S3 and CDMI™

A CDMI Guide for S3 Programmers

Version 1.0

SNIA Technical Proposal

May 22, 2013

Revision History

Revision	Date	Sections	Originator	Comments
1.0	5/22/2013	All	Alan G. Yoder, Ph.D Futurewei Technologies, Inc.	Original draft

Contents

1 Overview 6

2 Comparison Summary 7

3 Capabilities 12

4 Operational Details 16

4.1 Authentication 16

4.2 Return Codes 16

4.3 Operations on Containers 18

4.3.1 DELETE Bucket 18

4.3.2 DELETE Bucket lifecycle 19

4.3.3 DELETE Bucket policy 20

4.3.4 DELETE Bucket website 21

4.3.5 GET Bucket acl 21

4.3.6 GET Bucket lifecycle 24

4.3.7 GET Bucket policy 25

4.3.8 GET Bucket location 25

4.3.9 GET Bucket logging 26

4.3.10 GET Bucket notification 28

4.3.11 GET Bucket Object versions 28

4.3.12 GET Bucket requestPayment 28

4.3.13 GET Bucket versioning 28

4.3.14 GET Bucket website 28

4.3.15 HEAD Bucket 29

4.3.16 List Multipart Uploads 30

4.3.17 PUT Bucket 30

4.3.18 PUT Bucket ACL 31

4.3.19 PUT Bucket lifecycle 32

4.3.20 PUT Bucket policy 33

4.3.21 PUT Bucket logging 34

4.3.22 PUT Bucket notification 36

4.3.23 PUT Bucket requestPayment 36

4.3.24 PUT Bucket versioning 36

4.3.25 PUT Bucket website 36

4.4 Operations on Objects 37

4.4.1 DELETE Object 37

4.4.2 Delete Multiple Objects 38

4.4.3 GET Object 38

4.4.4 GET Object ACL 39

4.4.5 GET Object torrent 40

4.4.6 HEAD Object 40

4.4.7 POST Object 41
4.4.8 PUT Object 42
4.4.9 PUT Object acl 43
4.4.10 PUT Object - Copy 44
4.4.11 Initiate Multipart Upload 46
4.4.12 Upload Part 46
4.4.13 Upload Part - Copy 47
4.4.14 Complete Multipart Upload 47
4.4.15 Abort Multipart Upload 47
4.4.16 List Parts 47
5 Acknowledgements 48

1 Overview

This whitepaper is intended for application developers who are using cloud storage. It documents how to use CDMI to obtain functionality similar to Amazon's S3[1] cloud storage interface. It may also serve as a guide to implementors of systems that support CDMI who wish to support S3-like functionality.

The S3 interface is part of Amazon Web Services (AWS). It provides the interface to cloud storage. It is a proprietary interface, which limits the ability of some people to use it. Developers who wish to write to a standards-based interface may want to use a CDMI interface to accomplish the same ends. This paper discusses the similarities and differences between the two interfaces and serves as an initial guide for implementation. It is not, however, intended to be a detailed specification. In fact, it isn't a specification at all, so we have avoided normative keywords like *shall* and *required*.

[1] Trade and service marks used herein are the property of their respective owners

2 Comparison Summary

Table 1 provides an overview of the similarities and differences between the S3 and CDMI interfaces. Each function that S3 supports is discussed in further detail in following sections. For functionality that S3 does not support, please see the CDMI specification available at www.snia.org. The current version, as of early 2013, is 1.0.2.

