/ns/cmis/messaging/200908/"
  xmlns:atom="http://www.w3.org/2005/Atom"
  xmlns:app="http://www.w3.org/2007/app"
  xmlns:cmisra="http://docs.oasis-open.org/ns/cmis/restatom/200908/"
  type="text/html"
  rel="alternate"
```

```
  href="http://example.com/rep1/rendition/e170da7d-d322-472d-b1eb-67bdb1ec18ca/1"
  cmisra:renditionKind="cmis:thumbnail"/>
```

3.4.3 CMIS Link Relations

The listing below outlines the different link relation types in CMIS. This is in addition to the link relations specified by Atom and Atom Publishing Protocol. The registry for link relations is located at http://www.iana.org/assignments/link-relations/link-relations.xhtml.

The link element with a specified relation MUST be included if client can perform the operation. The repository SHOULD omit the link relation if the operation is not available. The operation may not be available due to a variety of reasons such as access control, administrative policies, or other mechanisms.

Links may have the following attribute in addition to the ones specified by Atom and Atom Publishing Protocol:

cmisra:id Specifies the CMIS Id of the resource referenced by the link. Repositories SHOULD include this attribute for elements such as `atom:link` that point to CMIS resources that have an id.

3.4.3.1 Existing Link Relations

Existing link relations should be used where appropriate by the implementation. In addition, the following link relations are leveraged for the CMIS specification:

`self`

- This link relation provides the URI to retrieve this resource again.
- Service: The appropriate service that generated the Atom entry or feed.
- Resources: All except AllowableActions, ACL and Content Streams

`service`

- The service link relation when provided on a CMIS resource MUST point to an AtomPub service document with only one workspace element. This workspace element MUST represent the repository containing that resource.
- Media Type: `application/atomsvc+xml`
- Resources: All except AllowableActions, ACL and Content Streams

`describedby`

- When used on a CMIS resource, this link relation MUST point to an Atom entry that describes the type of that resource.
- Service: `getTypeDefinition` on specified object
- Media Type: `application/atom+xml;type=entry`
- Resources: CMIS document, CMIS folder, CMIS relationship, CMIS policy, CMIS item objects and CMIS types

`via`

- When used on an Atom feed document, this link relation MUST point to the Atom entry representing the CMIS resource from whom this feed is derived.
- Media Type: `application/atom+xml;type=entry`
- Resources: All CMIS Feeds and Collections

edit-media

- When used on a CMIS document resource, this link relation MUST point to the URI for content stream of the CMIS document. This URI MUST be used to set or delete the content stream. This URI MAY be used to retrieve the content stream for the document.
- Service: `setContentStream` (PUT), `appendContentStream` (PUT), `deleteContentStream` (DELETE)
- Media Type: Specific to resource
- Resources: CMIS document

edit

- When used on a CMIS resource, this link relation MUST provide an URI that can be used with the HTTP PUT method to modify the `atom:entry` for the CMIS resource.
- Service: `getObject` (GET), `updateProperties` (PUT)
- Media Type: `application/atom+xml;type=entry`
- Resources: CMIS documents, CMIS folders, CMIS relationships, CMIS policies, and CMIS items

alternate

- This is used to express renditions on a CMIS resource. See section 2.1.4.2 Renditions.
- Service: `getContentStream` for specified rendition
- Resources: CMIS documents and folders

first

- This is used for paging. Please see the AtomPub specification.
- Media Type: `application/atom+xml;type=feed`
- Resources: All Feeds

previous

- This is used for paging. Please see the AtomPub specification.
- Media Type: `application/atom+xml;type=feed`
- Resources: All Feeds

next

- This is used for paging. Please see the AtomPub specification.
- Media Type:`application/atom+xml;type=feed`
- Resources: All Feeds

last

- This is used for paging. Please see the AtomPub specification.
- Media Type: `application/atom+xml;type=feed`
- Resources: All Feeds

Please see http://www.iana.org/assignments/link-relations/link-relations.xhtml for more information on these link relations.

3.4.3.2 Hierarchy Navigation Internet Draft Link Relations

CMIS leverages the following link relations:

up

- Service: getFolderParent, getObjectParents, getTypeDefinition, getObject
- Media Type: application/atom+xml;type=feed, application/atom+xml;type=entry
- Resources: CMIS documents, folders, policies, items, type definitions, folder children, folder descendants, folder tree, type children, type descendants
- This link relation MUST not be included on CMIS base type definitions or the CMIS root folder

down

- Service: getChildren, getDescendants, getTypeChildren, getTypeDescendants
- Media Type:
  - For children: application/atom+xml;type=feed
  - For descendants: application/cmistree+xml
  - The descendants feed resource when retrieved from the CMIS repository will use the Atom Feed Media Type (application/atom+xml;type=feed)
- Resources: CMIS folders, types

3.4.3.3 Versioning Internet Draft Link Relations

CMIS leverages the following link relations from the Internet Draft:

version-history

- Service: getAllVersions
- Media Type: application/atom+xml;type=feed
- Resources: CMIS documents

current-version

- Service: getObjectOfLatestVersion (major == FALSE)
- Media Type: application/atom+xml;type=entry
- Resources: CMIS documents

working-copy

- Service: getObject for Private Working Copy specified by cmis:versionSeriesCheckedOutId property
- Media Type: application/atom+xml;type=entry
- Resources: CMIS documents if a PWC exists

3.4.3.4 CMIS Specific Link Relations

CMIS defines the following link relations:

http://docs.oasis-open.org/ns/cmis/link/200908/allowableactions

- This link relation MUST point to a resource containing a CMIS AllowableActions document for the CMIS resource containing this link relation.
- Service: `getAllowableActions`
- Media Type: `application/cmisallowableactions+xml`
- Resources: CMIS documents, folders, policies, relationships, and items

`http://docs.oasis-open.org/ns/cmis/link/200908/relationships`

- This link relation MUST point to a resource containing an Atom feed of CMIS relationship resources for the CMIS resource containing this link relation.
- Service: `getObjectRelationships`
- Media Type: `application/atom+xml;type=feed`
- Resources: CMIS documents, folders, policies, and items

`http://docs.oasis-open.org/ns/cmis/link/200908/source`

- When used on a CMIS relationship resource, this link relation MUST point to an Atom entry document for the CMIS Resource specified by the `cmis:sourceId` property on the relationship.
- Source Link on Relationship
- Service: `getObject`
- Media Type: `application/atom+xml;type=entry`
- Resources: CMIS relationships

`http://docs.oasis-open.org/ns/cmis/link/200908/target`

- When used on a CMIS relationship resource, this link relation MUST point to an Atom entry document for the CMIS Resource specified by the `cmis:targetId` property on the relationship.
- Target Link on Relationship
- Service: `getObject`
- Media Type: `application/atom+xml;type=entry`
- Resources: CMIS relationships

`http://docs.oasis-open.org/ns/cmis/link/200908/policies`

- This link relation MUST point to a resource containing an Atom feed of CMIS policy resources for the CMIS resource containing this link relation.
- Service: `getAppliedPolicies`
- Media Type: `application/atom+xml;type=feed`
- Resources: CMIS documents, folders, relationships, policies, and items

`http://docs.oasis-open.org/ns/cmis/link/200908/acl`

- This link relation MUST point to a resource containing a CMIS ACL document for the CMIS resource containing this link relation.
- Service: `getACL`
- Media Type: `application/cmisacl+xml`
- Resources: CMIS documents, folders, relationships, policies, and items that are securable

http://docs.oasis-open.org/ns/cmis/link/200908/changes

- This link relation MUST point to an Atom feed containing the set of changes.
- Service: `getContentChanges`
- Media Type: `application/atom+xml;type=feed`
- Resources: AtomPub Workspace Element in Service document

http://docs.oasis-open.org/ns/cmis/link/200908/foldertree

- Used in AtomPub Service document to identify the folder tree for a specified folder.
- Service: `getFolderTree`
- Media Type: `application/atom+xml;type=feed`
- Resources: CMIS folders, also used in AtomPub Service document for root folder

http://docs.oasis-open.org/ns/cmis/link/200908/typedescendants

- Used in AtomPub Service document to identify the base types descendants.
- Service: `getTypeDescendants`
- Media Type: `application/atom+xml;type=feed`
- Resources: AtomPub Workspace Element in Service document

http://docs.oasis-open.org/ns/cmis/link/200908/rootdescendants

- Used in AtomPub Service document to identify the root folder descendants.
- Service: `getDescendants` for root folder
- Media Type: `application/atom+xml;type=feed`
- Resources: AtomPub Workspace Element in Service document

3.5 Atom Resources

For all Atom resources used in this specification, the following MUST be followed:

3.5.1 Feeds

Any feed MUST be a valid Atom Feed document and conform to the guidelines below for CMIS objects:

- `atom:updated` SHOULD be the latest time the folder or its contents was updated. If unknown by the underlying repository, it MUST be the current time.
- `atom:author/atom:name` MUST be the CMIS property `cmis:createdBy`.
- `atom:title` MUST be the CMIS property `cmis:name`.
- The `atom:link` with relation `self` MUST be generated to return the URI of the feed. If paging or any other mechanism is used to filter, sort, or change the representation of the feed, the URI MUST point back a resource with the same representation.
- A feed SHOULD contain the element `app:collection`, describing the appropriate media types supported for creation of new entries in the feed
- `atom:id` SHOULD be derived from `cmis:objectId`. This id MUST be compliant with atom's specification and be a valid URI.

- Feeds MAY be paged via the link relations specified in AtomPub. If more items are available than contained in the feed, then a link with the relation next MUST be included in the feed.

Any feed MUST be a valid Atom Feed document and conform to the guidelines below for CMIS types:

- `atom:updated` SHOULD be the latest time type definition was updated. If unknown by the underlying repository, it MUST be the current time.
- `atom:author/atom:name` is repository specific.
- `atom:title` MUST be the displayName attribute of the CMIS type definition.
- The `atom:link` with relation self MUST be generated to return the IRI of the feed.
- `atom:id` SHOULD be derived from the id attribute of the CMIS type definition. This id MUST be compliant with atom's specification and be a valid URI.
- Feeds MAY be paged via the link relations specified in AtomPub. If more items are available than contained in the feed, then a link with the relation next MUST be included in the feed.

If on the root type, all fields are repository specific.

Ordering of entries in a feed is repository-specific if the `orderBy` argument is not specified. If the `orderBy` argument is specified, the order of the entries in the feed SHOULD conform to the ordering specified by the `orderBy` argument. If a repository only supports a certain number of `orderBy` properties, it SHOULD ignore all additional properties.

3.5.2 Entries

At any point where an Atom document of type Entry is sent or returned, it must be a valid Atom Entry document and conform to the guidelines below for a cmis object:

- `atom:title` MUST be the `cmis:name` property.
- `app:edited` MUST be `cmis:lastModificationDate`.
- `atom:updated` MUST be `cmis:lastModificationDate`.
- `atom:published` MUST be `cmis:creationDate`.
- `atom:author/atom:name` MUST be `cmis:createdBy`.
- `atom:summary` SHOULD be `cmis:description`.
- All CMIS properties MUST be exposed in CMIS `cmis:properties` elements even if they are duplicated in an Atom element.
- `atom:id` SHOULD be derived from `cmis:objectId`. This id MUST be compliant with atom's specification and be a valid IRI.

For documents that support content streams:

The repository SHOULD use the `atom:content/src` attribute to point to the content stream. The client SHOULD use `cmisra:content` if the content is not well-formed or would have trouble fitting inside an `atom:content` element. The repository MUST use the `cmisra:content` element if provided by the client over the `atom:content` element.

Other objects:

(Folders, relationships, policies, items, and other document types that do not support content streams, etc.)
The repository SHOULD provide an `atom:summary` element and no `atom:content` element in order to comply with the Atom specification. Any value in the content field MUST be ignored if the Atom entry represents a non-document object by the CMIS repository when the Atom entry is POST'ed to a collection or sent to the repository via a PUT.

When POSTing an Atom Document, the Atom elements MUST take precedence over the corresponding writable CMIS property. For example, `atom:title` will overwrite `cmis:name`.

At any point where an Atom document of CMIS type is sent or returned, it must be a valid Atom Entry document and conform to the guidelines below for a CMIS type definition:

- `atom:title` MUST be the `cmis:displayName`
- The repository SHOULD populate the `atom:summary` tag with text that best represents a summary of the object. For example, the type description if available.

Any Atom element that is not specified is repository-specific.

3.5.2.1 Hierarchical Atom Entries

The repository SHOULD NOT provide any links to hierarchical objects if those capabilities are not supported with the exception of `getTypeDescendants` which is required.

For Atom entries that are hierarchical such as Folder Trees or Descendants, the repository MUST populate a `cmisra:children` element in the `atom:entry` with the enclosing feed of its direct children. This pattern continues until the depth is satisfied.

The `cmisra:children` element MUST include an `atom:feed` element that contains the children entries of this resource.

If an entry does not contain `cmisra:children` element, then the entry MAY have children even though it is not represented in the Atom entry.

Please see section 3.3.4 CMIS Tree.

3.6 Resources Overview

	Service	Resource	HTTP Method
Repository	getRepositories	AtomPub Service Document	GET
	getRepositoryInfo	AtomPub Service Document	GET
	getTypeChildren	Type Children Collection	GET
	getTypeDescendants	Type Descendants Feed	GET
	getTypeDefinition	Type Entry	GET
	createType	Type Children Collection	POST
	updateType	Type Entry	PUT
	deleteType	Type Entry	DELETE
Navigation	getChildren	Folder Children Collection	GET
	getDescendants	Folder Descendants Feed	GET
	getFolderTree	Folder Tree Feed	GET
	getFolderParent	Folder Entry	GET
	getObjectParents	Object Parents Feed	GET
	getCheckedOutDocs	Checked Out Collection	GET
Object	createDocument	Folder Children Collection or Unfiled Collection	POST
	createDocumentFromSource	See 3.1.9 Services not Exposed	
	createFolder	Folder Children Collection	POST
	createRelationship	Relationships Collection	POST
	createPolicy	Folder Children Collection or Unfiled Collection	POST
	getAllowableActions	AllowableActions Resource	GET
	getObject	Document Entry or PWC Entry or Folder Entry or Relationship Entry or Policy Entry or Item Entry or objectbyid URI template	GET
	getProperties	See 3.1.9 Services not Exposed	
	getObjectByPath	objectbypath URI template	GET
	getContentStream	Content Stream	GET
	getRenditions	See 3.1.9 Services not Exposed	
	updateProperties	Document Entry or PWC Entry or Folder Entry or Relationship Entry or Policy Entry or Item Entry	PUT
	bulkUpdateProperties	Bulk Update Collection	POST
	moveObject	Folder Children Collection	POST

	Service	Resource	HTTP Method
	deleteObject	Document Entry or Folder Entry or Relationship Entry or Policy Entry or Item Entry	DELETE
	deleteTree	Folder Tree Feed	DELETE
	setContentStream	Content Stream	PUT
	appendContentStream	Content Stream	PUT
	deleteContentStream	Content Stream	DELETE
Multi	addObjectToFolder	Folder Children Collection	POST
	removeObjectFromFolder	Unfiled Collection	POST
Disc	query	Query Collection	POST
	getContentChanges	Changes Feed	GET
Versioning	checkOut	Checked Out Collection	POST
	cancelCheckOut	PWC Entry	DELETE
	checkIn	PWC Entry	PUT
	getObjectOfLatestVersion	Document Entry	PUT
	getPropertiesOfLatestVersion	See 3.1.9 Services not Exposed	
	getAllVersions	All Versions Feed	GET
Rel	getObjectRelationships	Relationships Collection	GET
Policy	applyPolicy	Policies Collection	POST
	removePolicy	Policies Collection	DELETE
	getAppliedPolicies	Policies Collection	GET
ACL	applyACL	ACL Resource	PUT
	getACL	ACL Resource	GET

3.7 AtomPub Service Document

The AtomPub Service Document contains the set of repositories that are available.

How the client will get the initial AtomPub (APP) service document or the URI for the service document is repository specific. Examples are via URI, or loading the service document from disk.

The service document will also be available from Atom Entry and Atom Feed documents via a link relationship `service`. That AtomPub service document MUST then only contain one workspace element which MUST be the workspace representing the repository containing the Atom Entry or Atom Feed document.

3.7.1 HTTP GET

CMIS Services:

- `getRepositories`
- `getRepositoryInfo`

Arguments:

repositoryId (optional)

- This query parameter allows a client to specify a different repository than the one that is referenced by the URI.
- If specified, the repository MUST return the AtomPub services document for the specified repository if that repository exists.
- If not specified, the repository MUST return the service document for the repository that is referenced by URI.

Media Type:

- `application/atomsvc+xml`

Each repository is mapped to a `app:workspace` element in the AtomPub Service document. A workspace element MUST have a collection element for each of following collections. Each collection MUST also contain a `cmisra:collectionType` element with the given value.

- Root Folder Children Collection: Root folder of the repository
  - 'root' for the children collection of the root folder
  - `cmisra:collectiontype` = 'root'
- Types Children Collection: Collection containing the base types in the repository
  - 'types' for the children collection
  - `cmisra:collectiontype` = 'types'

The workspace element SHOULD also contain these collections if the repository supports this functionality:

- CheckedOut collection: collection containing all checked out documents user can see
  - `cmisra:collectiontype` = 'checkedout'
- Query collection: Collection for posting queries to be executed
  - `cmisra:collectiontype` = 'query'
- Unfiled collection: Collection for posting documents to be unfiled; read can be disabled
  - `cmisra:collectiontype` = 'unfiled'
- Bulk update collection: Collection for posting property updates for multiple objects at once

- cmisra:collectiontype = 'update'

The workspace element MUST also contain the following link element with the relation:

http://docs.oasis-open.org/ns/cmis/link/200908/typedescendants

- This link relation points to the Type Descendants Feed for the base types in the repository.

The workspace element MUST contain the following link elements of the following relations for those services which are supported by the repository:

http://docs.oasis-open.org/ns/cmis/link/200908/foldertree

- This link relation points to the Folder Tree Feed for the root folder.

http://docs.oasis-open.org/ns/cmis/link/200908/rootdescendants

- This link relation points to the Folder Descendants Feed for the root folder.

http://docs.oasis-open.org/ns/cmis/link/200908/changes

- This link relation points to the Changes Feed for the repository.

The workspace element may include app:collection elements for the collections that represent folders in the repository. However, an alternative approach, especially for a repository with many folders, is to not enumerate those collections here, but include the app:collection element per [RFC5023] in the Atom Feed document.

The repository MUST include the URI templates in the workspace elements.

3.7.1.1 URI Templates

CMIS defines the following URI templates:

- objectbyid
- objectbypath
- query
- typebyid

Repositories MUST provide the following URI templates:

- objectbyid
- objectbypath
- typebyid

Repositories MUST provide the URI template query if the repository supports query.

Repositories MAY extend that set of templates. Those URI template types will be repository specific. Repositories MAY have more than one entry per URI template type if the entries have different media types.

URI templates are simple replacement of the template parameter with the specified value. If a client does not want to specify a value for some of these variables, then the client MUST substitute an empty string for the variable.

For example, if the URI template that supports the variable {id} is:

```
http://example.com/rep1/getbyid/{id}
```

If the client wants to find the entry for an object with an id of 'obj_1' then the URI would be:

```
http://example.com/rep1/getbyid/obj_1
```

URI Templates MUST only be used with HTTP GET.

Arguments that are substituted for URI template parameters MUST be percent escaped according to [RFC3986]. Please see that RFC for more information.

All variables MUST be in the template.

3.7.1.1.1 Structure of URI Templates

Structure

```
<xs:complexType name="cmisUriTemplateType">
  <xs:sequence>
    <xs:element name="template" type="xs:string" />
    <xs:element name="type" type="xs:string" />
    <xs:element name="mediatype" type="xs:string" />
    <xs:any processContents="lax" namespace="##other" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>
```

Example

```
<cmisra:uritemplate xmlns:cmis="http://docs.oasis-open.org/ns/cmis/core/200908/"
  xmlns:cmism="http://docs.oasis-open.org/ns/cmis/messaging/200908/"
  xmlns:atom="http://www.w3.org/2005/Atom"
  xmlns:app="http://www.w3.org/2007/app"
  xmlns:cmisra="http://docs.oasis-open.org/ns/cmis/restatom/200908/">
  <cmisra:template>http://example.com/rep1/objectbyid/{id}?filter={filter}&amp;
      ↪ includeAllowableActions={includeAllowableActions}&amp;
      ↪ includePolicyIds={includePolicyIds}&amp;includeRelationships={includeRelationships}&amp;
      ↪ includeACL={includeACL}</cmisra:template>
  <cmisra:type>objectbyid</cmisra:type>
  <cmisra:mediatype>application/atom+xml;type=entry</cmisra:mediatype>
</cmisra:uritemplate>
```

3.7.1.1.2 Object By Id

This URI template provides a method for creating an URI that directly accesses an Atom entry representing document, folder, policy, relationship, or item objects. See section 3.11 Resources for more information.

Type: `objectbyid`

Media Type: `application/atom+xml;type=entry`

Service: `getObject` or `getObjectOfLatestVersion`

Variables that are supported by the template:

{id} maps to service parameter `objectId`.

{filter} maps to service parameter `filter`.

{includeAllowableActions} maps to service parameter `includeAllowableActions`.

{includePolicyIds} maps to service parameter `includePolicyIds`.

{includeRelationships} maps to service parameter `includeRelationships`.

{includeACL} maps to service parameter `includeACL`.

{renditionFilter} maps to service parameter `renditionFilter`.

{returnVersion}

- If no value is present or the value is 'this', `getObject` MUST be called.

- If the value is 'latest' `getObjectOfLatestVersion` MUST be called with the parameter `major` set to FALSE.
- If the value is 'latestmajor' `getObjectOfLatestVersion` MUST be called with the parameter `major` set to TRUE.

Example

