Content Management Interoperability Services (CMIS) Proposal

Symbolic Link Version 1.5

11 July 2016

Latest version:

http://docs.oasis-open.org/ns/cmis/extension/symbolicLink

Technical Committee:

OASIS Content Management Interoperability Services (CMIS) TC

Author:

Kenneth Baclawski <Ken@Baclawski.com>
Eric S. Chan <eric.s.chan@oracle.com>

Abstract:

This extension provides for a subclass of cmis:item called cmis:symbolicLink. An object of type cmis:symbolicLink has the property referenceId. The property referenceId specifies an object Id or latest accessible state identifier. When the Id of a symbolic link item is specified in a service request, the referenceId property of the item may be used instead of the specified symbolic link Id. The replacement of the symbolic link Id with another Id is called *dereferencing*. When dereferencing is performed, it is recursive: if the referenceId of a symbolic link item is itself the Id of a symbolic link item, then that symbolic link Id is also dereferenced. If a dereferencing cycle is detected, a cmis:invalidArgument exception is thrown. Every service call has optional parameters dereferenceInputObjectIds and dereferenceDescendantSymbolicLinks. The dereferenceInputObjectIds parameter is a Boolean parameter that specifies whether any objectId parameters are to be dereferenced. The default is true. In other words, if dereferenceInputObjectIds is not specified, then all objectId parameters are dereferenced. Setting dereferenceInputObjectIds to false allows one to set and to update the properties of the symbolic links themselves rather than the properties of the objects referenced by the symbolic links. The dereferenceInputObjectIds parameter only applies to objectId parameters. Any parameters of type Id that are not objectIds are not affected by the value of dereferenceInputObjectIds, and if any such parameters are symbolic links then they are always dereferenced. The dereferenceDescendantSymbolicLinks parameter is a Boolean parameter that specifies whether the symbolic links that occur as descendants in a folder tree should be dereferenced. The default is false.

Motivation

As a rough estimate, the 80% usage case for a CMIS repository is as a filesystem alternative. The POSIX operating-system standard supports symbolic links (also called soft links or symlinks). This includes Unix-like systems, Mac OS X and Windows (in the form of shortcuts). The popularity of symbolic links is a strong motivation for adding support for them in the CMIS standard. The POSIX operating-system standard also supports hard links. The hard link feature is similar to the CMIS `capabilityMultifiling`.

A symbolic link is broken if it does not reference an accessible object. A symbolic link could be broken because of access restrictions for the user rather than because the object does not exist. Since the access restrictions for the symbolic link could differ from those for the referenced object, it is not feasible to require that a symbolic link be automatically deleted when the referenced object is deleted.

The following use cases give some examples of the capabilities that symbolic links would support. Symbolic links offer some capabilities that are similar to those provided by multi-filing. The first two use cases below are such capabilities. However, there are also capabilities that cannot be simulated by multi-filing. The last five use cases below are examples of such capabilities.

Use Case 1. Store references to CMIS managed documents in multiple folders when multi-filing is not supported.

When a repository does not have the `CapabilityMultifiling`, an object can only be filed in a single folder. A symbolic link allows one to create and to manage references to objects in multiple folders.

Use Case 2. Provide additional names for the same document.

A symbolic link can be used when it is necessary or convenient to rename an object (or more precisely, to change the Id) yet still support the original name. This differs from multi-filing which provides a different access path for a document but does not provide a different objectId.

Use Case 3. Support the ability to file a latest accessible state identifier.

Since a latest accessible state identifier is not an object Id, it cannot be filed. Symbolic links allow such identifiers to be filed in folders that allow symbolic link items.

Use Case 4. Provide for multi-filing of file folders.

Since only documents can be filed in multiple folders, symbolic links provide the ability to file folders in multiple folders.

Use Case 5. Manage different references to the same object.

Since a symbolic link is a separate object, it can have its own secondary types and access control that are independent of the referenced object.

Use Case 6. Provide additional identifiers for objects other than documents and folders.

Use Case 7. Allow objects other than documents and folders to be filed in folders that do not allow such objects.