Table 1. Operational Summary

Function	S3 Support	CDMI Support
List the objects in a container	GET Bucket	GET Container
Display simplified access controls	GET Bucket acl	GET Container metadata item named `cdmi_acl`
Display Windows and NFS compatible access controls		GET Container metadata item named `cdmi_acl`
Discover retention autodelete interval for all objects in a mutable container	GET Bucket lifecycle	GET Container metadata item named `s3_lifecycle`
Discover whether a container shall be deleted at the end of its retention period		GET Container `cdmi_retention_autodelete` metadata item
Discover retention data on a container		GET Container `cdmi_retention_period` and `cdmi_retention_start_time` metadata items
Find legal holds that have been placed on a container		GET Container `cdmi_hold_id` metadata item
Discover the policy set on a container	GET Bucket policy	
Discover the geographic location(s) in which a container's data is stored	GET Bucket location (US and EU only)	GET Container `cdmi_geographic_placement_provided` metadata item (Full ISO 3166 support)
Get logging status	GET Bucket logging	GET logging queue metadata
Get status for full featured logging		GET logging queue metadata (CDMI logging features much richer functionality than S3)
Get status for lost data notifications	GET Bucket notification	
Get status for full featured notifications		GET notification queue metadata (CDMI allows arbitrary notifications based on "scopes", which are similar to S3 bucket policies.)

Function	S3 Support	CDMI Support
Get information on versions of all objects in a container	GET Bucket Object versions	An extension for versioning is available to working group members.
Discover the entity with fiduciary responsibility for a container	GET Bucket requestPayment	
Get information on what a bucket is doing vis-a-vis versioning	GET Bucket versioning	An extension for versioning is available to working group members.
Discover the URL of a website that represents a container	GET Bucket website	GET user metadata you have set to indicate the address you have put in place using DDNS or use Apache-style virtual hosting.
Get container metadata	HEAD Bucket	GET Container `metadata`
Identify received multi-part uploads	List Multipart Uploads	
Create or update top-level container	PUT Bucket	PUT Container
Create or update child containers		PUT Container
Place simple access controls	PUT Bucket acl	PUT Container?`metadata:cdmi_acl`
Place Windows and NFS-compatible access controls		PUT Container?`metadata:cdmi_acl`
Set autodelete period on writeable data	PUT Bucket lifecycle	Set a container metadata item `s3_lifecycle`. Emplace a script to run a daily check on the container.
Set autodelete on retention data		Set `cdmi_retention_autodelete` metadata item to "true"
Manage compliance retention intervals		Manage `cdmi_retention_period` and `cdmi_retention_start_time`
Place legal holds on a container		PUT Container?`cdmi_hold_id`
Manage policies on containers	PUT Bucket policy	
Manage simple logging parameters	PUT Bucket logging	PUT to a logging queue
Specify scope of full-featured logging operations		PUT to a logging queue
Turn on notifications for lost data	PUT Bucket notification	

Function	S3 Support	CDMI Support
Specify scope of full-featured notification queues		PUT to a notification queue
Specify the fiduciary authority for a container	PUT Bucket requestPayment	
Manage object versioning	PUT Bucket versioning	An extension for versioning is available to working group members.
Manage the web address of a container	PUT Bucket website	Use DDNS and set user metadata to record the address.
Delete an empty container	DELETE Bucket	DELETE Container
Delete a container and all its contents		DELETE Container
Discontinue setting an autodelete interval on objects in a container	DELETE Bucket lifecycle	DELETE the Container metadata item called `s3_lifecycle`
Discontinue using a bucket policy	DELETE Bucket policy	
Remove web address that contains a referral to a container	DELETE Bucket website	DELETE the reference
Delete an object	DELETE Object	DELETE Object
Delete multiple objects	Delete objects given in an XML list up to 1000 objects long	Deleting a container deletes all objects in the container. A jobs extension allows bulk deletes as well.
Get an object's contents	GET Object	GET Object (non-CDMI) or GET Object's value field (CDMI)
Get an object's user-defined metadata		GET Object's metadata
Get an object's ACL	GET Object ACL	GET Object ACL
Get object's ACL inheritance settings		GET Object ACL
Get object permissions regarding metadata		GET Object ACL
Download a file using BitTorrent	GET Object torrent	
Get object metadata	HEAD Object	GET Object `metadata`
Upload an object using a web form	POST Object	
Create or modify an object	PUT Object	PUT Object
Create an object, getting back a unique identifier in a flat namespace		POST Object

