

---

# 1 Browser Binding

## 1.1 Overview

The CMIS Browser Binding is intended to make it simpler for browser-based applications to find, create, update and delete content stored in CMIS repositories. Also this binding is optimized for use in browser applications, it can also be useful as a simpler HTTP based binding in other application models.

## 1.2 Common Service Elements

### 1.2.1 Protocol

HTTP shall be used as the protocol for service requests. HTTP GET shall be used for reading content and HTTP POST shall be used for creating, updating and deleting content.

### 1.2.2 Data Representation

Browser applications are typically written in JavaScript and a popular lightweight data representation format amongst JavaScript developers is JavaScript Object Notation (JSON) as described in RFC 4627 (see <http://www.ietf.org/rfc/rfc4627.txt>). So in this binding JSON shall be used to represent CMIS repositories, folders, documents, relationships and policies.

#### 1.2.2.1 Mapping Schema Elements to JSON

JSON only defines a few types, including Object, String, Number, Boolean, Null, and Arrays. Since not all the types used in the CMIS schema have direct JSON equivalents, some explanation of mapping is necessary.

CMIS	JSON
string	string
boolean	boolean
decimal	number
integer	number
datetime	string (xsd:dateTime format)
uri	string (unencoded url)
id	string
html	string

#### 1.2.3 Referencing Resources by Path and by Id

Resources can be referenced using either the full path or by Id.

When path is used, URI used to address the resource shall be relative to the `rootFolderUri` returned from the `getRepositoryInfo` service.

In the case of Id, the request parameter `id` shall be used.

27

28 Examples

29

30 Referencing an object by path.

31 GET /cmis/repository/123/myFolder/myDocument

32

33 Referencing an object by id.

34 GET /cmis/repository/123?id=0192018282

## 35 1.2.4 Paging

36

37 Since the number of objects returned from the navigation services can be huge, a mechanism for paging  
38 is provided. The optional input parameters defined on the navigation services, `maxItems` and  
39 `skipCount`, shall be represented as HTTP request parameters of the same name.

40

41 A JSON object with the key “`cmis:pageinfo`” shall be present if the request parameter `maxItems` is  
42 present. This object shall have the following JSON key/value pairs.

43

44 **boolean** `hasMoreItems`

45 **number** `numItems`

46

47 See the description in Section 2.2.1.1 for the meaning of these parameters.

48

49 Example:

50

51

```
52 GET /cmis/repository/123/myFolder?maxItems=10 HTTP/1.1
```

```
53   Host: www.example.com
```

```
54   User-Agent: Mozilla/5.0
```

55

```
56   {cmis:pageinfo":
```

```
57     {
```

```
58       "hasMoreItems": true,
```

```
59       "numItems": 127
```

```
60     }
```

```
61   }
```

```
62   . . .
```

63

64

## 65 1.2.5 Multipart Forms

66 Browser applications also typically use HTTP multipart forms as described in RFC 2388 (see  
67 <http://tools.ietf.org/html/rfc2388>) to create and update content. This is especially useful for updating file  
68 content with the addition of the FILE attribute in RFC 1867 (see <http://tools.ietf.org/html/rfc1867>). In this  
69 binding, HTTP POST of multipart/form-data shall be to update content streams, and perform multiple  
70 operations in a single POST.

## 71 1.2.6 Namespaces

72 JSON does not need to define namespaces since the scope of JSON object allows key/value pairs within  
73 it to be uniquely referenced, but when it becomes necessary to uniquely JSON objects because they  
74 may otherwise conflict with repository-defined values, the prefix “`cmis:`” shall be used to identify objects

75 unique to this specification. For example, it is necessary to include the prefix “cmis:” in the JSON key  
76 name for “cmis:name”, for otherwise this could conflict with a repository defined property with a key of  
77 “name” within the JSON object with a key of “properties”.

78  
79 But when not necessary to uniquely define a JSON object, the prefix shall be omitted so conserve space.  
80 For example, “cmis:read” can be expressed as “read” in a JSON array since the array element will  
81 only be referenced in the context of a specific CMIS property. As another example, it is not necessary to  
82 use the prefix of “cmis:” in the JSON object with a key of “properties” since that key is already  
83 unique within its containing object.

84  
85 **QUESTION:** Is it worth the bother to make this distinction or should we just use the “cmis:” prefix  
86 whenever we reference a CMIS schema element?

87 **COMMENT:** We could get more savings by eliminating the prefix by using more JSON nesting, e.g. use a  
88 “cmis:properties” key for a object to hold all the properties defined by the specification and a sibling  
89 object with a key of “properties” to hold all the other properties.

90

## 91 1.2.7 Authentication

92 Authentication SHOULD be handled by the transport protocol.

## 93 1.2.8 Return Codes

94 HTTP Return Codes shall be used to indicate success or failure of an operation. Please see the HTTP  
95 specification for more information on the HTTP status codes. These are provided as guidance from the  
96 HTTP specification. If any conflict arises, the HTTP specification is authoritative.