Symbolic links allow one to file (in folders that allow symbolic link items) references to objects that would not otherwise be fileable.

Specification

A new class is introduced called cmis:symbolicLink as a subclass of cmis:Item. An object in this class has a reference to another object or to a latest accessible state of a document. The reference is specified by the referenceId property. When the Id of a symbolic link object is specified in a service request, the referenceId property of the symbolic link object may be used instead of the specified symbolic link Id. The replacement of the symbolic link Id with an object id is called *dereferencing*. When dereferencing is performed, it is recursive: if the referenceId of a symbolic link object is itself the Id of a symbolic link object, then that symbolic link Id is also dereferenced. If the result of dereferencing is a latest accessible state identifier, then the latest accessible state identifier is replaced by a document Id as specified in the latest accessible state extension. The dereferencing operation can fail for a number of reasons:

1. The result of dereferencing does not represent the Id of an object. In this case, a cmis:objectNotFound exception is thrown.
2. The user does not have permission to access one of the objects encountered during the dereferencing operation. In this case, a cmis:permissionDenied or cmis:objectNotFound exception is thrown depending on the exception that is thrown by the attempt to access the object.
3. A dereferencing cycle is detected. In this case, a cmis:invalidArgument exception is thrown.
4. The dereferencing operation is problematic. If the repositoryId parameter of the service call is a symbolic link, then dereferencing the symbolic link requires access to the symbolic link object stored in the repository. This is called a *vicious circle*. In this case, a cmis:invalidArgument exception is thrown.

Dereferencing of symbolic links that are objectId parameters in a service call can be explicitly disabled by specifying the optional dereferenceInputObjectIds parameter to be false. The dereferenceInputObjectIds parameter only applies to objectId parameters. Any parameters of type Id that are not objectIds are unaffected by the value of dereferenceInputObjectIds, and if any of such parameters are symbolic links then they are always dereferenced.

While *input* object ids of symbolic links MUST be dereferenced by default, *output* symbolic link objects, by default, MUST NOT be dereferenced. The only output objects that may be dereferenced are descendants in a folder graph. This occurs when dereferenceDescendantSymbolicLinks is set to true.

Symbolic Link Object Type Definition

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

Attribute Values

The symbolic link object-type MUST have the following attribute values:

id

Value: cmis:symbolicLink

localName

Value: <repository-specific>

localNamespace

Value: <repository-specific>

queryName

Value: cmis:symbolicLink

displayName

Value: <repository-specific>

baseId

Value: cmis:item

parentId

Value: MUST be set to cmis:Item

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>

Property Definitions

The object-type inherits the property definition of cmis:item. In addition, it MUST have the following property definitions:

cmis:reference	Either the Id of an object or a latest accessible state identifier.
Property Type:	String
Inherited:	FALSE
Required:	TRUE
Cardinality:	single
Updatability:	oncreate
Choices:	Not Applicable
Open Choice:	Not Applicable
Queryable:	<repository specific>
Orderable:	<repository specific>

The rationale for the property type being String is to allow the reference to be a latest accessible state identifier. A CMIS repository MAY restrict the syntax of cmis:referenceId to conform with the syntax for an Id or a latest accessible state identifier.

Service Definitions

This section describes the changes to the service definitions.

Symbolic Link Dereferencing

All methods MAY specify the following input parameters:

Input Parameter:

> **Boolean `dereferenceInputObjectIds`** (optional) This specifies whether the symbolic link Ids that occur as or within the objectId input parameters of the method will be dereferenced. If this parameter is either not specified or set to true then the method MUST dereference all symbolic links that occur as or within the input parameters of the method. If this parameter is set to false then the method MUST NOT dereference any symbolic links that occur as or within the objectId input parameters of the method. Only objectId input parameters are affected by **`dereferenceInputObjectIds`**. The symbolic link Ids that occur as or within any input parameters of the method that are not objectId parameters MUST be dereferenced regardless of the value of **`dereferenceInputObjectIds`**.
>
> **Boolean `dereferenceDescendantSymbolicLinks`** (optional) This specifies whether the symbolic links that occur as descendants in a folder tree should be dereferenced. The parameter only applies to getChildren, getDescendants, getTree, deleteTree and getObjectByPath. If this parameter is unspecified or set to false then the method MUST NOT dereference any symbolic links that occur as descendants in a folder tree during processing of the method. If this parameter is set to true then the method MUST dereference all symbolic links that occur as descendants in a folder tree during processing of the method.