```
<cmisra:uritemplate xmlns:cmis="http://docs.oasis-open.org/ns/cmis/core/200908/"
  xmlns:cmism="http://docs.oasis-open.org/ns/cmis/messaging/200908/"
  xmlns:atom="http://www.w3.org/2005/Atom"
  xmlns:app="http://www.w3.org/2007/app"
  xmlns:cmisra="http://docs.oasis-open.org/ns/cmis/restatom/200908/">
  <cmisra:template>http://example.com/rep1/objectbyid/{id}?filter={filter}&amp;
      ↪ includeAllowableActions={includeAllowableActions}&amp;
      ↪ includePolicyIds={includePolicyIds}&amp;includeRelationships={includeRelationships}&amp;
      ↪ includeACL={includeACL}&amp; returnVersion={returnVersion}</cmisra:template>
  <cmisra:type>objectbyid</cmisra:type>
  <cmisra:mediatype>application/atom+xml;type=entry</cmisra:mediatype>
</cmisra:uritemplate>
```

3.7.1.1.3 Object By Path

This URI template provides a method for creating an URI that directly accesses an Atom entry representing document, folder, policy, relationship, or item objects. See section 3.11 Resources for more information.

Type: `objectbyid`

Media Type: `application/atom+xml;type=entry`

Service: `getObjectByPath`

Variables that are supported by the template:

{path} maps to service parameter `path`.

{filter} maps to service parameter `filter`.

{includeAllowableActions} maps to service parameter `includeAllowableActions`.

{includePolicyIds} maps to service parameter `includePolicyIds`.

{includeRelationships} maps to service parameter `includeRelationships`.

{includeACL} maps to service parameter `includeACL`.

{renditionFilter} maps to service parameter `renditionFilter`.

Example

```
<cmisra:uritemplate xmlns:cmis="http://docs.oasis-open.org/ns/cmis/core/200908/"
  xmlns:cmism="http://docs.oasis-open.org/ns/cmis/messaging/200908/"
  xmlns:atom="http://www.w3.org/2005/Atom"
  xmlns:app="http://www.w3.org/2007/app"
  xmlns:cmisra="http://docs.oasis-open.org/ns/cmis/restatom/200908/">
  <cmisra:template>http://example.com/rep1/objectbypath?p={path}&amp;filter={filter}&amp;
      ↪ includeAllowableActions={includeAllowableActions}&amp;
      ↪ includePolicyIds={includePolicyIds}&amp;includeRelationships={includeRelationships}&amp;
      ↪ includeACL={includeACL}</cmisra:template>
  <cmisra:type>objectbypath</cmisra:type>
  <cmisra:mediatype>application/atom+xml;type=entry</cmisra:mediatype>
</cmisra:uritemplate>
```

3.7.1.1.4 Query

This URI template provides a method for creating an URI that performs a query.

Type: `query`

Media Type: `application/atom+xml;type=feed`

Service: `query`

Variables that are supported by the template:

{q} maps to service parameter `statement`.

{searchAllVersions} maps to service parameter `searchAllVersions`.

{maxItems} maps to service parameter `maxItems`.

{skipCount} maps to service parameter `skipCount`.

{includeAllowableActions} maps to service parameter `includeAllowableActions`.

{includeRelationships} maps to service parameter `includeRelationships`.

{renditionFilter} maps to service parameter `renditionFilter`.

Example

```
<cmisra:uritemplate xmlns:cmis="http://docs.oasis-open.org/ns/cmis/core/200908/"
  xmlns:cmism="http://docs.oasis-open.org/ns/cmis/messaging/200908/"
  xmlns:atom="http://www.w3.org/2005/Atom"
  xmlns:app="http://www.w3.org/2007/app"
  xmlns:cmisra="http://docs.oasis-open.org/ns/cmis/restatom/200908/">
  <cmisra:template>http://example.com/rep1/query?q={q}&amp;
      ↪ searchAllVersions={searchAllVersions}&amp;maxItems={maxItems}&amp;
      ↪ skipCount={skipCount}&amp;includeAllowableActions={includeAllowableActions}=&amp;
      ↪ includeRelationships={includeRelationships}&amp;renditionFilter={renditionFilter}
      ↪ </cmisra:template>
  <cmisra:type>query</cmisra:type>
  <cmisra:mediatype>application/atom+xml;type=feed</cmisra:mediatype>
</cmisra:uritemplate>
```

3.7.1.1.5 Type By Id

This URI template provides a method for creating an URI that directly accesses a type definition.

Type: `typebyid`

Media Type: `application/atom+xml;type=entry`

Service: `getTypeDefinition`

Variables that are supported by the template:

{id} maps to service parameter `typeId`.

Example

```
<cmisra:uritemplate xmlns:cmis="http://docs.oasis-open.org/ns/cmis/core/200908/"
  xmlns:cmism="http://docs.oasis-open.org/ns/cmis/messaging/200908/"
  xmlns:atom="http://www.w3.org/2005/Atom"
  xmlns:app="http://www.w3.org/2007/app"
  xmlns:cmisra="http://docs.oasis-open.org/ns/cmis/restatom/200908/">
  <cmisra:template>http://example.com/rep1/type?id={id}</cmisra:template>
  <cmisra:type>query</cmisra:type>
```