97	CMIS Services Exception	HTTP Status Code
98	invalidArgument	400
99	objectNotFound	404
100	permissionDenied	403
101	notSupported	405
102	runtime	500
103	constraint	409
104	filterNotValid	400
105	streamNotSupported	403
106	storage	500
107	contentAlreadyExists	409
108	versioning	409
109	updateConflict	409
110	nameConstraintViolation	409

111

## 112 1.3 Services

### 113 1.3.1 Repository Services

#### 114 1.3.1.1 getRepositories, getRepositoryInfo

115 An HTTP GET operation shall be used to obtain a list of CMIS repositories, and information about each.

116 **QUESTION:** do we want to treat these into separate services so a client can get just a simple list of  
117 repositories without all the detailed properties?

### 118 1.3.1.1.1 Inputs

119 **None**

### 120 1.3.1.1.2 Outputs

121 A set of JSON objects representing repositories. Each repository is represented as a JSON object, with  
122 a key of `repositoryId` and key/value pairs representing each of the properties defined in the  
123 `cmisRepositoryInfoType` schema.

### 124 1.3.1.1.3 Exceptions Thrown & Conditions

125 See section "General Exceptions"

### 126 1.3.1.1.4 Example

```
127 GET /cmis/repositories HTTP/1.1
128 Host: www.example.com
129 User-Agent: Mozilla/5.0
130
131 {
132   "cmis:repositories" :
133   {
134     "repository123" :
135     {
136       "repositoryID" : "repository123",
137       "repositoryName", "Repository 123",
138       "repositoryDescription", "This is a CMIS repository",
139       "vendorName" : "Foo Inc.",
140       "productName" : "Product Bar",
141       "productVersion" : "1.0",
142       "rootFolderId" : "123",
143       "rootFolderUri" : "http://www.example.com/cmisis/repository/123",
144       "latestChangeLogToken" : "0123456789",
145       "thinClientURI" : "http://www.example.com/web/index.html",
146       "changesIncomplete" : true,
147       "changesOnType" :
148       [
149         "cmis:document",
150         "cmis:folder"
151       ]
152       "supportedPermissions" : "basic",
153       "propagation" : "objectonly",
154       "permission" : ["read", "write", "all"],
155       "permissionMapping" :
156       [
157         ["canMoveObject.Source", ["write", "read"]],
158         ["canDelete.Object", ["write"]]
159       ]
160       "principalAnonymous" : "anon",
161       "principalAnyone" : "any"
162     }
163   }
164 {
165   "repository456" :
166   {
167     "repositoryID" : "repository456",
168     "repositoryName", "Repository 456",
169     "repositoryDescription", "This is another CMIS repository",
170     "vendorName" : "Bar Inc.",
171     "productName" : "Product Foo",
172     "productVersion" : "1.0",
```

```
173     "rootFolderId" : "456",
174     "rootFolderUri" : "http://www.example.com/cmisis/repository/456",
175     "latestChangeLogToken" : "9876543210",
176     "thinClientURI" : "http://www.example.com/web/index.html",
177     "changesIncomplete" : true,
178     "changesOnType" :
179     [
180         "cmis:document",
181         "cmis:folder"
182     ]
183     "supportedPermissions" : "basic",
184     "propagation" : "objectonly",
185     "permission" : ["read", "write", "all"],
186     "permissionMapping" :
187     [
188         ["canMoveObject.Source", ["write", "read"]],
189         ["canDelete.Object", ["write"]]
190     ]
191     "principalAnonymous" : "anon",
192     "principalAnyone" : "any"
193     }
194 }
195 }
196 }
```

### 197 1.3.1.2 getTypeChildren

### 198 1.3.1.3 getTypeDescendants

### 199 1.3.1.4 getTypeDefinition

## 200 1.3.2 Navigation Services

### 201 1.3.2.1 getChildren

202 An HTTP GET operation shall be used to obtain a list of the child folders and documents of a folder.

#### 203 1.3.2.1.1 Inputs

204 **Required** The resource on which the GET operation is performed shall be specified as an id, or as a  
205 path. See section 1.2.3 for a description of how the id and path shall be specified.

206

207 **Optional** See section 2.2.3.1.1 for a list of the optional parameters for this service. When used, each  
208 parameter shall be specified as a request parameter.

#### 209 1.3.2.1.2 Outputs

210 A set of JSON objects representing folders and documents. Each child resource is represented as a  
211 JSON object, with a key of the value of the "cmis:name" property of the child resource, with key/value  
212 pairs in the object representing the properties of the child resource.