Updates to service definitions.

The createItem service description should add: “Creation of a symbolic link object does not check that the referenced object exists or is accessible.”

The deleteObject service description should add: “Deletion of an object does not also delete any symbolic link objects that may reference the deleted object.”

The following services have a single objectId parameter. If the service is called with a symbolic link as the objectId parameter and if **`dereferenceInputObjectIds`** is either unspecified or set to true, then the symbolic link parameter MUST be dereferenced. If **`dereferenceInputObjectIds`** is set to false, then the objectId parameter MUST NOT be dereferenced. Any parameters of type Id that are not the objectId parameter and that are or contain symbolic links MUST be dereferenced regardless of the value of **`dereferenceInputObjectIds`**.

getObjectParents

getAllowableActions

getObject

getProperties

updateProperties

moveObject

deleteObject

addObjectToFolder

removeObjectFromFolder

getObjectRelationships

applyPolicy

removePolicy

getAppliedPolicies

applyACL

getACL

The bulkUpdateProperties service has a parameter objectIdAndChangeToken that is an array of pairs of type <Id,String>. Some Ids that occur in this parameter may be symbolic links. If **`dereferenceInputObjectIds`** is either unspecified or set to true, then all symbolic links parameters occurring in objectIdAndChangeToken MUST be dereferenced. If **`dereferenceInputObjectIds`** is set to false, then all symbolic links occurring in objectIdAndChangeToken MUST NOT be dereferenced. Any parameters of type Id that are not contained in objectIdAndChangeToken and that are symbolic links MUST be dereferenced regardless of the value of **`dereferenceInputObjectIds`**.

Each of the following services has a single objectId parameter. If one of these services is called with a symbolic link as the objectId parameter and if **`dereferenceInputObjectIds`** is set to false, then the cmis:constraintException MUST be thrown since a symbolic link is a cmis:item and a cmis:item does not have a content stream and cannot be checked out.

getContentStream

getRenditions

setContentStream

appendContentStream

deleteContentStream

checkOut

cancelCheckOut

checkIn

All other services do not have any objectId parameters. If a symbolic link occurs as or within a parameter for any of the other services, then **`dereferenceInputObjectIds`** is ignored, and all symbolic links MUST be dereferenced.

The following services may output one or more objectIds that are normally the same as input parameters. If an input parameter was a symbolic link that was dereferenced, then the output objectId will be the dereferenced objectId not the symbolic link Id.

updateProperties

bulkUpdateProperties

moveObject

setContentStream

appendContentStream

deleteContentStream

The following services are affected by the **dereferenceDescendantSymbolicLinks** parameter. If this parameter is unspecified or set to false then the method MUST NOT dereference any symbolic links that occur as descendants of a folder. If this parameter is set to true then the method MUST dereference all symbolic links that occur as descendants of a folder. If a dereferenced symbolic link is a folder object, then the folder is treated as a child folder.

getChildren

getDescendants

getFolderTree

deleteTree

getObjectByPath

Sections 2.2.3.2 getDescendants and 2.2.3.3 getFolderTree have an additional Note:

If a symbolic link references an ancestor folder in the folder tree, then dereferencing the symbolic links results in an infinite folder tree. To prevent such an occurrence, the **depth** parameter cannot be set to -1 when the **dereferenceDescendantSymbolicLinks** parameter is set to true.

Sections 2.2.3.2.3 and 2.2.3.3.3 Exceptions Thrown & Conditions are updated with the following additional exceptions:

cmis:invalidArgument If the service is invoked with "depth = -1" and "dereferenceDescendantSymbolicLinks = true".

cmis:objectNotFound If an exception is thrown when one or more symbolic links are dereferenced. The exceptions that are thrown during dereferencing operations SHOULD be nested within the cmis:objectNotFound exception that is thrown by the service.