```
  <cmisra:mediatype>application/atom+xml;type=entry</cmisra:mediatype>
</cmisra:uritemplate>
```

3.8 Service Collections

These are the collections that are included on an AtomPub Service document in the workspace element.

For any HTTP verb not specified on a resource, each implementation MAY chose to implement that HTTP verb in a repository-specific manner.

3.8.1 Root Folder Collection

This collection provides access to the children of the root folder. Please see section 3.9.2 Folder Children Collection.

3.8.2 Query Collection

This is a collection for processing queries. If the implementation supports GET on this collection, then the implementation SHOULD at least return a feed consisting of zero or more Atom entries. These Atom entries should represent persisted objects related to query such as persisted queries, long running queries or search templates.

3.8.2.1 HTTP POST

CMIS Services:

- `query`

Accept:

- MUST support CMIS query document (`application/cmisquery+xml`)
- MAY support other media type

Media Type:

- `application/atom+xml;type=feed`

The feed returned MUST contain a set of Atom entries representing the result set from the query.

The Atom entries should contain the bare minimum necessary for Atom compliance [RFC4287]. The Atom entries MUST contain the CMIS extension element (`cmisra:object`) containing the properties specified by the query in the select clause of the query statement.

If all the selected properties can be mapped to the same type reference, then the repository MAY include additional information in the Atom entry.

Please see [RFC5023] Section 5.3.

Link Relations:

service Points to the service document containing the CMIS repository. The service document MUST contain only one workspace element.
Media Type: `application/atomsvc+xml`

first, next, previous, last Paging link relations as appropriate.

The following CMIS Atom extension element MAY be included inside the Atom feed:

- `cmisra:numItems`

The following CMIS Atom extension element MUST be included inside the Atom entries:

- `cmisra:object` inside `atom:entry`

Success Status Codes:

- 201 Created

Headers returned:

- `Location`
- `Content-Location`

Upon submission (creation) of a query document, a response must be returned with a Location header representing the feed for that query. If the query cannot be performed and an Atom feed returned, the repository MUST return the appropriate HTTP status code. In addition, the server SHOULD return the feed directly. If the server does so, the server SHOULD also return the Content-Location header.

Example:
Request: `atompub/doQuery-request.log`
Response: `atompub/doQuery-response.log`

3.8.3 Checked Out Collection

3.8.3.1 HTTP GET

This is a collection described in the service document that contains the Private Working Copies (PWCs) of the checked-out documents in the repository.

CMIS Services:

- `getCheckedOutDocs`

Arguments:

- filter
- folderId
- maxItems
- skipCount
- renditionFilter
- includeAllowableActions
- includeRelationships

Media Type:

- `application/atom+xml;type=feed`

Link Relations:

service Points to the service document containing the CMIS repository. The service document MUST contain only one workspace element.
Media Type: `application/atomsvc+xml`

first, next, previous, last Paging link relations as appropriate.

The following CMIS Atom extension element MAY be included inside the Atom feed:

- `cmisra:numItems`

The following CMIS Atom extension element MUST be included inside the Atom entries:

- `cmisra:object` inside `atom:entry`

Success Status Codes:

- 200 OK

3.8.3.2 HTTP POST

When an Atom Entry is POSTed to this collection, the document will be checked out. A `Content-Location` header MUST be returned containing the location of the Private Working Copy. The newly created PWC Entry MUST be returned.

CMIS Services:

- `checkOut`

Accept:

- MUST support Atom Entry documents with CMIS extensions `application/atom+xml;type=entry` or `application/cmisatom+xml`
- MAY support other media type

Media Type:

- `application/atom+xml;type=entry`

Success Status Codes:

- 201 Created

Headers returned:

- `Content-Location`

Example:
Request: `atompub/checkOut-request.log`
Response: `atompub/checkOut-response.log`

3.8.4 Unfiled Collection

This is a collection described in the service document to manage unfiled document, policy, and item objects.

3.8.4.1 HTTP POST

If this is called with an existing object, the object will be removed from all folders in the repository by default. If the optional argument `removeFrom` is specified, the object will only be removed from that folder.

If this is called with an entry that doesn't have an object id, a new, unfiled object will be created.

The removed or newly created Document Entry, Policy Entry, or Item Entry MUST be returned.

CMIS Services:

- removeObjectFromFolder
- createDocument
- createPolicy
- createItem

If the Atom Entry POSTed has a valid cmis:objectId property, removeObjectFromFolder will be performed. If the Atom Entry POSTed has no cmis:objectId property, the value of the cmis:objectTypeId property decides if createDocument, createPolicy, or createItem will be performed. In all other cases (invalid object id, the object does not exist, the object is not in that folder, the object type id is invalid, the base type is neither cmis:document nor cmis:policy nor cmis:item, etc.) the appropriate HTTP status code MUST be returned. See also 3.9.2 Folder Children Collection.

Arguments:

- removeFrom
- versioningState

Accept:

- MUST support Atom Entry documents with CMIS extensions application/atom+xml;type=entry or application/cmisatom+xml
- MAY support other media type

Media Type:

- application/atom+xml;type=entry

Success Status Codes:

- 201 Created

Headers returned:

- Location

Example:
Request: atompub/removeObjectFromFolder-request.log
Response: atompub/removeObjectFromFolder-response.log

3.8.5 Type Children Collection

This is a collection described in the service document that contains the types in the repository under the specified parent type. If no parent type is specified, then the base types are returned in the feed. This feed does not include any nesting and is a flat feed.

3.8.5.1 HTTP GET

This feed contains a set of Atom entries for each child type definition.

CMIS Services:

- getTypeChildren

Arguments:

- typeId
- includePropertyDefinitions
- maxItems
- skipCount

Media Type:

- application/atom+xml;type=feed

Link Relations:

service Points to the service document containing the CMIS repository. The service document MUST contain only one workspace element.
Media Type: application/atomsvc+xml

via Points to the type definition entry whose children represent this feed.

down Points to the Atom feed document representing the descendants collection for this same type.
Media Type: application/cmistree+xml

up Points to the parent type definition. If this is a children feed for a base object type, this link is not present.

first, next, previous, last Paging link relations as appropriate.

The following CMIS Atom extension element MAY be included inside the Atom feed:

- cmisra:numItems

The following CMIS Atom extension element MUST be included inside the Atom entries:

- cmisra:object inside atom:entry

Success Status Codes:

- 200 OK

Example:
Request: atompub/getTypeChildren-request.log
Response: atompub/getTypeChildren-response.log

3.8.5.2 HTTP POST

This creates a new object-type.

The server MUST return the appropriate HTTP status code if the specified parent type doesn't match this collection.

The created object-type entry MUST be returned.

CMIS Services:

- createType

Accept:

- MUST support Atom Entry documents with CMIS type extensions `application/atom+xml;type=entry` or `application/cmisatom+xml`

Media Type:

- `application/atom+xml;type=entry`

Success Status Codes:

- 201 Created

Headers returned:

- `Location`

3.8.6 Bulk Update Collection

This collection is used for bulk updates.

3.8.6.1 HTTP POST

The POSTed entry MUST include a CMIS Atom extension element `cmisra:bulkUpdate`. It contains the set of objects that should be updated, the new property values and secondary type modifications.

CMIS Services:

- `bulkUpdateProperties`

Accept:

- MUST support Atom Entry documents with CMIS extensions `application/atom+xml;type=entry`
- MAY support other media type

Media Type:

- `application/atom+xml;type=feed`

Link Relations:

service Points to the service document containing the CMIS repository. The service document MUST contain only one workspace element.
Media Type: `application/atomsvc+xml`

The following CMIS Atom extension element MAY be included inside the Atom feed:

- `cmisra:numItems`

The following CMIS Atom extension element MUST be included inside the Atom entries:

- `cmisra:object` inside `atom:entry`

The returned object entries MUST follow these rules:

- The property `cmis:objectId` MUST be set. The value MUST be the original object id even if the repository created a new version and therefore generated a new object id. New object ids are not exposed by this binding.
- The property `cmis:changeToken` MUST be set if the repository supports change tokens.
- All other properties are optional.

Success Status Codes:

- 201 Created

3.9 Collections

For any HTTP verb not specified on a resource, each implementation MAY chose to implement that HTTP verb in a repository-specific manner.

3.9.1 Relationships Collection

This collection manages relationships.

3.9.1.1 HTTP GET

This collection contains the set of relationships available (either source or target or both) from a specific item such as a document, folder, policy, or item.

CMIS Services:

- `getObjectRelationships`

Arguments:

- typeId
- includeSubRelationshipTypes
- relationshipDirection
- maxItems
- skipCount
- filter
- includeAllowableActions

Media Type:

- `application/atom+xml;type=feed`

Link Relations:

service Points to the service document containing the CMIS repository. The service document MUST contain only one workspace element.
Media Type: `application/atomsvc+xml`

first, next, previous, last Paging link relations as appropriate.

The following CMIS Atom extension element MAY be included inside the Atom feed:

- `cmisra:numItems`

The following CMIS Atom extension element MUST be included inside the Atom entries:

- `cmisra:object` inside `atom:entry`

Success Status Codes:

- 200 OK

3.9.1.2 HTTP POST

When an Atom entry with CMIS markup is POSTed to this collection, if that Atom entry represents a new CMIS relationship, then that relationship will be created.

The server MUST return the appropriate HTTP status code if the source is different than the sourceId or target different than the targetId for the source and targets specified in this collection.

The server MUST return the appropriate status code if the `cmis:objectTypeId` is not specified.

CMIS Services:

- `createRelationship`

Accept:

- MUST support Atom Entry documents with CMIS extensions `application/atom+xml;type=entry` or `application/cmisatom+xml`
- MAY support other media type

Media Type:

- `application/atom+xml;type=entry`

Success Status Codes:

- 201 Created

Headers returned:

- `Location`

3.9.2 Folder Children Collection

This collection managed folder children.

3.9.2.1 HTTP GET

CMIS Services:

- `getChildren`

Arguments:

- maxItems
- skipCount
- filter
- includeAllowableActions
- includeRelationships
- renditionFilter
  If specified, renditions will be returned as links with relation `alternate`.
- orderBy
- includePathSegment

Media Type:

- `application/atom+xml;type=feed`

Link Relations:

service Points to the service document containing the CMIS repository. The service document MUST contain only one workspace element.
Media Type: `application/atomsvc+xml`

via Points to the Atom entry of the folder generating this collection.

up Points to the Atom entry document for this folder's parent. If the root folder, this link relation MUST NOT be included.
Media Type: `application/atom+xml;type=entry`

down Points to the Atom feed document representing the descendants feed. This is represented as a feed with CMIS hierarchy extensions. If a repository does not support `capabilityGetDescendants`, then this link SHOULD NOT be included.
Media Type: `application/cmistree+xml`

http://docs.oasis-open.org/ns/cmis/link/200908/foldertree Points to the folder tree for this folder. This is represented as a feed with CMIS hierarchy extensions. If a repository does not support `capabilityGetFolderTree`, then this link SHOULD NOT be included.
Media Type: `application/atom+xml;type=feed`

first, next, previous, last Paging link relations as appropriate.

The following CMIS Atom extension element MAY be included inside the Atom feed:

- `cmisra:numItems`

The following CMIS Atom extension element MUST be included inside the Atom entries:

- `cmisra:object` inside `atom:entry`
- `cmisra:pathSegment` inside `atom:entry` if includePathSegment is TRUE

Success Status Codes:

- 200 OK

Example:
Request: `atompub/getChildren-request.log`
Response: `atompub/getChildren-response.log`

3.9.2.2 HTTP POST

CMIS Services:

- `createDocument`
- `createFolder`
- `createPolicy`
- `createItem`
- `moveObject`
- `addObjectToFolder`

POSTing an Atom Entry document with CMIS markup:

If the Atom entry has a CMIS property `cmis:objectId` that is valid for the repository, the object (document, folder, policy, or item) will be added to the folder.
When an object is added to the folder, in repositories that do not support multi-filing it will be removed from the previous folder and the operation treated as move. If the repository supports multiple folders, it will be added to the new folder. If the optional argument sourceFolderId is specified, then the object will be removed from the folder specified.

Creating a new CMIS object in that folder:

If the `cmis:objectId` property is missing, the object will be created and then added to the folder. If the `cmis:objectId` property is present but not a valid object Id, the repository MUST return the appropriate HTTP status code.

For documents:

- If Content Stream is not provided and it is required by the type definition, the repository MUST return the appropriate HTTP status code.
- Content Streams MAY be provided by any of the following mechanisms:
  - As part of the Atom entry via the `src` attribute on the content element. Implementers MAY support external references to content. If the URI in the `src` attribute is not reachable, then an appropriate HTTP status code should be returned.
  - As part of the Atom entry inlining via the AtomPub `content` element. Please see the AtomPub specification [RFC5023] for the processing model of the content element.
  - If the `cmisra:content` is provided by the client inside the `atom:entry`, the `cmisra:content` element MUST take precedence over the `atom:content` element. This element `cmisra:content` contains the content base64 encoded.
  - At a later time by replacing the `edit-media` link with a new content.
- The optional argument versioningState MAY specify additional versioning behavior such as checkin.

POSTing other document formats (AtomPub):

The behavior is repository specific when a non Atom entry or an atom document without the CMIS elements is posted to a folder collection.
For example, the repository MAY auto-create a document with a specific type (document) the client could edit.
If the repository does not support this scenario or another exception occurs, then the repository MUST return the appropriate HTTP status code.

Arguments:

sourceFolderId This parameter indicates the folder from which the object shall be moved from to the current specified folder. This parameter is not allowed for create operations.
If specified `moveObject` will be performed. If not specified, `addObjectToFolder` will be performed.

versioningState This optional argument MAY specify additional versioning behavior such as checkIn as major or minor. Please see CMIS Domain Model for more information on this parameter.

Accept:

- MUST support Atom Entry documents with CMIS extensions `application/atom+xml;type=entry` or `application/cmisatom+xml`
- MAY support other media type

Media Type:

- `application/atom+xml;type=entry`

Success Status Codes:

- 201 Created

Headers returned:

- `Location`

Example:
Request: `atompub/createDocument-request.log`
Response: `atompub/createDocument-response.log`

3.9.2.3 HTTP DELETE

See HTTP DELETE description in section 3.10.4 Folder Tree Feed.

3.9.3 Policies Collection

This collection managed policies applied to an object.

3.9.3.1 HTTP GET

The policy entries displayed here are specific to the object generating this collection.

CMIS Services:

- `getAppliedPolicies`

Arguments:

- filter

Media Type:

- `application/atom+xml;type=feed`

Link Relations:

service Points to the service document containing the CMIS repository. The service document MUST contain only one workspace element.
Media Type: `application/atomsvc+xml`

via Points to the Atom entry of the resource generating this collection.

first, next, previous, last Paging link relations as appropriate.

The following CMIS Atom extension element MAY be included inside the Atom feed:

- `cmisra:numItems`

The following CMIS Atom extension element MUST be included inside the Atom entries:

- `cmisra:object` inside `atom:entry`

Success Status Codes:

- 200 OK

3.9.3.2 HTTP POST

When an Atom Entry representing a Policy is posted to this collection, the policy will be applied to the object.

CMIS Services:

- `applyPolicy` (to object representing this collection of policies)

Accept:

- MUST support Atom Entry documents with CMIS extensions `application/atom+xml;type=entry` or `application/cmisatom+xml`
- MAY support other media type

Media Type:

- `application/atom+xml;type=entry`

Success Status Codes:

- 201 Created

Headers returned:

- `Location`

3.9.3.3 HTTP DELETE

This is the only collection where the URI's of the objects in the collection MUST be specific to that collection. A DELETE on the policy object in the collection is a removal of the policy from the object NOT a deletion of the policy object itself.

CMIS Services:

- `removePolicy`

Success Status Codes:

- 204 No Content

3.10 Feeds

For any HTTP verb not specified on a resource, each implementation MAY chose to implement that HTTP verb in a repository-specific manner.

3.10.1 Object Parents Feed

This is the set of parents for a specific object.

3.10.1.1 HTTP GET

CMIS Services:

- `getObjectParents`

Arguments:

- filter
- includeAllowableActions
- includeRelationships
- renditionFilter
- includeRelativePathSegment

Media Type:

- `application/atom+xml;type=feed`

Link Relations:

service Points to the service document containing the CMIS repository. The service document MUST contain only one workspace element.
Media Type: `application/atomsvc+xml`

first, next, previous, last Paging link relations as appropriate.

The following CMIS Atom extension element MUST be included inside the Atom entries:

- `cmisra:object` inside `atom:entry`
- `cmisra:relativePathSegment` inside `atom:entry` if includeRelativePathSegment is TRUE

Success Status Codes:

- 200 OK

Example:
Request: `atompub/getObjectParents-request.log`
Response: `atompub/getObjectParents-response.log`

3.10.2 Changes Feed

This is a link relationship described in the service document that contains the changes in the repository in the workspace element. The link relation pointing to this feed is `http://docs.oasis-open.org/ns/cmis/link/200908/changes`.

The ChangeLog Token is specified in the URI specified by the paging link notations. Through this binding it is not possible to retrieve the ChangeLog Token from the URIs.

3.10.2.1 HTTP GET

This feed MUST be ordered from oldest first to newest.

If the next changes does not exist yet, the link relation next MAY be available. If the next link relation is not available, the client should revisit the feed in the future and look for new items and the next link relation.

CMIS Services:

- `getContentChanges`

Arguments:

- filter
- maxItems
- includeACL
- includePolicyIds
- includeProperties
- changeLogToken
  If this parameter is specified, start the changes from the specified token. The changeLogToken is embedded in the paging link relations for normal iteration through the change list.

Media Type:

- `application/atom+xml;type=feed`

Link Relations:

service Points to the service document containing the CMIS repository. The service document MUST contain only one workspace element.
Media Type: `application/atomsvc+xml`

first, next, previous, last Paging link relations as appropriate. ChangeLogToken is incorporated into the URI specified by the next link relation.

The following CMIS Atom extension element MAY be included inside the Atom feed:

- `cmisra:numItems`

The following CMIS Atom extension element MUST be included inside the Atom entries:

- `cmisra:object` inside `atom:entry`

Success Status Codes:

- 200 OK

Example:
Request: `atompub/getContentChanges-request.log`
Response: `atompub/getContentChanges-response.log`

3.10.3 Folder Descendants Feed

This is a hierarchical feed comprising items under a specified folder to a specified depth. This is available via the link relation down with the `application/cmistree+xml` media type. Please see section 3.5.2.1 Hierarchical Atom Entries for more information on format.

If a repository does not support `capabilityGetDescendants`, then these resources SHOULD NOT be exposed.

3.10.3.1 HTTP GET

CMIS Services:

- `getDescendants`

Arguments:

- filter
- depth
- includeAllowableActions
- includeRelationships
- renditionFilter
- includePathSegment

Media Type:

- `application/atom+xml;type=feed`

Link Relations:

service Points to the service document containing the CMIS repository. The service document MUST contain only one workspace element.
Media Type: `application/atomsvc+xml`

via Points to the Atom entry of the folder generating this collection.

up Points to the Atom entry document for this folder's parent. If the root folder, this link relation MUST NOT be included.
Media Type: `application/atom+xml;type=entry`

down Points to the Atom feed document representing the children feed for this same folder.
Media Type: `application/atom+xml;type=feed`
Since this is the descendants, the descendants link SHOULD NOT be included.

http://docs.oasis-open.org/ns/cmis/link/200908/foldertree Points to the folder tree for this folder. If a repository does not support `capabilityGetFolderTree`, then this link SHOULD NOT be included.

The following CMIS Atom extension element MAY be included inside the Atom feed:

- `cmisra:numItems`

The following CMIS Atom extension element MUST be included inside the Atom entries:

- `cmisra:object` inside `atom:entry`
- `cmisra:pathSegment` inside `atom:entry` if includePathSegment is TRUE
- `cmisra:children` inside `atom:entry`

Success Status Codes:

- 200 OK

Example:
Request: `atompub/getDescendants-request.log`
Response: `atompub/getDescendants-response.log`

3.10.3.2 HTTP DELETE

See HTTP DELETE description in section 3.10.4 Folder Tree Feed.

3.10.4 Folder Tree Feed

This is a hierarchical feed comprising all the folders under a specified folder. This is available via the link relation foldertree with media type `application/atom+xml;type=feed`. Please see section 3.5.2.1 Hierarchical Atom Entries for more information on format.

3.10.4.1 HTTP GET

CMIS Services:

- `getFolderTree`

Arguments:

- filter
- depth
- includeAllowableActions
- includeRelationships
- renditionFilter
- includePathSegment

Media Type:

- `application/atom+xml;type=feed`

Link Relations:

service Points to the service document containing the CMIS repository. The service document MUST contain only one workspace element.
Media Type: `application/atomsvc+xml`

via Points to the Atom entry of the folder generating this collection.

up Points to the Atom entry document for this folder's parent. If the root folder, this link relation MUST not be included.
Media Type: `application/atom+xml;type=entry`

down Points to the Atom feed document representing the children feed for this same folder.
Media Type: `application/atom+xml;type=feed`

down Points to the descendants feed of the same folder. If a repository does not support `capabilityGetDescendant` then this link SHOULD NOT be included.
Media Type: `application/cmistree+xml`

The following CMIS Atom extension element MAY be included inside the Atom feed:

- `cmisra:numItems`

The following CMIS Atom extension element MUST be included inside the Atom entries:

- `cmisra:object` inside `atom:entry`
- `cmisra:pathSegment` inside `atom:entry` if includePathSegment is TRUE
- `cmisra:children` inside `atom:entry`

Success Status Codes:

- 200 OK

3.10.4.2 HTTP DELETE

This deletes the folder and all sub-folders.

If the DELETE method does not delete all items, invoking GET with infinite depth on the Folder Descendants Feed will return the items not deleted. Subsequent DELETE methods can be invoked on this URI.

Note: If the repository does not implement the Folder Descendants Feed, there is no mechanism to identify the resources that were not removed.

CMIS Services:

- `deleteTree`

Arguments:

- continueOnFailure
- unfileObjects

Success Status Codes:

- 200 OK, if successful. Body contains entity describing the status
- 202 Accepted, if accepted but deletion not yet taking place
- 204 No Content, if successful with no content
- 401 Unauthorized, if not authenticated
- 403 Forbidden, if permission is denied
- 500 Internal Server Error. The body SHOULD contain an entity describing the status

3.10.5 All Versions Feed

This is a feed comprised of all the versions of the given document. The feed MUST contain the newest versions at the beginning of the feed.

3.10.5.1 HTTP GET

CMIS Services:

- `getAllVersions`

Arguments:

- filter
- includeAllowableActions

Media Type:

- `application/atom+xml;type=feed`

Link Relations:

service Points to the service document containing the CMIS repository. The service document MUST contain only one workspace element.
Media Type: `application/atomsvc+xml`

via Points to the Atom entry of the resource generating this collection.

first, next, previous, last Paging link relations as appropriate.

The following CMIS Atom extension element MUST be included inside the Atom entries:

- `cmisra:object` inside `atom:entry`

Success Status Codes:

- 200 OK

Example:
Request: `atompub/getAllVersions-request.log`
Response: `atompub/getAllVersions-response.log`

3.10.6 Type Descendants Feed

This is a feed described in the service document that contains all the types under a specific type in the repository to a specific depth. If no parent type is specified, then the base types and their descendants are returned in the feed which is equivalent to all types in the repository if depth is infinite. The link relation is `http://docs.oasis-open.org/ns/cmis/link/200908/typedescendants`.

Types are nested using the CMIS hierarchy extension. Please see section 3.5.2.1 Hierarchical Atom Entries for more information on format.

3.10.6.1 HTTP GET

CMIS Services:

- `getTypeDescendants`

Arguments:

- typeId
- depth
- includePropertyDefinitions

Media Type:

- `application/atom+xml;type=feed`

Link Relations:

service Points to the service document containing the CMIS repository. The service document MUST contain only one workspace element.
Media Type: `application/atomsvc+xml`

via Points to the type definition whose descendants represent this feed. This link is not present if no parent type is specified.

up Points to the parent type definition. If this is a descendants feed for a base object type, this link is not present.

down Points to the children feed for the same type.

The following CMIS Atom extension element MAY be included inside the Atom feed:

- `cmisra:numItems`

The following CMIS Atom extension element MUST be included inside the Atom entries:

- `cmisra:type` inside `atom:entry`

Success Status Codes:

- 200 OK

3.11 Resources

For any HTTP verb not specified on a resource,each implementation MAY chose to implement that HTTP verb in a repository-specific manner.

3.11.1 Type Entry

This represents a type definition in the repository. This is enclosed as an Atom entry.

3.11.1.1 HTTP GET

CMIS Services:

- `getTypeDefinition`

Media Type:

- `application/atom+xml;type=entry`

Link Relations:

service Points to the service document containing the CMIS repository. The service document MUST contain only one workspace element.
Media Type: `application/atomsvc+xml`

up Points to the parent type as Atom entry if applicable.

down Points to the children feed of this type as Atom feed if applicable.
Media Type: `application/atom+xml;type=feed`

down Points to the descendants feed of this type as Atom feed if applicable.
Media Type: `application/cmistree+xml`

describedby Points to the type definition Atom entry of the base type of this type definition.

The following CMIS Atom extension element MUST be included inside the Atom entry:

- `cmisra:type`

Success Status Codes:

- 200 OK

Example:
Request: `atompub/getTypeDefinition-request.log`
Response: `atompub/getTypeDefinition-response.log`

3.11.1.2 HTTP PUT

This updates the object-type.

The updated object-type entry MUST be returned. See section 2.1.10 Object-Type Creation, Modification and Deletion for details.

CMIS Services:

- `updateType`

Accept:

- MUST support Atom Entry documents with CMIS type extensions `application/atom+xml;type=entry` or `application/cmisatom+xml`

Media Type:

- `application/atom+xml;type=entry`

Success Status Codes:

- 200 OK

Headers returned:

- `Location`

3.11.1.3 HTTP DELETE

This deletes the object-type.

CMIS Services:

- `deleteType`

Success Status Codes:

- 204 No Content

3.11.2 Document Entry

This is a CMIS Document instance.

3.11.2.1 HTTP GET

This returns the document.

CMIS Services:

- `getObject`
- `getObjectOfLatestVersion`

Arguments:

- returnVersion
  Used to differentiate between `getObject` and `getObjectOfLatestVersion`. Valid values are are described by the schema element `cmisra:enumReturnVersion`. If not specified, return the version specified by the URI.

- includeAllowableActions
- includeRelationships
- includePolicyIds
- includeACL
- filter
- renditionFilter

Media Type:

- `application/atom+xml;type=entry`

Link Relations:

service Points to the service document containing the CMIS repository. The service document MUST contain only one workspace element.
Media Type: `application/atomsvc+xml`

self Points to an URI that returns the Atom entry for this document. Please see Atom for more information.

edit Points to an URI that accepts PUT of Atom entry. Please see AtomPub for more information.

up Points to the atom feed containing the set of parents. If there is only one parent, the repository MAY point this link relation directly to the Atom entry of the parent.

version-history Points to Atom feed containing the versions of this document. If the document is not versionable, this link relation may not be on the resource.

current-version Points to the latest version of the document. Uses query parameter 'returnVersion' and enumReturnVersion. If this version is the current-version, this link relation MAY not be on the resource.

edit-media Same as `setContentStream`. Allows updating the content stream on this document. Please see AtomPub for more information.

working-copy Points to the private working copy if it exists.

describedby Points to the type definition as an Atom entry for the type of this document.

alternate This is used to identify the renditions available for the specified object. Please see section 3.1.6 Renditions.

http://docs.oasis-open.org/ns/cmis/link/200908/allowableactions Points to the allowable actions document for this object.

http://docs.oasis-open.org/ns/cmis/link/200908/relationships Points to the relationships feed for this object.

http://docs.oasis-open.org/ns/cmis/link/200908/policies Points to the policies feed for this object.

http://docs.oasis-open.org/ns/cmis/link/200908/acl Points to the ACL document for this object.

The following CMIS Atom extension element MUST be included inside the Atom entry:

- `cmisra:object`

Success Status Codes:

- 200 OK

Example:
Request: `atompub/getObject-request.log`
Response: `atompub/getObject-response.log`

3.11.2.2 HTTP PUT

This does a replacement of the Atom entry with the Atom entry document specified. If readwrite properties are not included, the repository SHOULD NOT modify them. The updated entry SHOULD be returned.

CMIS Services:

- `updateProperties`

Accept:

- Atom Entry documents with CMIS extensions `application/atom+xml;type=entry` or `application/cmisatom+xml`

Media Type:

- `application/atom+xml;type=entry`

Success Status Codes:

- 200 OK

Headers returned:

- `Location`

3.11.2.3 HTTP DELETE

This removes the document or all versions of the version series.

CMIS Services:

- `deleteObject`

Arguments:

- allVersions

Success Status Codes:

- 204 No Content

Example:
Request: `atompub/deleteObject-request.log`
Response: `atompub/deleteObject-response.log`

3.11.3 PWC Entry

This is the private working copy of the document (checkedout version of document).

3.11.3.1 HTTP GET

CMIS Services:

- `getObject`

Arguments:

- includeAllowableActions
- includeRelationships
- includePolicyIds
- includeACL
- filter
- renditionFilter

Media Type:

- `application/atom+xml;type=entry`

Media Type:

- `application/atom+xml;type=entry`

Link Relations:

service Points to the service document containing the CMIS repository. The service document MUST contain only one workspace element.
Media Type: `application/atomsvc+xml`

self Points to the URI to retrieve this Atom entry. Please see Atom for more information.

edit Points to the URI to update this Atom entry via PUT. Please see AtomPub for more information.

up Points to the Atom feed containing the set of parents. If there is only one parent, the repository MAY point this link relation directly to the Atom entry of the parent.

version-history Points to Atom feed containing the versions of this document.

edit-media Same as `setContentStream`. Allows updating the content stream on this document. Please see AtomPub for more information.

via Atom entry that created this PWC.

describedby Points to the type definition as an Atom entry for the type of this PWC entry.

alternate This is used to identify the renditions available for the specified object. Please see section 3.1.6 Renditions.

http://docs.oasis-open.org/ns/cmis/link/200908/allowableactions Points to the allowable actions document for this object.

http://docs.oasis-open.org/ns/cmis/link/200908/relationships Points to the relationships feed for this object.

http://docs.oasis-open.org/ns/cmis/link/200908/policies Points to the policies feed for this object.

http://docs.oasis-open.org/ns/cmis/link/200908/acl Points to ACL document for this object.

The following CMIS Atom extension element MUST be included inside the Atom entry:

- `cmisra:object`

Success Status Codes:

- 200 OK

3.11.3.2 HTTP PUT

This does a replacement of the Atom entry with the Atom entry document specified. If modifiable properties (whencheckedout or readwrite) are not included, the repository SHOULD NOT modify them. The updated entry SHOULD be returned.

CMIS Services:

- updateProperties
- checkIn

Media Type:

- application/atom+xml;type=entry

Arguments:

- checkinComment
- major
- checkin
  Used to differentiate between updateProperties or checkIn services. If TRUE, execute checkIn service.

Success Status Codes:

- 200 OK

Headers returned:

- Location

Example:
Request: atompub/checkIn-request.log
Response: atompub/checkIn-response.log

3.11.3.3 HTTP DELETE

This removes the document entry, in this case, cancels the check out. The PWC will be removed.

CMIS Services:

- cancelCheckOut

Success Status Codes:

- 204 No Content

3.11.4 Folder Entry

This is a CMIS Folder instance.

3.11.4.1 HTTP GET

CMIS Services:

- getObject

Arguments:

- includeAllowableActions
- includeRelationships

- includePolicyIds
- includeACL
- filter
- renditionFilter

Media Type:

- `application/atom+xml;type=entry`

Link Relations:

service Points to the service document containing the CMIS repository. The service document MUST contain only one workspace element.
Media Type: `application/atomsvc+xml`

self Points to an URI that returns the Atom entry for this folder. Please see Atom for more information.

edit Points to an URI that accepts PUT of Atom entry. Please see AtomPub for more information.

down Points to the feed document representing the children feed for this same folder.
Media Type: `application/atom+xml;type=feed`

down Points to the descendants feed of the same folder.
Media Type: `application/cmistree+xml`

up Points Atom entry of the parent. If the root folder, this link will not be present.

describedby Points to the type definition as an Atom entry for the type of this folder.

alternate This is used to identify the renditions available for the specified object. Please see section 3.1.6 Renditions.

http://docs.oasis-open.org/ns/cmis/link/200908/allowableactions Points to the allowable actions document for this object.

http://docs.oasis-open.org/ns/cmis/link/200908/relationships Points to the relationships feed for this object.

http://docs.oasis-open.org/ns/cmis/link/200908/policies Points to the policies feed for this object.

http://docs.oasis-open.org/ns/cmis/link/200908/acl Points to ACL document for this object.

The following CMIS Atom extension element MUST be included inside the Atom entry:

- `cmisra:object`

Success Status Codes:

- 200 OK

3.11.4.2 HTTP PUT

This does a replacement of the Atom entry with the Atom entry document specified. If readwrite properties are not included, the repository SHOULD NOT modify them. The updated entry SHOULD be returned.

CMIS Services:

- `updateProperties`

Accept:

- Atom Entry documents with CMIS extensions
`application/atom+xml;type=entry` or `application/cmisatom+xml`

Media Type:

- `application/atom+xml;type=entry`

Success Status Codes:

- 200 OK

Headers returned:

- `Location`

Example:
Request: `atompub/updateProperties-request.log`
Response: `atompub/updateProperties-response.log`

3.11.4.3 HTTP DELETE

This removes the folder from the repository. This is deletion of the folder only and not any contained objects.

CMIS Services:

- `deleteObject`

Success Status Codes:

- 204 No Content

3.11.5 Relationship Entry

This is a CMIS relationship instance. These objects are exposed via 'relationships' link type.

3.11.5.1 HTTP GET

CMIS Services:

- `getObject`

Arguments:

- includeAllowableActions
- includeRelationships
- includePolicyIds
- includeACL
- filter

Media Type:

- `application/atom+xml;type=entry`

Link Relations:

service Points to the service document containing the CMIS repository. The service document MUST contain only one workspace element.
Media Type: `application/atomsvc+xml`

self Points to an URI that returns the Atom entry for this relationship. Please see Atom for more information.

edit Points to an URI that accepts PUT of Atom entry. Please see AtomPub for more information.

describedby Points to the type definition as an Atom entry for the type of this relationship.

http://docs.oasis-open.org/ns/cmis/link/200908/allowableactions Points to the allowable actions document for this object.

http://docs.oasis-open.org/ns/cmis/link/200908/policies Points to the policies feed for this object.

http://docs.oasis-open.org/ns/cmis/link/200908/acl Points to the ACL document for this object.

http://docs.oasis-open.org/ns/cmis/link/200908/source Points to the Atom entry of the source object.

http://docs.oasis-open.org/ns/cmis/link/200908/target Points to the Atom entry of the target object.

The following CMIS Atom extension element MUST be included inside the Atom entry:

- `cmisra:object`

Success Status Codes:

- 200 OK

3.11.5.2 HTTP PUT

This does a replacement of the Atom entry with the Atom entry document specified. If readwrite properties are not included, the repository SHOULD NOT modify them. The updated entry SHOULD be returned.

CMIS Services:

- `updateProperties`

Accept:

- Atom Entry documents with CMIS extensions `application/atom+xml;type=entry` or `application/cmisatom+xml`

Media Type:

- `application/atom+xml;type=entry`

Success Status Codes:

- 200 OK

Headers returned:

- `Location`

3.11.5.3 HTTP DELETE

This removes the relationship from the repository.

CMIS Services:

- `deleteObject`

Success Status Codes:

- 204 No Content

3.11.6 Policy Entry

This is a CMIS policy instance.

3.11.6.1 HTTP GET

CMIS Services:

- `getObject`

Arguments:

- includeAllowableActions
- includeRelationships
- includePolicyIds
- includeACL
- filter

Media Type:

- `application/atom+xml;type=entry`

Link Relations:

service Points to the service document containing the CMIS repository. The service document MUST contain only one workspace element.
Media Type: `application/atomsvc+xml`

self Points to an URI that returns the Atom entry for this policy. Please see Atom for more information.

edit Points to an URI that accepts PUT of Atom entry. Please see AtomPub for more information.

describedby Points to the type definition as an Atom entry for the type of this policy.

http://docs.oasis-open.org/ns/cmis/link/200908/allowableactions Points to the allowable actions document for this object.

http://docs.oasis-open.org/ns/cmis/link/200908/relationships Points to the relationships feed for this object.

http://docs.oasis-open.org/ns/cmis/link/200908/acl Points to the ACL document for this object.

The following CMIS Atom extension element MUST be included inside the Atom entry:

- `cmisra:object`

Success Status Codes:

- 200 OK

3.11.6.2 HTTP PUT

This does a replacement of the Atom entry with the Atom entry document specified. If readwrite properties are not included, the repository SHOULD NOT modify them. The updated entry SHOULD be returned.

CMIS Services:

- `updateProperties`

Accept:

- Atom Entry documents with CMIS extensions
  `application/atom+xml;type=entry` or `application/cmisatom+xml`

Media Type:

- `application/atom+xml;type=entry`

Success Status Codes:

- 200 OK

Headers returned:

- `Location`

3.11.6.3 HTTP DELETE

This removes the policy from the repository. If this policy entry was discovered through a policy collection on an object, then `removePolicy` is performed rather than `deleteObject` on the policy itself.

CMIS Services:

- `deleteObject`

Success Status Codes:

- 204 No Content

3.11.7 Item Entry

This is a CMIS item instance.

3.11.7.1 HTTP GET

CMIS Services:

- `getObject`

Arguments:

- includeAllowableActions
- includeRelationships
- includePolicyIds
- includeACL
- filter

Media Type:

- `application/atom+xml;type=entry`

Link Relations:

service Points to the service document containing the CMIS repository. The service document MUST contain only one workspace element.
Media Type: `application/atomsvc+xml`

self Points to an URI that returns the Atom entry for this item. Please see Atom for more information.

edit Points to an URI that accepts PUT of Atom entry. Please see AtomPub for more information.

describedby Points to the type definition as an Atom entry for the type of this item.

http://docs.oasis-open.org/ns/cmis/link/200908/allowableactions Points to the allowable actions document for this object.

http://docs.oasis-open.org/ns/cmis/link/200908/relationships Points to the relationships feed for this object.

http://docs.oasis-open.org/ns/cmis/link/200908/acl Points to the ACL document for this object.

The following CMIS Atom extension element MUST be included inside the Atom entry:

- `cmisra:object`

Success Status Codes:

- 200 OK

3.11.7.2 HTTP PUT

This does a replacement of the Atom entry with the Atom entry document specified. If readwrite properties are not included, the repository SHOULD NOT modify them. The updated entry SHOULD be returned.

CMIS Services:

- `updateProperties`

Accept:

- Atom Entry documents with CMIS extensions `application/atom+xml;type=entry` or `application/cmisatom+xml`

Media Type:

- `application/atom+xml;type=entry`

Success Status Codes:

- 200 OK

Headers returned:

- `Location`

3.11.7.3 HTTP DELETE

This removes the item from the repository.

CMIS Services:

- `deleteObject`

Success Status Codes:

- 204 No Content

3.11.8 Content Stream

This is the content stream portion of the document object.

3.11.8.1 HTTP GET

This returns the content stream.

It is RECOMMENDED that HTTP Range requests are supported on this resource. Please see [RFC2616] for more information on HTTP Range requests. It is RECOMMENDED that HTTP compression is also supported.

CMIS Services:

- `getContentStream`

Arguments:

- streamId

Media Type:

- MIME type of the resource

Success Status Codes:

- 200 OK
- 206 Partial Content

3.11.8.2 HTTP PUT

Sets or replaces the content stream or appends a chunk of content to the existing content stream.

If the client wishes to set a new filename, it MAY add a `Content-Disposition` header, which carries the new filename. The disposition type MUST be "attachment". The repository SHOULD use the "filename" parameter and SHOULD ignore all other parameters. (See [RFC2183] and [RFC2231].)

CMIS Services:

- `setContentStream`
- `appendContentStream`

Arguments:

- overwriteFlag
  If not specified, this defaults to TRUE in this binding and behaves consistent with AtomPub.
- changeToken
- append
  If specified and set to `true`, `appendContentStream` is called. Otherwise `setContentStream` is called.
- isLastChunk

Success Status Codes:

- 200 OK, if the resource is updated
- 204 No Content, if the resource is updated
- 201 Created, if a new resource is created

Headers returned:

- `Content-Location`
- `Location`

3.11.8.3 HTTP DELETE

This removes the content stream.

CMIS Services:

- `deleteObject`

Success Status Codes:

- 204 No Content

3.11.9 AllowableActions Resource

This is an AllowableActions document.

3.11.9.1 HTTP GET

This returns the CMIS AllowableActions for a specified object.

CMIS Services:

- `getAllowableActions`

Media Type:

- `application/cmisallowableactions+xml`

Success Status Codes:

- 200 OK

Example:
Request: `atompub/getAllowableActions-request.log`
Response: `atompub/getAllowableActions-response.log`

3.11.10 ACL Resource

This is an ACL document.

3.11.10.1 HTTP GET

This returns the CMIS ACL for a specified object.

CMIS Services:

- `getACL`

Arguments:

- onlyBasicPermissions

Media Type:

- `application/cmisacl+xml`

Success Status Codes:

- 200 OK

Example:
Request: `atompub/getAcl-request.log`
Response: `atompub/getAcl-response.log`

3.11.10.2 HTTP PUT

This applies the CMIS ACL for a specified object. The updated ACL SHOULD be returned.

CMIS Services:

- `applyACL`

Arguments:

- ACLPropagation

Accept:

- `application/cmisacl+xml`

Media Type:

- `application/cmisacl+xml`

Success Status Codes:

- 200 OK

4 Web Services Binding

4.1 Overview

All services and operations defined in the domain model specification are presented in the Web Services binding.

The WSDL for these services reference two XSD documents. One defines elements for the primary data types of documents, folders, relationships, policies, items, and secondary types as well as collections of these types of objects. The second XSD defines the message formats for each of the CMIS services; the messages often refer to the data types defined in the first XSD schema. The WSDL presents exactly the abstract services defined in the Services section.

The normative CMIS Web Services binding is defined by the WSDL and XSD as well as the details given here in this part of the CMIS specification.

4.1.1 WS-I

A CMIS Web Services binding MUST comply with WS-I Basic Profile 1.1 and Basic Security Profile 1.0.

4.1.2 Authentication

A CMIS Web Services binding SHOULD support WS-Security 1.1 for Username Token Profile 1.1 and MAY also support other authentication mechanisms. A CMIS repository MAY grant access to all or a subset of the CMIS services to unauthenticated clients.

4.1.3 Content Transfer

All endpoints of the Web Services binding MUST be MTOM enabled.

4.1.4 Reporting Errors

Services MUST report errors via SOAP faults. The `CMIS-Messaging.xsd` defines a basic fault structure that includes an error code and an error message and the WSDL for each service defines specific messages that have the basic fault format.

4.2 Web Services Binding Mapping

The Domain Model defines all services, operations, parameters and objects of CMIS. The Web Services binding is an exact one-to-one mapping of this definition with small exceptions that are explained in the next section. Operations and parameters are named exactly after their counterparts in the Services section. All rules and exceptions defined there apply to the Web Services binding. Optional parameters and optional return values are not set if they are missing or their value is NULL.

4.3 Additions to the Services section

4.3.1 updateProperties and checkIn Semantics

This binding supports partial properties updates. All properties passed to `updateProperties` or `checkIn` will be updated to their new values. Properties that are passed without a value will be set to their default value or un-set if no default value is defined. All others property values remain untouched.

4.3.2 Content Ranges

This binding supports the retrieval of content ranges. The operation `getContentStream` accepts two optional parameters:

Integer offset The first byte of the content to retrieve. Default value is 0.

Integer length The length of the range in bytes. Default value is the size of the content minus the offset.

If the offset value is greater than the size of the content the repository SHOULD throw a constraint exception. If offset + length is greater than the size of the content the repository should deliver the content from the offset to the end of the content.

4.3.3 Extensions

On all input messages and some output messages exists an element called extension. This element is used to provide vendor or repository-specific information between client and server.

All of the types referenced by the schema also support xs:any for vendor or repository-specific information.

4.3.4 Web Services Specific Structures

This binding requires specific structures that are not part of the general CMIS schema.

4.3.4.1 cmisFaultType and cmisFault

`cmisFaultType` and `cmisFault` SHOULD be used to generate SOAP faults.

See section 4.1.4 Reporting Errors.

4.3.4.2 cmisRepositoryEntryType

`cmisRepositoryEntryType` is the return structure of `getRepositories`. It contains the id and the name of a repository.

4.3.4.3 cmisTypeContainer

`cmisTypeContainer` is the return structure of `getTypeDescendants`. It holds a type hierarchy.

4.3.4.4 cmisTypeDefinitionListType

`cmisTypeDefinitionListType` is the return structure of `getTypeChildren`. It contains a list of types, the hasMoreItems flag and the numItem element.

Example:
Request: `webservices/getTypeChildren-request.log`
Response: `webservices/getTypeChildren-response.log`

4.3.4.5 cmisObjectInFolderType, cmisObjectParentsType and cmisObjectInFolderContainerType

`cmisObjectInFolderType` holds, in addition to a `cmisObjectType` object, a path segment string. It is used in all operations that support the includePathSegments parameter. `cmisObjectParentsType` is similar but has a relative path segment string instead of a path segment. For details about path segments and relative path segments see section 2.1.5.3 Paths.

`cmisObjectInFolderContainerType` contains a folder hierarchy.

Example:
Request: `webservices/getChildren-request.log`
Response: `webservices/getChildren-response.log`

Example:
Request: `webservices/getDecendants-request.log`
Response: `webservices/getDecendants-response.log`

4.3.4.6 cmisObjectListType and cmisObjectInFolderListType

`cmisObjectListType` and `cmisObjectInFolderListType` hold lists of `cmisObjectType` and `cmisObjectInFolderType` structures. They also contain the hasMoreItems flag and the numItems element that are returned by operations that return these lists.

4.3.4.7 cmisContentStreamType

`cmisContentStreamType` wraps a content stream and additional information about the stream.

		Client to Repository	Repository to Client
length	Length of the content stream in bytes. If set, it MUST be a positive number. If the length is unknown it MUST NOT be set. If it is used in the context of `appendContentStream` this is the size of the chunk in bytes.	SHOULD be set	SHOULD be set
mimeType	MIME Media Type of the content stream. For the primary content of a document it SHOULD match the value of the property `cmis:contentStreamMimeType`. If it is used in the context of `appendContentStream` this either MUST NOT be set or MUST match the MIME Media Type of the document. If it is the first chunk it SHOULD be set and defines the MIME Media Type of the document.	SHOULD be set	MUST be set
filename	Filename of the content stream. For the primary content of a document it SHOULD match the value of the property `cmis:contentStreamFileName`.	SHOULD be set	SHOULD be set
stream	The content stream. MUST be present even if the content stream has 0 bytes.	MUST be set	MUST be set

4.3.4.8 cmisACLType

`cmisACLType` is the return structure of `getACL` and `applyACL`. It contains the current Access Control List (ACL) of the object and the exact flag that indicates if the ACL fully describes the permission of this object.

Example:
Request: `webservices/getAcl-request.log`
Response: `webservices/getAcl-response.log`

4.3.4.9 cmisExtensionType

`cmisExtensionType` is a placeholder for extensions. See section 4.3.3 Extensions.

5 Browser Binding

5.1 Overview

The CMIS Browser Binding is based upon JSON (Java Script Object Notation, [RFC4627]) and patterns used in Web applications. This binding is specifically designed to support applications running in a web browser but is not restricted to them. It is based on technologies that developers who build such applications already understand, including HTML, HTML Forms, JavaScript and JSON (Java Script Object Notation). Importantly, it does not require a JavaScript library, but rather takes advantage of capabilities already built into modern browsers.

While this binding is optimized for use in browser applications, it can also serve as an easy to use binding in other application models.

5.2 Common Service Elements

5.2.1 Protocol

HTTP MUST be used as the protocol for service requests. HTTP GET MUST be used for reading content and HTTP POST MUST be used for creating, updating and deleting content. Using just those two HTTP verbs makes it possible to develop applications that rely on built-in browser capabilities (e.g. HTML Forms) and typical server configurations.

The use of HTTPS is RECOMMENDED for repositories that contain non-public data.

5.2.2 Data Representation

Browser applications are typically written in JavaScript. A popular lightweight data representation format amongst JavaScript developers is JavaScript Object Notation (JSON) as described in [RFC4627]. JSON MUST be used to represent the various CMIS objects described by the data model.

5.2.3 Schema

This specification provides a formal definition of the CMIS JSON elements. The formal definition is short and precise and allows implementations to validate CMIS JSON at runtime.

Since there is not yet a JSON schema language approved by a standards body, this specification uses a schema language called Orderly that is introduced by Lloyd Hilaiel on http://orderly-json.org/. Since the definition of Orderly on http://orderly-json.org/ may proceed independently of this specification, and because we may need to extend Orderly to define some elements of CMIS, we provide a description of Orderly in appendix B Schema Language (Orderly) of this document.

5.2.4 Mapping Schema Elements to JSON

JSON only defines a few types, including Object, String, Number, Boolean, Null and Array. Not all types used in the CMIS schema have direct JSON equivalents. The following table describes the mapping between

CMIS and JSON types.

CMIS	JSON
string	string
boolean	boolean
decimal	number
integer	number
datetime	number (Dates are represented in milliseconds from 1970/01/01 00:00:00 UTC with a day containing 86,400,000 milliseconds. Negative numbers represent dates before 1970/01/01 00:00:00 UTC.)
uri	string
id	string
html	string

5.2.5 URL Patterns

The URLs used by the Browser Binding are meant to be predictable in order to simplify client development. The URL patterns allow objects to be referenced by either object Id or path. Section 5.3 URLs provides the details of how clients can construct these URLs.

5.2.6 Multipart Forms

Browser applications typically use HTTP multipart forms as described in [RFC2388] to create and update content. This is especially useful for updating file content with the addition of the FILE attribute in [RFC1867]. In this binding, HTTP POST of `multipart/form-data` MUST be used to update content streams.

5.2.7 Properties in a "value not set" state

The JSON value "null" MUST be used by the server when returning values that have not been set.

5.2.8 Callback

Modern browsers restrict a JavaScript function from making HTTP calls to servers other than the one on which the function originated. This set of restrictions is called the "same-origin policy" (see [SameOriginPolicy]). A CMIS client web application and the CMIS repositories it uses may often be deployed to different servers. This specification allows JavaScript clients to access repositories on other servers, within the constraints of the same-origin policy, by using the "JSON with Padding" (JSONP) pattern.

This binding introduces a parameter called `callback` to allow CMIS clients to use this pattern.

The `callback` MAY be included by clients on read operations defined by this protocol that answer a JSON object. The server MUST respond to valid read requests containing this token by answering the token, followed by an open parenthesis, followed by the JSON object returned, followed by a close parenthesis. If the parameter is included in a request, the server MUST validate that its value is not empty but the server MUST NOT do any additional validation of the token, such as, for example, assuring it conforms to JavaScript function naming conventions. If the parameter value is empty, or if the parameter is used on a service for which it is not allowed, then the `invalidArgument` exception MUST be used to signal the error.

Example:

```
showRepositoryInfo (
{
    "A1":{
```

```
        "repositoryId":"A1",
        "repositoryDescription":"A Repository",
        "vendorName":"OASIS",
        "productName":"Repository Server",
        "productVersion":"1.0",
        "cmisVersionSupported":"1.1",
        "changesIncomplete":true,
        "rootFolderUrl":"http:\/\/example.com\/cmis\/repository\/123\/root",
        "latestChangeLogToken":"0",
        "rootFolderId":"100",
        "repositoryName":"Apache Chemistry OpenCMIS InMemory Repository",
        "repositoryUrl":"http:\/\/example.com\/cmis\/repository\/123",
        "changesOnType":[].
        "capabilities":{
            "capabilityContentStreamUpdatability":"anytime",
            "capabilityPWCSearchable":false,
            "capabilityQuery":"bothcombined",
            "capabilityRenditions":"none",
            "capabilityACL":"none",
            "capabilityGetFolderTree":true,
            "capabilityGetDescendants":true,
            "capabilityVersionSpecificFiling":false,
            "capabilityUnfiling":true,
            "capabilityJoin":"none",
            "capabilityAllVersionsSearchable":false,
            "capabilityMultifiling":true,
            "capabilityChanges":"none",
            "capabilityPWCUpdatable":true
        },
    }
}
)
```

5.2.9 Authentication

This specification RECOMMENDS the authentication mechanisms described in the following sections. Repositories MAY provide more, other or no authentication mechanisms.

Furthermore, this specification RECOMMENDS the use of HTTPS (see [RFC2818]) to protect credentials and data.

5.2.9.1 Basic Authentication for Non-Browser Clients

Repositories SHOULD accept HTTP Basic Authentication (see [RFC2617] Section 2).

If the provided credentials are incorrect or unknown or entirely missing, a repository MAY return the HTTP status code 403 (Forbidden) instead of the HTTP status code 401 (Unauthorized). This prevents web browsers from providing a login dialog and subsequently remembering the credentials. This in turn can prevent a form of cross-site request forgery (CSRF).

5.2.9.2 Authentication with Tokens for Browser Clients

The authentication mechanism described in this section addresses the following scenario:

A web application is hosted on one domain; the CMIS browser binding interface is served from another domain. There is no proxy process on the server that hosts the web application. That is, all communication between the application and the repository has to happen in the web browser via JavaScript. The "same-origin policy" (see [SameOriginPolicy]) enforced by the web browser prohibits a direct and secure two-way communication between the application and the repository.

To access the repository, a user has to authenticate and has to authorize the application (and only this application, not all scripts in the web browser) to make CMIS calls.

5.2.9.2.1 JSONP and Form Requests

Cross-domain requests should use JSONP and callbacks (see section 5.2.8 Callback) for GET requests and should use HTML forms for POST requests.

A token SHOULD be added to each request to prevent cross-site request forgery (CSRF) attacks. For this purpose, a parameter `token` MUST be added to the parameters of a GET request and a control `token` MUST be added to the controls of a HTML form.

The repository SHOULD return a `permissionDenied` error if the client sends an invalid token.

If the client sends any other form of authentication (Basic Authentication, OAuth, etc.), the token MAY be omitted. It is RECOMMENDED for web applications always to provide a token, even if another form of authentication is in place.

5.2.9.2.2 Login and Tokens

Tokens are obtained from the repository in the following way.

The repository provides a JavaScript script that the web application includes into its HTML page via the HTML `<script>` tag.

This script provides four functions:

cmisServiceURL() This function returns the Service URL. See section 5.3.1 Service URL.

cmisLogin(callback) This function triggers the login process. The web application MUST call this function before it calls any other functions. How the login works is repository specific. A repository MAY replace the application page in the web browser with a login page and later return back to the application page.

The function takes a callback function. It is called when the login process has been completed. The callback function MUST accept a boolean parameter. The repository MUST provide TRUE if the login process was successful and FALSE if it was not successful.

cmisLogout(callback) This function triggers the logout process. How the logout works is repository specific. After a successful logout, `cmisNextToken()` MUST NOT return a valid token. The application MAY call `cmisLogin()` again to trigger a new login.

The function takes a callback function. It is called when the logout process has been completed. The callback function MUST accept a boolean parameter. The repository MUST provide TRUE if the logout process was successful and FALSE if it was not successful.

cmisNextToken(callback) This function calls the provided callback function with a new token. How this token is generated and obtained is repository specific. Whether the repository returns unique tokens or the same token for a user or for all users is repository specific. The repository SHOULD signal an invalid state (e.g. no logged in user) with an empty token (empty string).

The flow in a web application could looks like this:

```
<script src="http://cmis.example.com/cmis.js"/>
<script>
  cmisLogin(function(success) {
    if (success) {
      displayRootFolder();
    } else {
      showLoginErrorMessage();
    }
  });


  function displayRootFolder() {
    cmisNextToken(function(token) {
      loadChildren('/', ..., token);
    });
  }
```