#### 213 1.3.2.1.3 Exceptions Thrown & Conditions

214 See section "General Exceptions"

#### 215 1.3.2.1.4 Example

```
216 GET /cmis/repository/123/myFolder?includePathSegment HTTP/1.1
217 Host: www.example.com
218 User-Agent: Mozilla/5.0
219
220
```

```

221 {
222 "myChildFolder 1" :
223 {
224   "properties" :
225   {
226     "cmis:objectid" : "f1234567890",
227     "cmis:baseTypeId" : "cmis:folder",
228     "cmis:objectTypeId" : "personnelRecords",
229     "cmis:createdBy" : "Gregory Melahn",
230     "cmis:creationDate" : "2010-07-16T03:10:32.088-05:00",
231     "cmis:lastModifiedBy" : "David Neuscheler",
232     "cmis:lastModificationDate" : "2010-07-22T05:18:52.018-05:00",
233     "cmis:changeToken" : `1234567890`,
234     "cmis:parentId" : "f9876543210",
235     "cmis:path" : "/myChildFolder",
236     "cmis:allowedChildObjectIds" :
237     [
238       "Type012", "Type345"
239     ]
240     "cmis:pathSegment" : "myChildFolder"
241   }
242 }
243 {
244 {
245 "myChildDocument" :
246 {
247   "properties" :
248   {
249     "cmis:objectid" : "d1234567890",
250     "cmis:baseTypeId" : "cmis:document",
251     "cmis:objectTypeId" : "personnelRecord",
252     "cmis:createdBy" : "Ryan McVeigh",
253     "cmis:creationDate" : "2010-07-16T03:10:32.088-05:00",
254     "cmis:lastModifiedBy" : "David Neuscheler",
255     "cmis:lastModificationDate" : "2010-07-22T05:18:52.018-05:00",
256     "cmis:changeToken" : `1234567890`,
257     "cmis:parentId" : "f9876543210",
258     "cmis:path" : "/myFolder/MyChildDocument",
259     "cmis:isUmmutable" : false,
260     "cmis:isLatestVersion" : true,
261     "cmis:isMajorVersion" : false,
262     "cmis:isLatestMajorVersion" : false,
263     "cmis:versionLabel" : "1.1",
264     "cmis:versionSeriesId" : "v001",
265     "cmis:isVersionSeriesCheckedOut" : true,
266     "cmis:versionSeriesCheckedOutBy" : "Gregory Melahn",
267     "cmis:versionSeriesCheckedOutId" : "v001abc",
268     "cmis:checkinComment" : "this is a change",
269     "cmis:contentStreamLength" : 1002991,
270     "cmis:contentStreamMimeType" : "application/pdf",
271     "cmis:contentStreamFileName" : "doc001.pdf",
272     "cmis:contentStreamUri" :
273     "http://www.example.com/cm/repository/123/myFolder/doc001.pdf" ,
274   }
275 }
276 }
277 }
278

```

279 **1.3.2.2 getDescendants**

280 **1.3.2.3 getFolderTree**

281 **1.3.2.4 getFolderParent**

282 **1.3.2.5 getObjectParents**

283 **1.3.2.6 getCheckedOutDocs**

284 **1.3.3 Object Services**

285 **1.3.3.1 createDocument**

286 **1.3.3.2 createDocumentFromSource**

287 **1.3.3.3 createFolder**

288 **1.3.3.4 createRelationship**

289 **1.3.3.5 createRelationship**

290 **1.3.3.6 createPolicy**

291 **1.3.3.7 getAllowableActions**

292 **1.3.3.8 getObject**

293 **1.3.3.9 getProperties**

294 **1.3.3.10 getObjectByPath**

295 **1.3.3.11 getContentStream**

296 **1.3.3.12 getRenditions**

297 **1.3.3.13 updateProperties**

298 **1.3.3.14 moveObject**

299 **1.3.3.15 deleteObject**

300 **1.3.3.16 deleteTree**

301 **1.3.3.17 setContentStream**

302 **1.3.3.18 deleteContentStream**

303 **1.3.4 Multi-filing Services**

304 **1.3.4.1 addObjectToFolder**

305 **1.3.4.2 removeObjectFromFolder**

306 **1.3.5 Discovery Services**

307 **1.3.5.1 query**

308 **1.3.5.2 getContentChanges**

309 **1.3.6 Versioning Services**

310 **1.3.6.1 checkOut**

311 **1.3.6.2 cancelCheckOut**

312 **1.3.6.3 checkin**

313 **1.3.6.4 getObjectOfLatestVersion**

314 **1.3.6.5 getPropertiesOfLatestVersion**

315 **1.3.6.6 getAllVersions**

316 **1.3.7 Relationship Services**

317 **1.3.7.1 getObjectRelationships**

318 **1.3.8 Policy Services**

319 **1.3.8.1 applyPolicy**

320 **1.3.8.2 removePolicy**

321 **1.3.8.3 getAppliedPolicies**

322 **1.3.9 ACL Services**

323 **1.3.9.1 getACL**

324 **1.3.9.2 applyACL**

325