The following figures illustrate the effect of the **dereferenceDescendantSymbolicLinks** parameter for getChildren, getDescendants, getFolderTree, deleteTree and getObjectByPath.

Figure 1: A Folder Graph with no Dereferencing

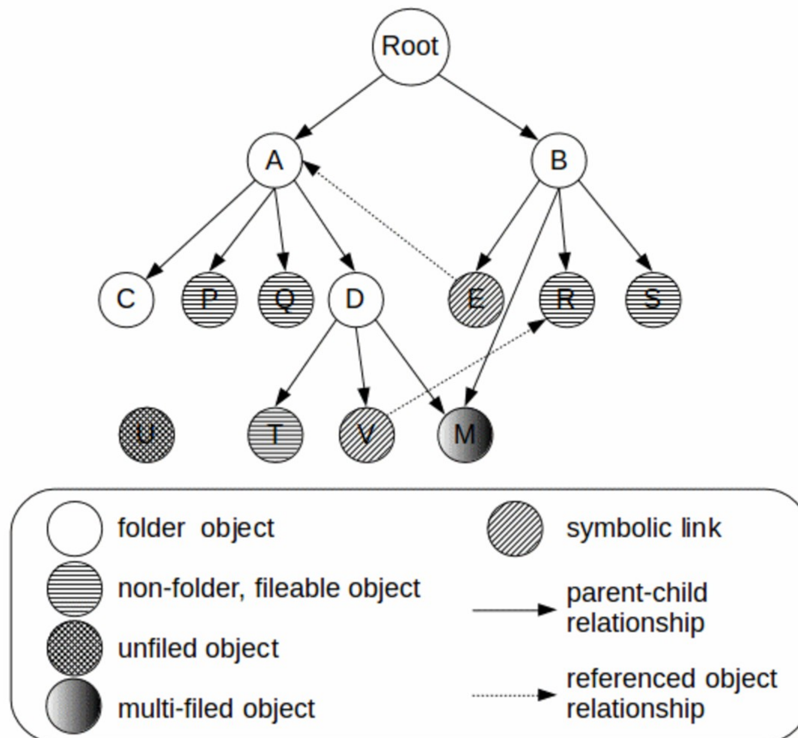
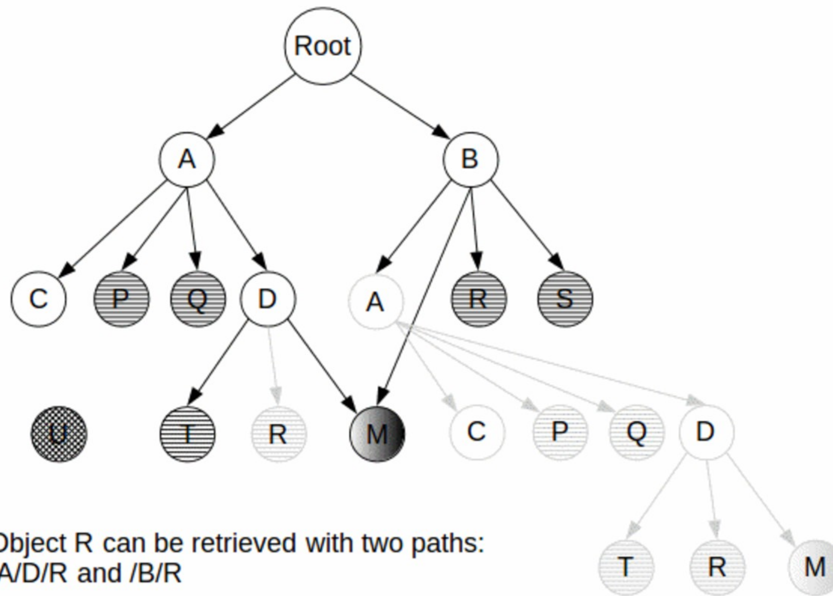


Figure 1 shows a folder graph that includes two symbolic links. This is the graph when the **dereferenceDescendantSymbolicLinks** parameter is unspecified or set to false.