```
  ...
</script>
```

5.2.10 Error Handling and Return Codes

HTTP status codes MUST be used to indicate success or failure of an operation. Please see the HTTP specification for more information on the HTTP status codes. These are provided as guidance from the HTTP specification. If any conflict arises, the HTTP specification is authoritative.

CMIS Services Exception	HTTP Status Code
General Exceptions	
`invalidArgument`	400
`notSupported`	405
`objectNotFound`	404
`permissionDenied`	403
`runtime`	500
Specific Exceptions	
`constraint`	409
`contentAlreadyExists`	409
`filterNotValid`	400
`nameConstraintViolation`	409
`storage`	500
`streamNotSupported`	403
`updateConflict`	409
`versioning`	409

This binding also introduces an object to return additional information about the response. CMIS repositories SHOULD include this object in responses. When present, the object MUST include the following JSON properties.

string `exception` A string containing one of the CMIS services exceptions describe in section 2.2.1.4 Exceptions.

string `message` A string containing a message that provides more information about what caused the exception.

Example:

```
GET /cmis/repository/123/myFolder?maxItems=abc HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0


HTTP/1.1 400 Bad Request
Content-Type: application/json
Content-Length: xxxx

{
  "exception": "invalidArgument",
  "message": "The parameter 'maxItems' is not valid."
}
```

If the query parameter `suppressResponseCodes=true` is set, the repository MUST always return the HTTP status code 200.

Example:

```
GET /cmis/repository/123/myFolder?maxItems=abc&suppressResponseCodes=true HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0


HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: xxxx

{
  "exception": "invalidArgument",
  "message": "The parameter 'maxItems' is not valid."
}
```

5.2.11 Succinct Representation of Properties

The default representation of properties may contain redundant information such as the display name, property type, cardinality, etc. to simplify JavaScript code in a web application. This extra data is superfluous for clients that cache type and property definitions. In order to reduce the message sizes, the Browser Binding supports a parameter, which advises the repository to send a compact form of the properties.

A client MAY add the query parameter `succinct` (HTTP GET) or the control `succinct` (HTTP POST) with the value "true" to a request. If this is set, the repository MUST return properties in a succinct format. That is, whenever the repository renders an object or a query result, it MUST populate the `succinctProperties` value and MUST NOT populate the `properties` value. See the schema elements `http://docs.oasis-open.org/ns/cmis/browser/201103/object` and `http://docs.oasis-open.org/ns/cmis/browser/201103/queryResult`.

Example of a folder representation with `succinct` flag set to "true":

```
{
  "succinctProperties" : {
    "cmis:allowedChildObjectTypeIds" : ["*"],
    "cmis:objectTypeId" : "cmis:folder",
    "cmis:lastModifiedBy" : "unknown",
    "cmis:path" : "\/My_Folder-0-0\/My_Folder-1-0",
    "cmis:name" : "My_Folder-1-0",
    "cmis:createdBy" : "unknown",
    "cmis:objectId" : "102",
    "cmis:creationDate" : 1342160009207,
    "cmis:changeToken" : "1342160009207",
    "cmis:baseTypeId" : "cmis:folder",
    "cmis:lastModificationDate" : 1342160009207,
    "cmis:parentId" : "101"
  }
}
```

5.3 URLs

5.3.1 Service URL

The document returned by the Service URL provides the repository information for all available repositories. How the client will get the Service URL is repository specific.

The Service URL MUST return the repository infos of all available repositories (see section 2.1.1 Repository). Each repository info MUST contain two additional properties:

- The Repository URL (repository info property: `repositoryUrl`)
- The Root Folder URL (repository info property: `rootFolderUrl`)

5.3.2 Repository URL

The Repository URL provides access to data that is independent of the folder hierarchy such as type definitions, query and content changes. It can be obtained using the `getRepositories` or `getRepositoryInfo` services.

5.3.3 Root Folder URL

The Root Folder URL is used to build Object URLs (see section 5.3.4 Object URLs). It can be obtained using the `getRepositories` or `getRepositoryInfo` services.

5.3.4 Object URLs

An object is either identified by a parameter `objectId` added to the Root Folder URL or by a path that is appended to the Root Folder URL. If the parameter `objectId` is set, it takes precedence over the path.

The two forms of an Object URL are:

- `<root>`?objectId=`<objectId>`
  where `<root>` is the Root Folder URL and `<objectId>` is a CMIS object id.
- `<root>`/`<path>`
  where `<root>` is the Root Folder URL and `<path>` is an absolute CMIS path to an object.

Examples:

- If the Root Folder URL is `http://example.com/cmis/repository/123` and the object id is `1a2b-3c4d-5e6f` then the Object URL is:
  `http://example.com/cmis/repository/123?objectId=1a2b-3c4d-5e6f`
- If the Root Folder URL is `http://example.com/cmis/repository/123` and the object path is `/myFolder/myDocument` then the Object URL is:
  `http://example.com/cmis/repository/123/myFolder/myDocument`

5.4 Services

Read operations use HTTP GET. The particular data that is returned by a read operation is determined by the query parameter `cmisselector`.

If the `cmisselector` parameter is absent, the following default values are used:

Base Type	cmisselector
cmis:document	`content`
cmis:folder	`children`
cmis:relationship	`object`
cmis:policy	`object`
cmis:item	`object`

The value of the `cmisselector` parameter is case insensitive.

All operations that create, modify, or delete objects or change the state of the repository in any way use HTTP POST. Since this binding is optimized for use in browser applications, the format of the transferred data is aligned to the capabilities of HTML forms and described in this specification in HTML terms. See section 5.4.4 Use of HTML Forms for a description of how HTML forms are used for CMIS services.

All operations that return the HTTP status code 201 SHOULD also return a HTTP Location header.

The Schema Elements mentioned in the following sections refer to the CMIS Orderly schema. See also section 5.2.3 Schema.

5.4.1 Service URL

Service:	`getRepositories`
HTTP method:	GET
Argument `cmisselector`:	
Arguments:	• token
Response:	JSON representation of the repository infos of all repositories
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/repositories`
Success HTTP status code:	200 OK

The Service URL has no selector.

Example:
Request: `browser/getRepositoryInfos-request.log`
Response: `browser/getRepositoryInfos-response.log`

5.4.2 Repository URL

5.4.2.1 Selector "repositoryInfo"

Service:	`getRepositoryInfo`
HTTP method:	GET
Argument `cmisselector`:	`repositoryInfo`
Arguments:	• token
Response:	JSON representation of the repository info of the specified repository
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/repositories`
Success HTTP status code:	200 OK

Example:
Request: `browser/getRepositoryInfo-request.log`
Response: `browser/getRepositoryInfo-response.log`

5.4.2.2 Selector "typeChildren"

Service:	`getTypeChildren`
HTTP method:	GET
Argument `cmisselector`**:**	`typeChildren`
Arguments:	• typeId • includePropertyDefinitions • maxItems • skipCount • token
Response:	JSON representation of the types that are immediate children of the specified typeId, or the base types if no typeId is provided
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/typeList`
Success HTTP status code:	200 OK

Example:
Request: `browser/getTypeChildren-request.log`
Response: `browser/getTypeChildren-response.log`

5.4.2.3 Selector "typeDescendants"

Service:	getTypeDescendants
HTTP method:	GET
Argument cmisselector:	typeDescendants
Arguments:	• typeId • depth • includePropertyDefinitions • token
Response:	JSON representation of all types descended from the specified typeId, or all the types in the repository if no typeId is provided
Schema Element:	http://docs.oasis-open.org/ns/cmis/browser/201103/typeContainer
Success HTTP status code:	200 OK

5.4.2.4 Selector "typeDefinition"

Service:	getTypeDefinition
HTTP method:	GET
Argument cmisselector:	typeDefinition
Arguments:	• typeId • token
Response:	JSON representation of the specified type
Schema Element:	http://docs.oasis-open.org/ns/cmis/browser/201103/typeDefinitionType
Success HTTP status code:	200 OK

Example:
Request: browser/getTypeDefinition-request.log
Response: browser/getTypeDefinition-response.log

5.4.2.5 Selector "checkedOut"

Service:	`getCheckedOutDocs`
HTTP method:	GET
Argument `cmisselector`**:**	`checkedOut`
Arguments:	• filter • maxItems • skipCount • orderBy • renditionFilter • includeAllowableActions • includeRelationships • succinct • token
Response:	JSON representation of the documents that have been checked out in the repository
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/objectList`
Success HTTP status code:	200 OK

Example:
Request: `browser/checkOut-request.log`
Response: `browser/checkOut-response.log`

5.4.2.6 Action "createDocument"

Service:	`createDocument`
HTTP method:	POST
Control `cmisaction`:	`createDocument`
Relevant CMIS Controls:	• Single-value Properties • Multi-value Properties • Content • Versioning State • Policies • Adding Access Control Entries (ACEs) • Removing Access Control Entries (ACEs) • Succinct • Token
Response:	JSON representation of the newly created, unfiled document
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/object`
Success HTTP status code:	201 Created

Example:
Request: `browser/createDocument-request.log`
Response: `browser/createDocument-response.log`

5.4.2.7 Action "createDocumentFromSource"

Service:	`createDocumentFromSource`
HTTP method:	POST
Control `cmisaction`:	`createDocumentFromSource`
Relevant CMIS Controls:	• Source Id • Single-value Properties • Multi-value Properties • Content • Versioning State • Policies • Adding Access Control Entries (ACEs) • Removing Access Control Entries (ACEs) • Succinct • Token
Response:	JSON representation of the newly created, unfiled document
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/object`
Success HTTP status code:	201 Created

5.4.2.8 Action "createRelationship"

Service:	createRelationship
HTTP method:	POST
Control cmisaction:	createRelationship
Relevant CMIS Controls:	• Single-value Properties • Multi-value Properties • Policies • Adding Access Control Entries (ACEs) • Removing Access Control Entries (ACEs) • Succinct • Token
Response:	JSON representation of the newly created relationship
Schema Element:	http://docs.oasis-open.org/ns/cmis/browser/201103/object
Success HTTP status code:	201 Created

5.4.2.9 Action "createPolicy"

Service:	createPolicy
HTTP method:	POST
Control cmisaction:	createPolicy
Relevant CMIS Controls:	• Single-value Properties • Multi-value Properties • Policies • Adding Access Control Entries (ACEs) • Removing Access Control Entries (ACEs) • Succinct • Token
Response:	JSON representation of the newly created, unfiled policy
Schema Element:	http://docs.oasis-open.org/ns/cmis/browser/201103/object
Success HTTP status code:	201 Created

5.4.2.10 Action "createItem"

Service:	createItem
HTTP method:	POST
Control cmisaction:	createItem
Relevant CMIS Controls:	• Single-value Properties • Multi-value Properties • Policies • Adding Access Control Entries (ACEs) • Removing Access Control Entries (ACEs) • Succinct • Token
Response:	JSON representation of the newly created, unfiled item
Schema Element:	http://docs.oasis-open.org/ns/cmis/browser/201103/object
Success HTTP status code:	201 Created

5.4.2.11 Action "bulkUpdate"

Service:	bulkUpdateProperties
HTTP method:	POST
Control cmisaction:	bulkUpdate
Relevant CMIS Controls:	• Object Ids • Change Tokens • Single-value Properties • Multi-value Properties • Adding Secondary Type Ids • Removing Secondary Type Ids • Token
Response:	List of ids of the updated objects with their new ids and change tokens
Schema Element:	<Array>http://docs.oasis-open.org/ns/cmis/browser/201103/objectIdAndChangeToken
Success HTTP status code:	200 OK

5.4.2.12 Selector "query"

Service:	query
HTTP method:	GET
Argument cmisselector:	query
Arguments:	• q (maps to the parameter statement) • searchAllVersions • maxItems • skipCount • includeAllowableActions • includeRelationships • renditionFilter • succinct • token
Response:	JSON representation of the results of the query
Schema Element:	http://docs.oasis-open.org/ns/cmis/browser/201103/queryResultList
Success HTTP status code:	200 OK

Example:
Request: browser/doQuery-request.log
Response: browser/doQuery-response.log

5.4.2.13 Action "query"

Service:	query
HTTP method:	POST
Control cmisaction:	query
Relevant CMIS Controls:	• Query • Succinct • Token
Response:	JSON representation of the results of the query
Schema Element:	http://docs.oasis-open.org/ns/cmis/browser/201103/queryResultList
Success HTTP status code:	200 OK

Example:
Request: `browser/doQuery-request.log`
Response: `browser/doQuery-response.log`

5.4.2.14 Selector "contentChanges"

Service:	`getContentChanges`
HTTP method:	GET
Argument `cmisselector`**:**	`contentChanges`
Arguments:	• changeLogToken • includeProperties • includePolicyIds • includeACL • maxItems • succinct • token
Response:	JSON representation of the changed objects. The list object SHOULD contain the next change log token.
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/objectList`
Success HTTP status code:	200 OK

Example:
Request: `browser/getContentChanges-request.log`
Response: `browser/getContentChanges-response.log`

5.4.2.15 Action "createType"

Service:	createType
HTTP method:	POST
Control cmisaction:	createType
Relevant CMIS Controls:	• Type • Token
Response:	JSON representation of the newly created type
Schema Element:	http://docs.oasis-open.org/ns/cmis/browser/201103/typeDefinitionType
Success HTTP status code:	201 Created

5.4.2.16 Action "updateType"

Service:	updateType
HTTP method:	POST
Control cmisaction:	updateType
Relevant CMIS Controls:	• Type • Token
Response:	JSON representation of the updated type
Schema Element:	http://docs.oasis-open.org/ns/cmis/browser/201103/typeDefinitionType
Success HTTP status code:	200 OK

5.4.2.17 Action "deleteType"

Service:	deleteType
HTTP method:	POST
Control cmisaction:	deleteType
Relevant CMIS Controls:	• Type Id • Token
Response:	*empty*
Schema Element:	
Success HTTP status code:	200 OK

5.4.2.18 Selector "lastResult"

HTTP method:	GET
Argument `cmisselector`:	`lastResult`
Arguments:	• token
Response:	See section 5.4.4.4 Access to Form Response Content
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/transaction`
Success HTTP status code:	200 OK

5.4.3 Object URL

5.4.3.1 Selector "children"

Service:	`getChildren`
HTTP method:	GET
Argument `cmisselector`:	`children`
Arguments:	• maxItems • skipCount • filter • includeAllowableActions • includeRelationships • renditionFilter • orderBy • includePathSegment • succinct • token
Response:	JSON representation of the children of the specified folder
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/objectInFolderList`
Success HTTP status code:	200 OK

The selector can be omitted since `getChildren` only works on folders and the selector "children" is the default selector for folders.

Example:
Request: `browser/getChildren-request.log`
Response: `browser/getChildren-response.log`

5.4.3.2 Selector "descendants"

Service:	`getDescendants`
HTTP method:	GET
Argument `cmisselector`**:**	`descendants`
Arguments:	• filter • depth • includeAllowableActions • includeRelationships • renditionFilter • includePathSegment • succinct • token
Response:	JSON representation of the descendants of the specified folder
Schema Element:	`<Array>http://docs.oasis-open.org/ns/cmis/browser/201103/objectContainer`
Success HTTP status code:	200 OK

Example:
Request: `browser/getDecendants-request.log`
Response: `browser/getDecendants-response.log`

5.4.3.3 Selector "folderTree"

Service:	`getFolderTree`
HTTP method:	GET
Argument `cmisselector`:	`folderTree`
Arguments:	• filter • depth • includeAllowableActions • includeRelationships • renditionFilter • includePathSegment • succinct • token
Response:	JSON representation of the folder tree of the specified folder
Schema Element:	`<Array>http://docs.oasis-open.org/ns/cmis/browser/201103/objectContainer`
Success HTTP status code:	200 OK

5.4.3.4 Selector "parent"

Service:	`getFolderParent`
HTTP method:	GET
Argument `cmisselector`:	`parent`
Arguments:	• filter • succinct • token
Response:	JSON representation of the parent folder of the specified folder
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/object`
Success HTTP status code:	200 OK

5.4.3.5 Selector "parents"

Service:	`getObjectParents`
HTTP method:	GET
Argument `cmisselector`**:**	`parents`
Arguments:	• filter • includeRelationships • renditionFilter • includeAllowableActions • includeRelativePathSegment • succinct • token
Response:	JSON representation of the folders that are the parents of the specified object
Schema Element:	`<Array>http://docs.oasis-open.org/ns/cmis/browser/201103/objectParent`
Success HTTP status code:	200 OK

Example:
Request: `browser/getObjectParents-request.log`
Response: `browser/getObjectParents-response.log`

5.4.3.6 Selector "checkedout"

Service:	`getCheckedOutDocs`
HTTP method:	GET
Argument `cmisselector`:	`checkedout`
Arguments:	• filter • maxItems • skipCount • orderBy • renditionFilter • includeAllowableActions • includeRelationships • succinct • token
Response:	JSON representation of the documents that have been checked out in this folder
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/objectList`
Success HTTP status code:	200 OK

5.4.3.7 Action "createDocument"

Service:	`createDocument`
HTTP method:	POST
Control `cmisaction`:	`createDocument`
Relevant CMIS Controls:	• Single-value Properties • Multi-value Properties • Content • Versioning State • Policies • Adding Access Control Entries (ACEs) • Removing Access Control Entries (ACEs) • Succinct • Token
Response:	JSON representation of the newly created document in this folder
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/object`
Success HTTP status code:	201 Created

Example:
Request: `browser/createDocument-request.log`

Response: `browser/createDocument-response.log`

5.4.3.8 Action "createDocumentFromSource"

Service:	`createDocumentFromSource`
HTTP method:	POST
Control `cmisaction`:	`createDocumentFromSource`
Relevant CMIS Controls:	• Source Id • Single-value Properties • Multi-value Properties • Content • Versioning State • Policies • Adding Access Control Entries (ACEs) • Removing Access Control Entries (ACEs) • Succinct • Token
Response:	JSON representation of the newly created document in this folder
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/object`
Success HTTP status code:	201 Created

5.4.3.9 Action "createFolder"

Service:	`createFolder`
HTTP method:	POST
Control `cmisaction`:	`createFolder`
Relevant CMIS Controls:	• Single-value Properties • Multi-value Properties • Policies • Adding Access Control Entries (ACEs) • Removing Access Control Entries (ACEs) • Succinct • Token
Response:	JSON representation of the newly created folder in this folder
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/object`
Success HTTP status code:	201 Created

5.4.3.10 Action "createPolicy"

Service:	`createPolicy`
HTTP method:	POST
Control `cmisaction`:	`createPolicy`
Relevant CMIS Controls:	• Single-value Properties • Multi-value Properties • Policies • Adding Access Control Entries (ACEs) • Removing Access Control Entries (ACEs) • Succinct • Token
Response:	JSON representation of the newly created policy in this folder
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/object`
Success HTTP status code:	201 Created

5.4.3.11 Action "createItem"

Service:	`createItem`
HTTP method:	POST
Control `cmisaction`:	`createItem`
Relevant CMIS Controls:	• Single-value Properties • Multi-value Properties • Policies • Adding Access Control Entries (ACEs) • Removing Access Control Entries (ACEs) • Succinct • Token
Response:	JSON representation of the newly created item in this folder
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/object`
Success HTTP status code:	201 Created

5.4.3.12 Selector "allowableActions"

Service:	`getAllowableActions`
HTTP method:	GET
Argument `cmisselector`**:**	`allowableActions`
Arguments:	• token
Response:	JSON representation of the allowable actions of the specified object
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/allowableActions`
Success HTTP status code:	200 OK

Example:
Request: `browser/getAllowableActions-request.log`
Response: `browser/getAllowableActions-response.log`

5.4.3.13 Selector "object"

Service:	`getObject`
HTTP method:	GET
Argument `cmisselector`**:**	`object`
Arguments:	• filter • includeRelationships • includePolicyIds • renditionFilter • includeACL • includeAllowableActions • succinct • token
Response:	JSON representation of the specified object
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/object`
Success HTTP status code:	200 OK

Example:
Request: `browser/getObject-request.log`
Response: `browser/getObject-response.log`

5.4.3.14 Selector "properties"

Service:	`getProperties`
HTTP method:	GET
Argument `cmisselector`:	`properties`
Arguments:	• filter • succinct • token
Response:	JSON representation of the properties of the specified object
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/properties`
Success HTTP status code:	200 OK

5.4.3.15 Selector "object"

Service:	`getObjectByPath`
HTTP method:	GET
Argument `cmisselector`:	`object`
Arguments:	• filter • includeRelationships • includePolicyIds • renditionFilter • includeACL • includeAllowableActions • succinct • token
Response:	JSON representation of the specified object
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/object`
Success HTTP status code:	200 OK

5.4.3.16 Selector "content"

Service:	`getContentStream`
HTTP method:	GET
Argument `cmisselector`**:**	`content`
Arguments:	• streamId • download • token
Response:	The content stream
Schema Element:	
Success HTTP status code:	200 OK

The selector can be omitted since `getContentStream` only works on documents and the selector `content` is the default selector for documents.

If the request is not authenticated (no authentication HTTP header, no token, etc.) or the authentication details are invalid (wrong credentials, expired token, etc.), the repository MAY redirect the request to a login page (using HTTP status code 307 (Temporary Redirect)).

How the login works is repository specific. If the user can be logged on, the repository either returns the requested document content directly or redirects to a URL that provides the document content.

To control if the web browser should load and show the content or offer it for download, an application can ask the repository to set the HTTP header `Content-Disposition` (see [RFC6266]) by setting the `download` parameter. Valid values are `inline` (default) and `attachment`. These values correspond to the `disposition-type` in [RFC6266]. If the `download` parameter is not provides, the repository SHOULD set a `Content-Disposition` header with the disposition type `inline`.

5.4.3.17 Selector "renditions"

Service:	`getRenditions`
HTTP method:	GET
Argument `cmisselector`**:**	`renditions`
Arguments:	• renditionFilter • maxItems • skipCount • token
Response:	JSON representation of the renditions for the specified object
Schema Element:	`<Array>http://docs.oasis-open.org/ns/cmis/browser/201103/rendition`
Success HTTP status code:	200 OK

5.4.3.18 Action "update"

Service:	`updateProperties`
HTTP method:	POST
Control `cmisaction`:	`update`
Relevant CMIS Controls:	• Single-value Properties • Multi-value Properties • Change Token • Succinct • Token
Response:	JSON representation of the updated object
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/object`
Success HTTP status code:	200 OK, if the object has been updated 201 Created, if a new version has been created

Example:
Request: `browser/updateProperties-request.log`
Response: `browser/updateProperties-response.log`

5.4.3.19 Action "move"

Service:	`moveObject`
HTTP method:	POST
Control `cmisaction`:	`move`
Relevant CMIS Controls:	• Target folder Id • Source folder Id • Succinct • Token
Response:	JSON representation of the moved object
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/object`
Success HTTP status code:	201 Created

5.4.3.20 Action "delete"

Service:	deleteObject
HTTP method:	POST
Control cmisaction:	delete
Relevant CMIS Controls:	• All Versions • Token
Response:	*empty*
Schema Element:	
Success HTTP status code:	200 OK

Example:
Request: browser/deleteObject-request.log
Response: browser/deleteObject-response.log

5.4.3.21 Action "deleteTree"

Service:	deleteTree
HTTP method:	POST
Control cmisaction:	deleteTree
Relevant CMIS Controls:	• All Versions • Unfile Objects • Continue On Failure • Token
Response:	*empty* if successful. A list of Ids if at least one object could not been deleted.
Schema Element:	
Success HTTP status code:	200 OK

When the operation fails, meaning that some objects in the tree are not deleted, an instance of type http://docs.oasis-open.org/ns/cmis/browser/201103/ids containing a list of ids of the objects not deleted MUST be returned.

5.4.3.22 Action "setContent"

Service:	`setContentStream`
HTTP method:	POST
Control `cmisaction`:	`setContent`
Relevant CMIS Controls:	• Overwrite Flag • Change Token • Content • Succinct • Token
Response:	JSON representation of the object
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/object`
Success HTTP status code:	201 Created

5.4.3.23 Action "appendContent"

Service:	`appendContentStream`
HTTP method:	POST
Control `cmisaction`:	`appendContent`
Relevant CMIS Controls:	• IsLastChunk • Change Token • Content • Succinct • Token
Response:	JSON representation of the object
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/object`
Success HTTP status code:	200 OK

5.4.3.24 Action "deleteContent"

Service:	deleteContentStream
HTTP method:	POST
Control cmisaction:	deleteContent
Relevant CMIS Controls:	• Change Token • Succinct • Token
Response:	JSON representation of the object
Schema Element:	http://docs.oasis-open.org/ns/cmis/browser/201103/object
Success HTTP status code:	200 OK

5.4.3.25 Action "addObjectToFolder"

Service:	addObjectToFolder
HTTP method:	POST
Control cmisaction:	addObjectToFolder
Relevant CMIS Controls:	• Folder Id • All Versions • Succinct • Token
Response:	JSON representation of the object
Schema Element:	http://docs.oasis-open.org/ns/cmis/browser/201103/object
Success HTTP status code:	201 Created

5.4.3.26 Action "removeObjectFromFolder"

Service:	`removeObjectFromFolder`
HTTP method:	POST
Control `cmisaction`:	`removeObjectFromFolder`
Relevant CMIS Controls:	• Folder Id • Succinct • Token
Response:	JSON representation of the object
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/object`
Success HTTP status code:	201 Created

Example:
Request: `browser/removeObjectFromFolder-request.log`
Response: `browser/removeObjectFromFolder-response.log`

5.4.3.27 Action "checkOut"

Service:	`checkOut`
HTTP method:	POST
Control `cmisaction`:	`checkOut`
Relevant CMIS Controls:	• succinct • token
Response:	JSON representation of the Private Working Copy Note: The `contentCopied` flag is not returned. Clients may check content related properties such as the `cmis:contentStreamLength` property if the the Private Working Copy has a content stream.
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/object`
Success HTTP status code:	201 Created

Example:
Request: `browser/checkOut-request.log`
Response: `browser/checkOut-response.log`

5.4.3.28 Action "cancelCheckOut"

Service:	`cancelCheckOut`
HTTP method:	POST
Control `cmisaction`:	`cancelCheckOut`
Relevant CMIS Controls:	• token
Response:	*empty*
Schema Element:	
Success HTTP status code:	200 OK

5.4.3.29 Action "checkIn"

Service:	`checkIn`
HTTP method:	POST
Control `cmisaction`:	`checkIn`
Relevant CMIS Controls:	• Major • Single-value Properties • Multi-value Properties • Content • Checkin Comment • Policies • Adding Access Control Entries (ACEs) • Removing Access Control Entries (ACEs) • Succinct • Token
Response:	JSON representation of the new version
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/object`
Success HTTP status code:	201 Created

Example:
Request: `browser/checkIn-request.log`
Response: `browser/checkIn-response.log`

5.4.3.30 Selector "object"

Service:	`getObjectOfLatestVersion`
HTTP method:	GET
Argument `cmisselector`:	`object`
Arguments:	• filter • includeRelationships • includePolicyIds • renditionFilter • includeACL • includeAllowableActions • returnVersion – If no value is present or the value is 'this', `getObject` MUST be called. – If the value is 'latest' `getObjectOfLatestVersion` MUST be called with the parameter `major` set to FALSE. – If the value is 'latestmajor' `getObjectOfLatestVersion` MUST be called with the parameter `major` set to TRUE. • succinct • token
Response:	JSON representation of the specified object
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/object`
Success HTTP status code:	200 OK

5.4.3.31 Selector "properties"

Service:	`getPropertiesOfLatestVersion`
HTTP method:	GET
Argument `cmisselector`**:**	`properties`
Arguments:	• filter • returnVersion – If no value is present or the value is 'this', `getProperties` MUST be called. – If the value is 'latest' `getPropertiesOfLatestVersion` MUST be called with the parameter `major` set to FALSE. – If the value is 'latestmajor' `getPropertiesOfLatestVersion` MUST be called with the parameter `major` set to TRUE. • succinct • token
Response:	JSON representation of the properties of the specified object
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/properties`
Success HTTP status code:	200 OK

5.4.3.32 Selector "versions"

Service:	`getAllVersions`
HTTP method:	GET
Argument `cmisselector`**:**	`versions`
Arguments:	• filter • includeAllowableActions • succinct • token
Response:	JSON representation of all the versions in the Version Series
Schema Element:	`<array>http://docs.oasis-open.org/ns/cmis/browser/201103/object`
Success HTTP status code:	200 OK

Example:
Request: `browser/getAllVersions-request.log`
Response: `browser/getAllVersions-response.log`

5.4.3.33 Selector "relationships"

Service:	`getObjectRelationships`
HTTP method:	GET
Argument `cmisselector`:	`relationships`
Arguments:	• includeSubRelationshipTypes • relationshipDirection • typeId • maxItems • skipCount • filter • includeAllowableActions • succinct • token
Response:	JSON representations of the relationships of the specified object
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/objectList`
Success HTTP status code:	200 OK

5.4.3.34 Selector "policies"

Service:	`getAppliedPolicies`
HTTP method:	GET
Argument `cmisselector`:	`policies`
Arguments:	• filter • succinct • token
Response:	JSON representations of the policies applied to the specified object
Schema Element:	`<array>http://docs.oasis-open.org/ns/cmis/browser/201103/object`
Success HTTP status code:	200 OK

5.4.3.35 Action "applyPolicy"

Service:	`applyPolicy`
HTTP method:	POST
Control `cmisaction`:	`applyPolicy`
Relevant CMIS Controls:	• Policy Id • Succinct • Token
Response:	JSON representation of the updated object
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/object`
Success HTTP status code:	200 OK

5.4.3.36 Action "removePolicy"

Service:	`removePolicy`
HTTP method:	POST
Control `cmisaction`:	`removePolicy`
Relevant CMIS Controls:	• Policy Id • Succinct • Token
Response:	JSON representation of the updated object
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/object`
Success HTTP status code:	200 OK

5.4.3.37 Action "applyACL"

Service:	`applyACL`
HTTP method:	POST
Control `cmisaction`:	`applyACL`
Relevant CMIS Controls:	• Adding Access Control Entries (ACEs) • Removing Access Control Entries (ACEs) • ACL propagation • Token
Response:	JSON representation of the updated ACL
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/acl`
Success HTTP status code:	200 OK

5.4.3.38 Selector "acl"

Service:	`getACL`
HTTP method:	GET
Argument `cmisselector`:	`acl`
Arguments:	• onlyBasicPermissions • token
Response:	JSON representation of the ACL
Schema Element:	`http://docs.oasis-open.org/ns/cmis/browser/201103/acl`
Success HTTP status code:	200 OK

Example:
Request: `browser/getAcl-request.log`
Response: `browser/getAcl-response.log`

5.4.4 Use of HTML Forms

As described in section 5.4 Services HTML forms are used to create, update and delete CMIS content.

The form submission method (HTML form attribute "method") MUST be "POST". If a content stream is not attached to the form, the encoding type (HTML form attribute "enctype") MUST be either `application/x-www-form-urlencoded` or `multipart/form-data`. If a content stream is attached, the encoding type MUST be `multipart/form-data`.

The names of the controls within the form are defined by the patterns in the following sections. All control names are case-insensitive as defined by the HTML specification. Control names MUST be unique within

a form. If the control value of an optional parameter is set to an empty string ("") the default value MUST be used.

A client MAY add controls to a form that are not defined by CMIS as long as the control names don't conflict with the patterns described in this specification.

Since control values are strings, all other data types have to be serialized to strings. The same rules that apply to the serialization to JSON apply here.

5.4.4.1 Action

An HTML form used to POST CMIS content MUST include a control named "cmisaction" that indicates the CMIS operation to be performed. See section 5.4 Services for valid control values. The value of the control is case insensitive.

Example:

```
<input name="cmisaction" type="hidden" value="createDocument" />
```

5.4.4.2 Structured and Array Parameters

Some CMIS operations require structured parameters and arrays of values. Since HTML forms don't support that usage, some CMIS operation parameters are split into multiple controls in a form.

For example, a CMIS property is split into a control that holds the property id and another control that hold property value. The association between the two controls is done by convention.

The entirety of all properties is made up of an array of these property controls.

Names of controls that are part of an array end with "[<index>]" where <index> is a positive integer. Arrays MUST always start with the index 0 and MUST be gapless.

Example:

An array of three properties looks like this in a HTML form:

```
<input name="propertyId[0]"    type="hidden" value="cmis:name" />
<input name="propertyValue[0]" type="text"   value="my document" />

<input name="propertyId[1]"    type="hidden" value="cmis:objectTypeId" />
<input name="propertyValue[1]" type="hidden" value="my:firstObjectType" />

<input name="propertyId[2]"    type="hidden" value="my:intProperty" />
<input name="propertyValue[2]" type="text"   value="42" />
```

If a client sends invalid, incomplete or inconsistent data the repository SHOULD throw an `invalidArgument` exception.

5.4.4.3 CMIS Controls

This section lists all HTML form controls used by CMIS services.

5.4.4.3.1 Succinct

This flag indicates that the property presentation must be succinct. See section 5.2.11 Succinct Representation of Properties.

Control name: `succinct`

Control value: "true" or not set

Example:

```
<input name="succinct" type="hidden" value="true" />
```

5.4.4.3.2 Token

This is used to authorize the request. See section 5.2.9.2 Authentication with Tokens for Browser Clients.

Control name: `token`
Control value: Token String

Example:

```
<input name="token" type="hidden" value="e45ab575d6fe4aab901e9" />
```

5.4.4.3.3 Object Id

This is used if one object should be addressed.

Control name: `objectId`
Control value: Object Id

Example:

```
<input name="objectId" type="hidden" value="1234-abcd-5678" />
```

5.4.4.3.4 Object Ids

This is used if multiple objects should be addressed. (Only applies to `bulkUpdateProperties`.)

Control name: `objectId[<idIndex>]`
Control value: Object Id

Example:

```
<input name="objectId[0]" type="hidden" value="1234-abcd-5678" />
<input name="objectId[1]" type="hidden" value="9876-bcde-5432" />
<input name="objectId[2]" type="hidden" value="7654-qwer-2345" />
```

5.4.4.3.5 Folder Id

Control name: `folderId`
Control value: Folder Id

Example:

```
<input name="folderId" type="hidden" value="1234-abcd-5678" />
```

5.4.4.3.6 Source Id

Control name: `sourceId`
Control value: Source Id

Example:

```
<input name="sourceId" type="hidden" value="1234-abcd-5678" />
```

5.4.4.3.7 Source folder Id

Control name: `sourceFolderId`
Control value: Folder Id

Example:

```
<input name="sourceFolderId" type="hidden" value="1234-abcd-5678" />
```

5.4.4.3.8 Target folder Id

Control name: `targetFolderId`
Control value: Folder Id

Example:

```
<input name="targetFolderId" type="hidden" value="1234-abcd-5678" />
```

5.4.4.3.9 Policy Id

Control name: `policyId`
Control value: Policy Id

Example:

```
<input name="policyId" type="hidden" value="1234-abcd-5678" />
```

5.4.4.3.10 Type Id

Control name: `typeId`
Control value: Object-type Id

Example:

```
<input name="typeId" type="hidden" value="my:type" />
```

5.4.4.3.11 Single-value Properties

A single-value property is made up of a pair of a `propertyId` control and a `propertyValue` control with the same `<propIndex>`. To unset the property, the client must submit form data in which the `propertyId` MUST be present and the `propertyValue` control MUST NOT be present.

`<propIndex>` does not imply any order.

5.4.4.3.11.1 Property Id

Control name: `propertyId[<propIndex>]`
Control value: Property Id

5.4.4.3.11.2 Property Value

Control name: `propertyValue[<propIndex>]`
Control value: Property Value

Example:

```
<input name="propertyId[0]"    type="hidden" value="my:firstname" />
<input name="propertyValue[0]" type="text"   value="John" />

<input name="propertyId[1]"    type="hidden" value="my:lastname" />
<input name="propertyValue[1]" type="text"   value="Smith" />
```

5.4.4.3.12 Multi-value Properties

A multi-value property is made up of a `propertyId` control and a series of `propertyValue` controls with the same `<propIndex>`. The `propertyValue` controls MUST have a second index `<seqIndex>`. This array MUST also always start with the index 0 and MUST be gapless. To unset the property, NO `propertyValue` control MUST be present.

`<propIndex>` does not imply any order, but `<seqIndex>` defines the order of the values.

5.4.4.3.12.1 Property Id

Control name: `propertyId[<propIndex>]`
Control value: Property Id

5.4.4.3.12.2 Property Value

Control name: `propertyValue[<propIndex>][<seqIndex>]`
Control value: Property value at position `<seqIndex>`

Example:

```
<input name="propertyId[0]"       type="hidden" value="my:countries" />
<input name="propertyValue[0][0]" type="text"   value="Germany" />
<input name="propertyValue[0][1]" type="text"   value="United States" />
<input name="propertyValue[0][2]" type="text"   value="France" />
<input name="propertyValue[0][3]" type="text"   value="United Kingdom" />
```

```
<input name="propertyValue[0][4]" type="text"   value="Switzerland" />

<input name="propertyId[1]"       type="hidden" value="my:colors" />
<input name="propertyValue[1][0]" type="text"   value="red" />
<input name="propertyValue[1][1]" type="text"   value="green" />
<input name="propertyValue[1][2]" type="text"   value="blue" />
```

5.4.4.3.13 Adding Secondary Type Ids

Secondary type ids can be added by providing a list of ids.

Control name: `addSecondaryTypeId[<typeIndex>]`

Control value: Secondary Type Id

Example:

```
<input name="addSecondaryTypeId[0]" type="hidden" value="my:audit" />
<input name="addSecondaryTypeId[1]" type="hidden" value="my:comment" />
```

5.4.4.3.14 Removing Secondary Type Ids

Secondary type ids can be removed by providing a list of ids.

Control name: `removeSecondaryTypeId[<typeIndex>]`

Control value: Secondary Type Id

Example:

```
<input name="removeSecondaryTypeId[0]" type="hidden" value="my:favorite" />
<input name="removeSecondaryTypeId[1]" type="hidden" value="my:workflow123" />
```

5.4.4.3.15 Adding Access Control Entries (ACEs)

In order to add an ACE to a CMIS object, a client passes a control named `addACEPrincipal` along with a set of corresponding `addACEPermission` controls. An index value `<addACEIndex>` links the principal with its permissions, and a second index `<permIndex>` differentiates the permissions.

`<addACEIndex>` and `<permIndex>` don't imply any order.

5.4.4.3.15.1 Principal

Control name: `addACEPrincipal[<addACEIndex>]`

Control value: Principal Id

5.4.4.3.15.2 Permission

Control name: `addACEPermission[<addACEIndex>][<permIndex>]`

Control value: Permission String

Example:

```
<input name="addACEPrincipal[0]"     type="hidden" value="john" />
<input name="addACEPermission[0][0]" type="hidden" value="cmis:read" />
<input name="addACEPermission[0][1]" type="hidden" value="perm:publish" />

<input name="addACEPrincipal[1]"     type="hidden" value="mary" />
<input name="addACEPermission[1][0]" type="hidden" value="cmis:all" />
```

5.4.4.3.16 Removing Access Control Entries (ACEs)

In order to remove an ACE to a CMIS object, a client passes a control named `removeACEPrincipal` along with a set of corresponding `removeACEPermission` controls. An index value `<removeACEIndex>` links the principal with its permissions, and a second index `<permIndex>` differentiates the permissions.

`<removeACEIndex>` and `<permIndex>` don't imply any order.

5.4.4.3.16.1 Principal

Control name: `removeACEPrincipal[<removeACEIndex>]`
Control value: Principal Id

5.4.4.3.16.2 Permission

Control name: `removeACEPermission[<removeACEIndex>][<permIndex>]`
Control value: Permission String

Example:

```
<input name="removeACEPrincipal[0]"     type="hidden" value="tom" />
<input name="removeACEPermission[0][0]" type="hidden" value="cmis:write" />
<input name="removeACEPermission[0][1]" type="hidden" value="perm:publish" />

<input name="removeACEPrincipal[1]"     type="hidden" value="bob" />
<input name="removeACEPermission[1][0]" type="hidden" value="perm:forward" />
```

5.4.4.3.17 ACL propagation

In order to specify how to propagate ACE's, a control named `ACLPropagation` is used.

Control name: `ACLPropagation`
Control value: ACL propagation enum ("objectonly", "propagate", "repositorydetermined")

Example:

```
<input name="ACLPropagation" type="hidden" value="propagate" />
```

5.4.4.3.18 Policies

Policies are assigned and removed to CMIS objects by including a control named `policy` with an index of `<policyIndex>`. A policy list is made up of a series of these `policy` controls.

`<policyIndex>` does not imply any order.

Control name: `policy[<policyIndex>]`

Control value: Policy Id

Example:

```
<input name="policy[0]" type="hidden" value="1111-aaaa-2222" />
<input name="policy[1]" type="hidden" value="3333-bbbb-4444" />
<input name="policy[2]" type="hidden" value="5555-cccc-6666" />
```

5.4.4.3.19 Change Token

A CMIS change token is included by using a form control named `changeToken`. If the value of the control is set to the empty string, then the repository MUST treat the change token as not set.

Control name: `changeToken`

Control value: Change Token

Example:

```
<input name="changeToken" type="hidden" value="8923653942" />
```

5.4.4.3.20 Change Tokens

If multiple objects are addressed, this list of change tokens matches the list of object ids (see section 5.4.4.3.4 Object Ids). The index of each change token MUST match the index of the corresponding object id.

The rules defined in section 5.4.4.3.19 Change Token apply to each change token.

Control name: `changeToken[<idIndex>]`

Control value: Change Token

Example:

```
<input name="changeToken[0]" type="hidden" value="8923653942" />
<input name="changeToken[1]" type="hidden" value="1234567890" />
<input name="changeToken[2]" type="hidden" value="5555555555" />
```

5.4.4.3.21 All Versions

Indicates if only this version or all versions should be affected.

Control name: `allVersions`

Control value: Boolean ("true", "false")

Example:

```
<input name="allVersions" type="hidden" value="true" />
```

5.4.4.3.22 Unfile Objects

Indicates how `deleteTree` should delete objects.

Control name: `unfileObjects`
Control value: enumUnfileObject ("unfile", "deletesinglefiled", "delete")

Example:

```
<input name="unfileObjects" type="hidden" value="delete" />
```

5.4.4.3.23 Continue On Failure

Indicates if `deleteTree` should continue on failure.

Control name: `continueOnFailure`
Control value: Boolean ("true", "false")

Example:

```
<input name="continueOnFailure" type="hidden" value="true" />
```

5.4.4.3.24 Overwrite Flag

Indicates if `setContentStream` should overwrite the existing content.

Control name: `overwriteFlag`
Control value: Boolean ("true", "false")

Example:

```
<input name="overwriteFlag" type="hidden" value="true" />
```

5.4.4.3.25 IsLastChunk

Indicates if `appendContentStream` should consider this chunk the last chunk of the document content.

Control name: `isLastChunk`
Control value: Boolean ("true", "false")

Example:

```
<input name="isLastChunk" type="hidden" value="true" />
```

5.4.4.3.26 Major

Indicates if the major or minor version is expected.

Control name: `major`
Control value: Boolean ("true", "false")

Example:

```
<input name="major" type="hidden" value="true" />
```

5.4.4.3.27 Versioning State

When a document is checked in, a control named `versioningState` is used to set the versioning state.

Control name: `versioningState`
Control value: Versioning state enum ("none", "major", "minor", "checkedout")

Example:

```
<input name="versioningState" type="hidden" value="major" />
```

5.4.4.3.28 Checkin Comment

When a document is checked in, a control named `checkinComment` is used to include the checkin comment.

Control name: `checkinComment`
Control value: Checkin comment

Example:

```
<input name="checkinComment" type="text" value="My comment" />
```

5.4.4.3.29 Query

A CMIS query can be constructed using a control named `statement` and set of controls to specify the query options.

5.4.4.3.29.1 Statement

Control name: `statement`
Control value: CMIS query statement

5.4.4.3.29.2 Search all versions

Control name: `searchAllVersions`
Control value: Boolean ("true", "false")

5.4.4.3.29.3 Include relationships

Control name: `includeRelationships`
Control value: includeRelationships enum ("none", "source" ,"target", "both")

5.4.4.3.29.4 Rendition filter

Control name: `renditionFilter`
Control value: Rendition filter. See section 2.2.1.2.4 Renditions.

5.4.4.3.29.5 Include allowable actions

Control name: `includeAllowableActions`
Control value: Boolean ("true", "false")

5.4.4.3.29.6 Max Items

Control name: `maxItems`
Control value: Non-negative integer. See section 2.2.1.1 Paging.

5.4.4.3.29.7 Skip Count

Control name: `skipCount`
Control value: Non-negative integer. See section 2.2.1.1 Paging.

Example:

```
<input name="statement" type="text"   value="SELECT * FROM cmis:document" />
<input name="searchAllVersions"       type="hidden" value="false" />
<input name="includeRelationships"    type="hidden" value="none" />
<input name="renditionFilter"         type="hidden" value="cmis:none" />
<input name="includeAllowableActions" type="hidden" value="false" />
<input name="maxItems"                type="hidden" value="100" />
<input name="skipCount"               type="hidden" value="0" />
```

5.4.4.3.30 Content

A "file" select control SHOULD be used to attach content. See [RFC1867].

Control name: `content`
Control value: *none*

```
<input name="content" type="file" />
```

5.4.4.3.31 Type

Control name: `type`
Control value: JSON representation of the type definition.
Schema element:
`http://docs.oasis-open.org/ns/cmis/browser/201103/typeDefinitionType`

```
<textarea name="type">
{
    "id":"my:documentType",
    "baseId":"cmis:document",
    "parentId":"cmis:document",
    "displayName":"My Document Type",
    "description":"My new type",
    "localNamespace":"local",
    "localName":"my:documentType",
    "queryName":"my:documentType",
    "fileable":true,
```

```
    "includedInSupertypeQuery":true,
    "creatable":true,
    "fulltextIndexed":false,
    "queryable":false,
    "controllableACL":true,
    "controllablePolicy":false,
    "propertyDefinitions":{
        "my:stringProperty":{
            "id":"my:stringProperty",
            "localNamespace":"local",
            "localName":"my:stringProperty",
            "queryName":"my:stringProperty",
            "displayName":"My String Property",
            "description":"This is a String.",
            "propertyType":"string",
            "updatability":"readwrite",
            "inherited":false,
            "openChoice":false,
            "required":false,
            "cardinality":"single",
            "queryable":true,
            "orderable":true,
        }
    }
}
</textarea>
```

5.4.4.4 Access to Form Response Content

JSON response content is subject to the same security constraints as any other kind of JavaScript which means that a browser will not allow JavaScript in a page to access JSON objects included in an HTML Form response if that response came from a different domain than the rest of the page content. For example, suppose a browser displayed an HTML Form from Web Server `foo.example.com` to create a document in a CMIS repository on server `bar.example.com`. When the user submits the form, there is no way for the page to access the JSON object representing the new document created as a response to the submitted form.

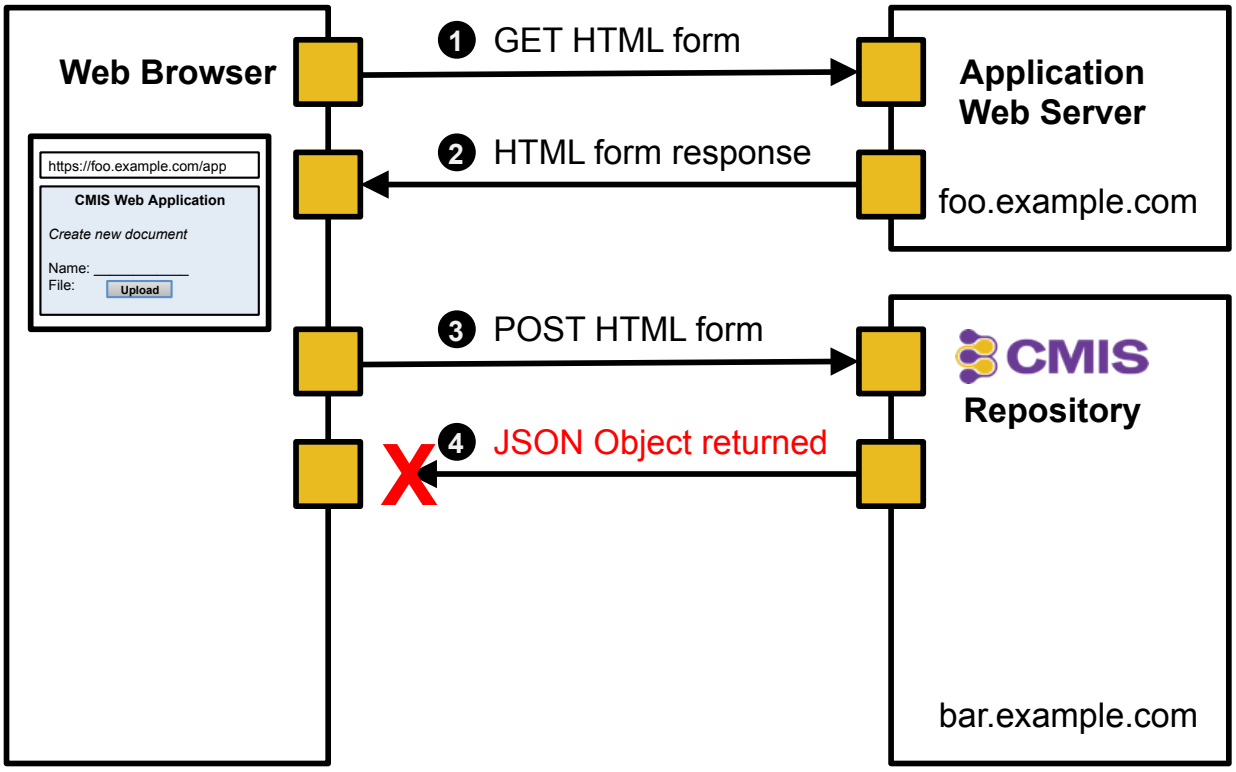

Figure 5.1: Web application cannot retrieve the JSON object returned by the repository

To make it possible for a browser client to access the JSON content answered from the CMIS repository, the repository MUST keep the core result details (the status code, object id and potential error message) of a completed request and make those details available to the client in a later request.

To correlate the result of a CMIS request with the later call to retrieve the result of that request, the "token" is used (see section 5.2.9.2 Authentication with Tokens for Browser Clients).

After the operation has been performed, the client can retrieve the result by sending a HTTP GET requested to the Repository URL with the selector set to `lastResult` and an parameter `token` which is set to the same token string previously sent with the form.

The result details MUST be answered as a JSON object containing these elements.

integer `code` An integer containing the HTTP status code for the operation.

string `objectId` A string containing the id of the object, if the operation was successful. If the operation was not successful, the value of this string is undefined.

string `exception` A string containing the exception, if the operation was not successful. If the operation was successful, the value of this string is undefined.

string `message` A string containing the error message, if the operation was not successful. If the operation was successful, the value of this string is undefined.

The result details SHOULD

- only be available to the same client (as defined by the client's IP address) that called the operation.
- not be kept longer than an hour, since they are supposed to be retrieved immediately after the operation by the client.
- only be retrievable once. That is, a second attempt SHOULD return an `invalidArgument` error (code = 0).

If the value of the parameter `token` is invalid, the "code" field of this JSON object MUST be set to 0.

If the `token` control is not specified in the form, the repository does not need to keep the result details because there is no way for the client to retrieve them. Note that in this case some other form of authentication SHOULD be in place.

If the `token` control is specified, the repository MUST return the HTTP status code 200 and a HTML page. This is necessary to avoid problems with certain web browsers, which cannot display a JSON response or ask the end user to save the JSON response. Since the purpose of this method is to fetch the result of the request at a later point in time, the immediate response doesn't matter and can be empty.

Example:

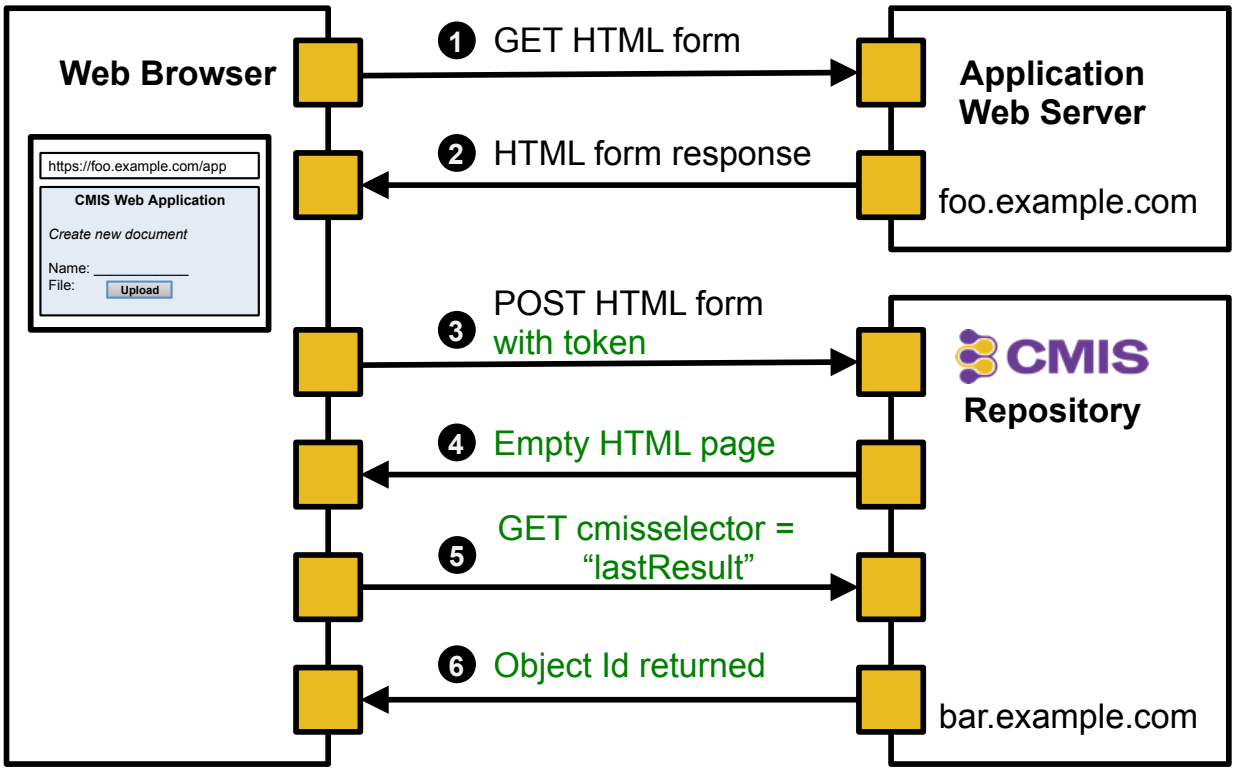

Figure 5.2: Selector "lastResult" returns the last service result from the repository

When the client submits the HTML form, it can include a form control with the name "token" like this:

```
<input name="token" type="hidden" value="e45ab575d6fe4aab901e9" />
```

Soon thereafter, the client could retrieve the results of the form post by making a request like this

```
http://example.com/cmis/repository/123?cmisselector=lastResult&
callback=showNewDocumentId&token=e45ab575d6fe4aab901e9
```

and then, the repository would answer a JSON object that contains the result details, like

```
{
   "code" : 201,
   "objectId" : "1234-abcd-5678",
   "exception" : null,
   "message" : null
}
```

The client can then retrieve the details for the object using its objectId.

5.4.4.4.1 Client Implementation Hints

Whenever the `token` control is used, the repository must respond with a HTML page. The content of this page is not defined in this specification and might be empty. In general, the response is not useful for an end-user.

Therefore, clients should redirect the response to a hidden HTML iframe. The iframe's `onLoad` event can be used as an operation status notification. When it is triggered the operation is complete on the repository side and it is safe then to retrieve the results.

5.4.4.4.2 Server Implementation Hints

The use of this method can make CMIS stateful since the server has to remember details of a previous service request. However, the state can be kept entirely on the client, to eliminate the need for the server to be stateful at all.

5.4.4.4.2.1 State on Server

Result details are non-permanent data and don't need to be persisted. The repository might store the state in-memory or in shared session state.

When a repository receives a `lastResult` request it should check the IP address of the client and the expiration time of the result details before it replies. This ensures that the data is not being retrieved by a malicious client, and that the requested data is relevant.

5.4.4.4.2.2 State on Client

The state can be managed on the client side using browser cookies, which keeps the repository stateless.

When a `token` control is sent with the form data, the repository can attach a cookie to its POST response. The cookie name is derived from the `token` value and the cookie value would contain the result details.

When the repository receives a `lastResult` request, it also receives the cookies from the browser. So, if the repository can find a cookie that matches the `token` parameter value it can send back the cookie value and delete the cookie. If there is no corresponding cookie, it can reply with an error message.

Since the browser takes care of the cookie expiration and cookies can only be sent from the originating client, there are no new additional security and lifecycle issues for the repository to handle.

6 Conformance

Specification:

This specification references a number of other specifications. In order to comply with this specification, an implementation MUST implement the portions of referenced specifications necessary to comply with the required provisions of this specification. Additionally, the implementation of the portions of the referenced specifications that are specifically cited in this specification MUST comply with the rules for those portions as established in the referenced specification.
An implementation conforms to this specification if it satisfies all of the MUST or REQUIRED level requirements defined within this specification.

Domain Model:

Normative text within this specification takes precedence over the CMIS schemas.
That is, the normative text in this specification further constrains the schemas and/or WSDL that are part of this specification; and this specification contains further constraints on the elements defined in referenced schemas.

Clients:

Client implementations MAY implement the AtomPub Binding or the Web Services Binding or the Browser Binding.

Repositories:

Repositories MUST implement the following CMIS protocol bindings:

- AtomPub Binding
- Web Services Binding

Repositories SHOULD implement the following CMIS protocol binding:

- Browser Binding

AtomPub Binding:

This specification references a number of other specifications. In order to comply with this specification, an implementation MUST implement the portions of referenced specifications necessary to comply with the required provisions of this specification. Additionally, the implementation of the portions of the referenced specifications that are specifically cited in this specification MUST comply with the rules for those portions as established in the referenced specification. Additionally normative text within this specification takes precedence over the CMIS RestAtom XML Schema. That is, the normative text in this specification further constrains the schemas that are part of this specification; and this specification contains further constraints on the elements defined in referenced schemas.
The CMIS RestAtom XML takes precedence over any examples or non-normative outlines included either in this document or as standalone examples.

Web Services Binding:

Normative text within this specification takes precedence over the CMIS Messaging XML and CMIS WSDL. That is, the normative text in this specification further constrains the schemas and WSDL that are part of this specification; and this specification contains further constraints on the elements defined in referenced schemas.

The CMIS Messaging XML and CMIS WSDL takes precedence over any examples or non-normative outlines included either in this document or as standalone examples.

Browser Binding:

Normative text within this specification takes precedence over the CMIS Orderly Schema. That is, the normative text in this specification further constrains the schema that is part of this specification; and this specification contains further constraints on the elements defined in the referenced schema.

The CMIS Orderly Schema takes precedence over any examples or non- normative outlines included either in this document or as standalone examples.

Appendix A. IANA Considerations

A.1 Content-Type Registration

A.1.1 CMIS Query

A CMIS Query Document, when serialized as XML 1.0, can be identified with the following media type:

MIME media type name: application

MIME subtype name: cmisquery +xml

Mandatory parameters: None

Optional parameters: "charset": This parameter has semantics identical to the charset parameter of the "application/xml" media type as specified in [RFC3023].

Encoding considerations: Identical to those of "application/xml" as described in [RFC3023], Section 3.2.

Security considerations: As defined in this specification. In addition, as this media type uses the "+xml" convention, it shares the same security considerations as described in [RFC3023], Section 10.

Interoperability considerations: There are no known interoperability issues.

Published specification: This specification.

Applications that use this media type: No known applications currently use this media type.

Additional information:

Magic number(s): As specified for "application/xml" in [RFC3023], Section 3.2.

File extension: .cmisquery

Fragment identifiers: As specified for "application/xml" in [RFC3023], Section 5.

Base URI: As specified in [RFC3023], Section 6.

Macintosh File Type code: TEXT

Person and email address to contact for further information: OASIS CMIS TC <cmis@lists.oasis-open.org>

Intended usage: COMMON

Author/Change controller: IESG

A.1.2 CMIS AllowableActions

A CMIS Allowable Actions Document, when serialized as XML 1.0, can be identified with the following media type:

MIME media type name: application

MIME subtype name: cmisallowableactions +xml

Mandatory parameters: None.

Optional parameters: "charset": This parameter has semantics identical to the charset parameter of the "application/xml" media type as specified in [RFC3023].

Encoding considerations: Identical to those of "application/xml" as described in [RFC3023], Section 3.2.

Security considerations: As defined in this specification. In addition, as this media type uses the "+xml" convention, it shares the same security considerations as described in [RFC3023], Section 10.

Interoperability considerations: There are no known interoperability issues.

Published specification: This specification.

Applications that use this media type: No known applications currently use this media type.

Additional information:

Magic number(s): As specified for "application/xml" in [RFC3023], Section 3.2.

File extension: .cmisallowableactions

Fragment identifiers: As specified for "application/xml" in [RFC3023], Section 5.

Base URI: As specified in [RFC3023], Section 6.

Macintosh File Type code: TEXT

Person and email address to contact for further information: OASIS CMIS TC <cmis@lists.oasis-open.org>

Intended usage: COMMON

Author/Change controller: IESG

A.1.3 CMIS Tree

A CMIS Tree Document, when serialized as XML 1.0, can be identified with the following media type:

MIME media type name: application

MIME subtype name: cmistree +xml

Mandatory parameters: None.

Optional parameters: "charset": This parameter has semantics identical to the charset parameter of the "application/xml" media type as specified in [RFC3023].

Encoding considerations: Identical to those of "application/xml" as described in [RFC3023], Section 3.2.

Security considerations: As defined in this specification. In addition, as this media type uses the "+xml" convention, it shares the same security considerations as described in [RFC3023], Section 10.

Interoperability considerations: There are no known interoperability issues.

Published specification: This specification.

Applications that use this media type: No known applications currently use this media type.

Additional information:

Magic number(s): As specified for "application/xml" in [RFC3023], Section 3.2.

File extension: .cmistree

Fragment identifiers: As specified for "application/xml" in [RFC3023], Section 5.

Base URI: As specified in [RFC3023], Section 6.

Macintosh File Type code: TEXT

Person and email address to contact for further information: OASIS CMIS TC <cmis@lists.oasis-open.org>

Intended usage: COMMON

Author/Change controller: IESG

A.1.4 CMIS Atom

A CMIS Atom Document, when serialized as XML 1.0, can be identified with the following media type:

MIME media type name: application

MIME subtype name: cmisatom +xml

Mandatory parameters: None.

Optional parameters: "charset": This parameter has semantics identical to the charset parameter of the "application/xml" media type as specified in [RFC3023]. "type": This parameter has semantics identical to the type parameter of the "application/atom+xml" as specified in [RFC4287]

Encoding considerations: Identical to those of "application/xml" as described in [RFC3023], Section 3.2.

Security considerations: As defined in this specification. In addition, as this media type uses the "+xml" convention, it shares the same security considerations as described in [RFC3023], Section 10.

Interoperability considerations: There are no known interoperability issues.

Published specification: This specification.

Applications that use this media type: No known applications currently use this media type.

Additional information:

Magic number(s): As specified for "application/xml" in [RFC3023], Section 3.2.

File extension: .cmisatom

Fragment identifiers: As specified for "application/xml" in [RFC3023], Section 5.

Base URI: As specified in [RFC3023], Section 6.

Macintosh File Type code: TEXT

Person and email address to contact for further information: OASIS CMIS TC <cmis@lists.oasis-open.org>

Intended usage: COMMON

Author/Change controller: IESG

Please see section 3.1.1 on why this media type is needed above the Atom Media Type.

A.1.5 CMIS ACL

A CMIS ACL Document, when serialized as XML 1.0, can be identified with the following media type:

MIME media type name: application

MIME subtype name: cmisacl +xml

Mandatory parameters: None.

Optional parameters: "charset": This parameter has semantics identical to the charset parameter of the "application/xml" media type as specified in [RFC3023].

Encoding considerations: Identical to those of "application/xml" as described in [RFC3023], Section 3.2.

Security considerations: As defined in this specification. In addition, as this media type uses the "+xml" convention, it shares the same security considerations as described in [RFC3023], Section 10.

Interoperability considerations: There are no known interoperability issues.

Published specification: This specification.

Applications that use this media type: No known applications currently use this media type.

Additional information:

Magic number(s): As specified for "application/xml" in [RFC3023], Section 3.2.

File extension: .cmisacl

Fragment identifiers: As specified for "application/xml" in [RFC3023], Section 5.

Base URI: As specified in [RFC3023], Section 6.

Macintosh File Type code: TEXT

Person and email address to contact for further information: OASIS CMIS TC <cmis@lists.oasis-open.org>

Intended usage: COMMON

Author/Change controller: IESG

Appendix B. Schema Language (Orderly)

We wish to thank Lloyd Hilaiel for his work in defining the Orderly language, and express our gratitude for allowing the use of Orderly in this specification.

The following is a description of the Orderly Language. In this description, we have liberally copied sections from the original Orderly definition from http://Orderly-json.org/. In some cases, there may be differences between the description here and the description from http://Orderly-json.org/. In any case, the description of Orderly in this specification SHALL be used to describe the JSON elements of this specification.

B.1 Overview

Orderly is an ergonomic micro-language that can represent a subset of JSONSchema. Orderly is designed to feel familiar to the average programmer and to be extremely easy to learn and remember. This document provides a conversational overview of Orderly as well as a normative grammar.

B.2 A subset of JSONSchema

JSONSchema attempts to provide a representation for three distinct types of information about JSON structures:

- Data structure (for documentation and validation purposes)
- Storage attributes (information pertinent to tools that wish to persist JSON data)
- Interaction Control (providing hints on how to render a UI where data can be manipulated).

Orderly purposefuly ignores all features of JSONSchema which aren't useful for validation, including the following attributes:

- options (label/value)
- title
- description
- transient
- hidden
- disallow
- extends
- identity

An exhaustive list of the differences between Orderly and JSONSchema is below.

B.3 A Non-Normative Tutorial

A collection of Non-normative examples of Orderly:

B.3.1 Comments and Whitespace

Orderly supports comments, comments are initiated with either '#' or '//' and continue to the first encountered newline ('\n').

Orderly doesn't rely overmuch on whitespace, leaving the decision of how to format your schema up to you.

B.3.2 Property Names

Property names may be anything that is allowed inside JSON strings. Unlike JSON itself, however, Orderly provides a shorthand that allows a subset of strings to be represented without quotes.

For instance these are all valid Orderly:

```
string foo;
string "foo";
string "this is a property name with spaces";
```

B.3.3 Common Properties

From the JSONSchema specification, four options exist which apply to all data types:

The optional property indicates a value which is not required in a conformant JSON instance. Optional values are represented in Orderly with a trailing question mark:

```
string name?;
string "name"?;
```

The requires property says that if a value is present in the instance JSON, another named value MUST also be present. In Orderly a requirement on another type is expressed by placing the property name (optionally quoted) enclosed in angle brackets at the end of a type definition:

```
string town <state>;
```

Multiple properties MAY be required, and SHOULD be separated with commas:

```
string town <state,zip>;
```

The enum property specifies a set of allowable values for a key in the JSON document.

```
string mood [ "happy", "sad", "meh" ];
integer secretOfLife [ 7, 42 ];
```

In a JSONSchema document the default property specifies a default value for a property. One could imagine that as an input object passes validation it will be automatically augmented with default values for required properties missing in the instance object. The specification of default values in Orderly looks something like assignment in most programming languages:

```
string mood [ "happy", "sad", "meh" ] = "happy"; # optimistically default to "happy"
```

B.3.4 String Types

Strings are specified in Orderly using the string type specifier. Strings in JSONSchema support "minLength" and "maxLength" properties, which are represented in Orderly using curly braces immediately after the type:

```
string{4,12} login;
```

Omission of a specification of either minimum or maximum is allowed:

```
string{4,} login; # login requires at least 4 chars
string{,32} name; # name may not be longer than 32 chars
```

Regular expressions are supported in JSONSchema for string values. In Orderly you may directly provide a regular expression using '/' syntax to denote the beginning and end of the regular expression:

```
string mood /^((happy)|(sad)|(meh))$/;
```

B.3.5 Number and Integer types

Numbers are specified in Orderly using the number type specifier. In JSONSchema numbers and integers support ranges, in Orderly these ranges for numbers are specified in the same way we specify ranges for strings:

```
number{0.02, 0.98} numNum;
integer{0,10} rating
```

Syntactically, numbers in Orderly follow the same rules as numbers in JSON.

B.3.6 Boolean Types

Boolean types are represented in Orderly using the boolean type specifier:

```
boolean iShouldStay;
```

B.3.7 Object Types

Objects are represented in Orderly using the object type specifier:

```
object {
    string foo;
    integer bar;
    number baz;
};
```

Object definitions may be "closed", meaning that properties that are not explicitly mentioned are not allowed, or "open". A trailing star (*) indicates an "open" object defintion:

```
object {
    string foo;
    # whatever other properties you want, thanks to that star
}*;
```

B.3.8 Array Types

Arrays are specified using the array type specifier. Schemas for arrays elements may be specified in one of two ways. First, we can specify a single schema that governs all array members, with the schema enclosed by square brackets:

```
array [
    numbers{0.00, 1.00};
] weights; # an array of floating point weights between 0 and 1.
```

Alternately, "tuple typing" may be used to specify the allowable values for an array, in this case a list of schemas that apply to each member of the array in sequence:

```
array {
    integer;
    string;
    number;
} artificial;
```

When tuple typing is used, the * operator may be used to allow additional elements at the end of an array. For instance, to specify an array where the first element is an integer and the remaining are of arbitrary number and type, one might use the following schema:

```
array { integer; }* intFollowedByWhatever;
```

Finally, array types also support range semantics, for min/max number of elements:

```
array { integer; } {0,10} myArrayOfSmallInts;
```

B.3.9 Additional properties in arrays and objects

JSONSchema provides the additionalProperties attribute, which allows a schema author to either:

- specify that a valid instance object/array may not have any properties not in the schema
- specify an additional schema that applies to any additional properties in the instance object or array that are not explicitly mentioned in the schema

Orderly allows you to specify if additional properties SHOULD be allowed, but does not allow you to specify a schema which governs these additional properties. A trailing * in Orderly indicates additional properties are allowed, and occurs immediately after the definition of nested schemas (the closing curly brace) for both objects:

```
object {
    string name;
    string title;
}* employee;
```

And for arrays:

```
array { integer; string; }* myOpenTupleTypedArray
```

B.3.10 Null Types

The null type in JSONSchema specifies a value that MUST be null. The null type specifier is Orderly's equivalent:

```
null likeAir;
```

As explained in the JSONSchema proposal, null is useful "mainly for purpose of being able use union types to define nullability". For example:

```
union {
    string [ "Sr.", "Jr.", "III" ];
    null;
} suffix;
```

B.3.11 Any types

"Any types" are represented in Orderly using the any type specifier:

```
any notes;
```

B.3.12 Unions

It is possible in JSONSchema to specify a property that may be of one of many different types. In Orderly this functionality is represented using the union type specifier:

```
union {
    string;
    number;
} myUnion;
```

A key syntactic feature to note is the supported (required?) omission of property names where they would be meaningless.

B.3.13 Maps

Associative arrays are neither defined in Orderly nor in JSONSchema. The CMIS Browser Binding introduces associative arrays (“maps”) to describe a collection of unique keys and a collection of values.

Maps describe JSON objects without fixing the property names and the number of properties. The keys become JSON object property names and have to be non-null strings. Keys can be restricted, for example, by defining a min and max length, regular expressions, an enum, etc. The values data type can be defined by any unnamed entry including null.

Maps are specified using the map type specifier. Key and value types are defined within curly braces. The key type first, followed by “=>”, followed by the value type:

For example:

```
map { string => boolean } isAllowed;
map { string{2,10} => union { string; integer; null; } } things;
map { string [ "happy", "sad", "meh" ] => integer } intMapping;
```

B.3.14 Extensions or Extra Properties

Orderly is capable of concisely representing a subset of JSONSchema, however at times it might be desirable to be able to represent properties in JSONSchema that are not supported natively in Orderly. For this reason the backtick operators will allow you to encode a JSON object as part of an Orderly schema.

For example to attach a description to a schema entry one might generate something like:

```
string `{"description": "The name of the service"}`;
```

The author has full control over formatting, as whitespace is ignored:

```
string `{
  "title": "Service Name",
  "description": "The name of the service",
  "ui_hints": "Use the blink tag"
}`;
```

B.3.15 ID's

Schema elements can have an id, specified using the property “id”.

For example:

```
object {
    id "http://docs.oasis-open.org/ns/cmis/browser/201103/ACLcapabilities";
    string supportedPermissions [ "basic", "repository", "both" ];
    string propagation [ "repositorydetermined", "objectonly", "propagate" ];
    array { ref "http://docs.oasis-open.org/ns/cmis/browser/201103/permissionDefinition" }
        ↪ permissions;
    ref "http://docs.oasis-open.org/ns/cmis/browser/201103/permissionMapping" mapping?;
 }*;
```

B.3.16 References

The reference type specifier “ref” is used to refer to another Orderly schema element using the “id” described in section B.3.15. For example:

```
object {
    string name;
    string title;
    ref "http://json-schema.org/card" secretary;
    array {
        ref "http://json-schema.org/card";
    } reports;
} employee;
```

B.3.17 Bases

The specifier "base" is used to define the base schema element of the element. All properties are inherited from the base element. For example:

```
object {
    id "http://example.com/person";
    string firstname;
    string lastname;
} person;

object {
    id "http://example.com/employee";
    base "http://example.com/person";
    integer employee_number;
} employee;

// The schema element "http://example.com/employee" consists of
// the properties "firstname", "lastname" and "employee_number".
```

B.3.18 More Complex Examples

A number with a range, enumerated possible values, and a default value:

```
integer{0,256} powerOfTwo[1,2,4,8,16,32,64,128,256] = 1;
```

An object with enumerated possible values and a default.

```
object {
  string beast;
  number normalTemperature;
} temps [ { "beast": "canine", "normalTemperature": 101.2 },
    { "beast": "human", "normalTemperature": 98.6 } ]
  = { "beast": "canine", "normalTemperature": 101.2 };
```

B.3.19 Cautions

When you stare hard enough at the grammar of a non-trivial language you usually learn quite a deal. Sometimes what you learn can be surprising or downright confusing. Here's a tour of the darker parts alleys of Orderly:

Brackets and braces -- visually a tad confusing:

```
integer{7,42} secretOfLife[7,42];
```

And a little bit more confusing:

```
array { integer{7,42}[7,42]; } secretOfLife;
```

B.4 The Normative Grammar

```
Orderly_schema
    unnamed_entry ';'
    unnamed_entry

named_entries
    named_entry ';' named_entries
    named_entry
    # nothing

unnamed_entries
    unnamed_entry ';' unnamed_entries
    unnamed_entry
    # nothing

named_entry
    definition_prefix property_name definition_suffix
    string_prefix property_name string_suffix

unnamed_entry
    definition_prefix definition_suffix
    string_prefix string_suffix

definition_prefix
    'id'
    'integer' optional_range
    'number' optional_range
    'boolean'
    'null'
    'any'
    # a tuple-typed array
    'array'  '{' unnamed_entries '}' optional_additional_marker optional_range
    # a simple-typed array (notice the '*' marker is disallowed)
    'array'  '[' unnamed_entry ']' optional_range
    'object' '{' named_entries '}' optional_additional_marker
    'union'  '{' unnamed_entries '}'
    'map' '{' map_key '=>' unnamed_entries '}' optional_optional_marker
    'ref' ref_string
    'base' ref_string

string_prefix
    'string' optional_range

string_suffix
    optional_perl_regex definition_suffix

definition_suffix
    optional_enum_values optional_default_value optional_requires \
        optional_optional_marker optional_extra_properties
    # nothing

map_key
     string_prefix string_suffix

ref_string
     '"' json_string '"'

csv_property_names
    property_name "," csv_property_names
    property_name

optional_extra_properties
    '`' json_object '`'
    # nothing

optional_requires
    '<' csv_property_names '>'
    # nothing

optional_optional_marker
```

```
    '?'
    # nothing

optional_additional_marker
    '*'
    # nothing

optional_enum_values
    json_array
    # nothing

optional_default_value
    '=' json_value
    # nothing

optional_range
    '{' json_number ',' json_number '}'
    '{' json_number ',' '}'
    '{' ',' json_number '}'
    '{' ',' '}'                         # meaningless, yes.
    # nothing

property_name
    json_string
    [A-Za-z_\-]+                        # (alpha & underbar & dash)

optional_perl_regex # perl compatible regular expressions are supported
    '/' ([^/]|\/) '/' # a Perl 5 compatible regular expression
    #nothing

----------------------------------------
---------- [The JSON Grammar] ----------
----------------------------------------

json_object
    {}
    { members }
members
    pair
    pair , members
pair
    json_string : json_value
json_array
    []
    [ elements ]
elements
    json_value
    json_value , elements
json_value
    json_string
    json_number
    json_object
    json_array
    'true'
    'false'
    'null'

----------------------------------------

json_string
    ""
    " chars "
chars
    char
    char chars
char
    any-Unicode-character-
        except-quote-or-backslash-or-
        control-character
    \" #double quote (")
    \\
    \/
```

```
    \b
    \f
    \n
    \r
    \t
    \u four-hex-digits
json_number
    int
    int frac
    int exp
    int frac exp
int
    digit
    digit1-9 digits
    - digit
    - digit1-9 digits
frac
    . digits
exp
    e digits
digits
    digit
    digit digits
e
    e
    e+
    e-
    E
    E+
    E-
```

Appendix C. Acknowledgements

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

Allart, Philippe	Adullact
Boses, Michael	Quark
Brown, Mr. Jay	IBM
Carlson, Mr Mark	Oracle Corporation
Carr, Mr. Derek	IBM
Caruana, Mr. David	Alfresco Software
Cava, Mr. Bill	Ektron
Chan, Mr. Eric	Oracle Corporation
Charles, Mr. Sameer	Magnolia International AG
Chow, Alexander	Liferay, Inc.
Chow, Derek	Genus Technologies, LLC
Choy, David	Individual
Churchland, Mr. David	Hewlett-Packard
Croisier, Mr. Stephane	RSD
Davis, Cornelia	EMC Corporation
de Kruijff, Mr. Bram	GX Software
Doong, Ms. Jane	IBM
Duerig, Mr. Michael	Adobe Systems
Dufault, Randy	Genus Technologies, LLC
Eberding, Karsten	Individual
Ewing, Mr. Andrew	Hewlett-Packard
Fanning, Betsy	AIIM
Frederiksen, Steffen	Content Technologies ApS
Garroni, Mr. Paolo	ISIS Papyrus America Inc.
Geisert, Mr. Uwe	Open Text Corporation
Goetz, Mr. Paul	SAP AG
Guillaume, Florent	Nuxeo
Haag, Mr. Alexander	WeWebU Software AG
Harmetz, Adam	Microsoft Corporation
Hemmert, Mr. Valentin	WeWebU Software AG
Hermes, Mr. Martin	SAP AG
Hind, Dr. Andrew	Alfresco Software
Hübel, Mr. Jens	SAP AG
Janssen, Mr. Gershon	Individual
Jean, Mr. Raphael	Entropysoft

Joyer, Mr. Neil	Microsoft Corporation
Kadlabalu, Hareesh	FatWire
Klamerus, Mr. Mark	Individual
Klevenz, Mr. Stephan	SAP AG
Kraft, Mr. Boris	Magnolia International AG
Lee, Mr. GI	Zia Consulting, Inc.
Macmillan, Ms. Alison	Oracle Corporation
Malabarba, Mr. Scott	IBM
McVeigh, Mr. Ryan	Zia Consulting, Inc.
Melahn, Mr. Gregory	IBM
Michel, Mr. James	WeWebU Software AG
Miller, Mr. Pat	Microsoft Corporation
Monks, Peter	Alfresco Software
Mooty, Mr. Mathew	Microsoft
Müller, Mr. Florian	SAP AG
Newton, John	Alfresco Software
Nuescheler, David	Adobe Systems
OBrien, Tom	Ektron
Palmer, Dr. Jody	Open Text Corporation
Patel, Mr. Alpesh	Ektron
Pausch, Rainer	WeWebU Software AG
Piegaze, Mr. Peeter	Adobe Systems
Pitfield, Mr. David	Oracle Corporation
Pole, Thomas	Harris Corp
Quinn, Norrie	EMC Corporation
Randall, Craig	Adobe Systems
Rodriguez, Celso	ASG Software Solutions
Roth, Steve	Oracle Corporation
Ryan, Mr. Patrick	IBM
Schnabel, Bryan	Individual
Schreiber, Angela	Adobe Systems
Tazbaz, Paul	Wells Fargo
Ward, Mr. Patrick	Booz Allen Hamilton

Appendix D. Change log

The following changes have been made to CMIS 1.0 Errata 1:

CMIS-580 CONTAINS escaping needs additional clarification

CMIS-587 objectbyid template should support returnVersion parameter

CMIS-653 WSDL needs soapAction value in operation declarations for increased interoperability with .NET

CMIS-654 Missing XML attribute types in CMIS schema

CMIS-655 Folder Children (restrict acceptable values for OrderBy)

CMIS-658 Standardize queryName for properties

CMIS-669 add CMIS Type Mutability for next version of spec

CMIS-673 Add {renditionFilter} to CMIS URI template for query

CMIS-692 applyAcL should allow empty response

CMIS-693 CMIS Repository Extensions

CMIS-707 Add recommendation to use Content-Disposition header with AtomPub setContentStream operation

CMIS-708 Clarify Browser Binding DateTime property values

CMIS-709 In Section 2.2.4.9.2, list of outputs for getObjectByPath appears to be incomplete

CMIS-711 Allow creation of unfiled documents with AtomPub

CMIS-712 Add a description property to all base types

CMIS-713 Add secondary object types

CMIS-714 Proposal to add Retention and Legal Hold Policies for next version of spec

CMIS-715 API for cross site request forgery defense

CMIS-719 Browser Binding

CMIS-720 The keys of the maps returned by a Browser Binding query should be the query names/aliases

CMIS-721 Add queryable cmis:isPrivateWorkingCopy property

CMIS-723 Need to express other cmis:object types to clients.

CMIS-727 Add bulkUpdateProperties operation

CMIS-728 Clarify whether PWC is a latest version or not

CMIS-729 Add new Wildcard section in Query

CMIS-730 Timezone should not be optional in a datetime string in the query BNF (fulltext syntax)

CMIS-732 Clarification limitation of Date value range in browser binding

CMIS-734 Rename "clientToken" to "callback"

CMIS-735 Add capability to append to a content stream

CMIS-736 Remove ETAG recommendation from AtomPub section

CMIS-737 CMIS specification should include a UML diagram for the domain data model

CMIS-738 Add introduction sub-section to 1.1 draft to enumerate new 1.1 features.

CMIS-739 Client hint with iframe as target does not work in all browsers

CMIS-741 New proposal for section 5.2.9.3 "Authentication with Tokens for Browser Clients"

CMIS-743 Add a download type parameter to the Browser Binding getContentStream() service

CMIS-744 Clarify the meaning fo fullTextIndexed and use consistently.

CMIS-745 The type ids for the secondary types defined in 2.1.6 are not valid QNames

CMIS-746 2.1.14.3 Character escape for text search expression should include literal double-quote.

CMIS-747 Metadata inclusion in reponses for properties is verbose and there is no option to control it. Objects do not include any type metadata which is not consistent.