Figure 2 shows what the folder graph appears to be when the **dereferenceDescendantSymbolicLinks** parameter is set to true. Those nodes in the graph that differ between the two graphs are shown in gray. Note that getObjectParents is not affected by the value of the **dereferenceDescendantSymbolicLinks** parameter. In other words, while dereferencing allows retrieval of objects using additional paths, it does not change the parents of the objects. Compared with the situation in Figure 1, the graph in Figure 2 has an additional path for retrieving object R: /B/R. However, object R still only has one parent: B. Similarly, object M now has an additional retrieval path: /B/A/D/M.

Figure2: A Folder Graph after Dereferencing



Object R can be retrieved with two paths:
/A/D/R and /B/R

Object M can be retrieved with three paths:
/A/D/M, /B/M and /B/A/D/M

Issues

A number of issues were raised by Florian Müller concerning symbolic links. The following are the issues and how they were addressed.

Issue 1: Let's say a symbolic link is filed in a folder. The referenced object doesn't exist or the user has no permissions to see the references object. When a client calls getChildren on the folder, what is returned? The symbolic link object or nothing? Both are problematic.

Response:

By default, no dereferencing occurs, so only the symbolic links are shown. This is also the default Unix behavior.

As in Unix, there is no requirement that the referenced object of a symbolic link either exist or be accessible. There is no requirement for the repository to check this when the symbolic link is created, and deletion of the referenced object has no effect on any symbolic links.

If dereferenceDescendantSymbolicLinks is set to true, then the dereferencing operations can throw exceptions. If any dereferencing operation throws an exception, then the service request throws an exception. If exceptions can nest exceptions (as in Java), then the exceptions that were thrown during dereferencing should be nested, but this is not required. Presumably, the same issue would come up during bulk processing.

Issue 2: If the symbolic link object and the referenced object are in the same folder and the client calls getChildren, does the repository return the object twice in the response? (If so, that might break some clients. If not, it is inconsistent.)

Response:

By default, no dereferencing occurs, so only the symbolic links are shown.

If dereferenceDescendantSymbolicLinks is set to true, then objects can appear twice. There is an example of this in the proposal. Since dereferenceDescendantSymbolicLinks is false by default, it is the client's responsibility to handle this situation if this parameter is being explicitly set to true.

Issue 3: How can a client identify if an object ID belongs to a symbolic link or to another object? That's necessary to decide of an updateProperties call with the disableDereferencing makes sense.

Response:

If dereferenceInputObjectIds is set to false, then no dereferencing is done to the objectId parameters. Other Id parameters that are symbolic links will always be dereferenced, but there isn't a compelling reason to have the ability to inhibit dereferencing for those parameters.

Issue 4: What should happen if getDescendants or getFolderTree run into a cycle? If this throws an exception, then a single user can easily break a lot (intentionally or unintentionally).

Response:

This is explicitly handled in the proposal. The service request would throw an exception if depth=-1 and dereferenceDescendantSymbolicLinks is set to true. This could only happen intentionally since the default for the latter is false.

Other kinds of exception could also occur during dereferencing when getDescendants or getFolderTree are requested. These would cause an exception to be thrown by getDescendants or getFolderTree. (Note: The proposal could be changed so that the service must omit symbolic links that cannot be dereferenced and no exception would be thrown in such a case.) As multiple exceptions could be thrown, it would be useful to nest them, but this is not required.

Extension Definition

CMIS 1.2

The extension is supported if there is a cmis:symbolicLink subclass of cmis:Item.

The following is the feature extension entry in the repository info:

ID	http://docs.oasis-open.org/ns/cmis/extension/symbolicLink/1.5
URL	http://docs.oasis-open.org/ns/cmis/extension/symbolicLink
Common Name	Symbolic Link
Version Label	5
Description	This extension provides for an symbolic link object that references another object. The reference may be recursive. A symbolic link may be fileable and can be managed independently of the object that it references.