Function	S3 Support	CDMI Support
Set an object's ACL	PUT Object acl	PUT Object `metadata:acl` (includes full set of permissions, principals, and flags as used in NTFS and NFSv4)
Copy an object already in the cloud to a new location	PUT Object - Copy	PUT Object setting `"copy"` in message body JSON
Move an object already in the cloud to a new location		PUT Object setting `"move"` in message body JSON
Begin uploading an object in multiple pieces	POST Object, specifying an upload ID	PUT Object with X-CDMI-Partial header = `"true"`
Upload part of an object	PUT Object with partnumber and upload ID	PUT Object with Content-Range header set and X-CDMI-Partial header = `"true"`
Copy part of one object to another object	PUT Object with XML describing source	Specified in a future extension to CDMI (“Copy Range”)
Complete updating an object by parts	POST Object with XML containing number of parts	PUT Object with X-CDMI-Partial header = `"false"`
Abort a Multipart Upload	DELETE Object with upload id	DELETE Object
List parts of an object received so far by the server	GET Object with upload id and additional metadata	Client is responsible for tracking successful returns.
Create a FIFO queue that can contain arbitrary types of objects		PUT Queue
Dequeue an object from a queue		GET Queue or DELETE Queue `values:<n>`
Create an identity management namespace		PUT Domain
Add users and groups to a management namespace		PUT to the domain `cdmi_domain_members` field
Query a management namespace for user and group info		GET to the domain `cdmi_domain_members` field
Export a container via NFS		Add an `"exports"` data structure to the container metadata
Export a container via CIFS/SMB		Add an `"exports"` data structure to the container metadata
Export a container via OCCI, iSCSI or WebDAV		Add an `"exports"` data structure to the container metadata

Function	S3 Support	CDMI Support
Create a snapshot of a container		PUT Container, with `"snapshot"` request metadata set to the name of the target snapshot
Export a container in serialized format		Convert an object into JSON format with all binary data encoded with escaped JSON and base64 encodings and place children arrays last
Import a serialized container		Reverse the process mentioned above
Log security events		PUT to a logging queue with a `"cdmi_logging_class"` of `"cdmi_security_logging"`
Log metadata change events		PUT to a logging queue with a `"cdmi_logging_class"` of `"cdmi_datasystem_logging"`
Log object accesses and changes		PUT to a logging queue with a `"cdmi_logging_class"` of `"cdmi_object_logging"`
Query a container for objects matching a specified set of conditions		PUT to a query queue with a scope specification

3 Capabilities

CDMI uses the notion of *capabilities objects* to encapsulate the various behaviors and capabilities of a given class of storage. For example, a Platinum storage class might offer retention and autodeletion, while a Bronze one does not. A ReadOnly storage class will probably not support any sort of object or container deletion or modification, and so on.

A CDMI implementation that is capable of S3-like behavior must implement the following CDMI capabilities for the objects and containers in question (see Table 2). It may, of course, implement other functionality; however, discussion in this paper is limited to the functionality necessary to support S3-like activities.

In general, clients should check the relevant capabilities object to make sure that the capability for a given operation is both existent and "true" before performing an operation. As this adds a round trip to an operation if done every time a new location is accessed, an alternative, especially for container and object capabilities, may involve waiting to inspect a capability until an operational error is encountered.

N.B. "Support" for an operation, such as deleting an object, should not be confused with "permission" to perform the operation. Support is indicated by capabilities that are present and set to `"true",` while permission is controlled by Access Control Lists (ACLs).

Table 2. System-Wide Capabilities (see CDMI 12.1.1)

Capability	Description
cdmi_dataobjects	Indicates support for the CDMI data path
cdmi_security_access_control	Indicates support for ACLs
cdmi_serialization_json	Indicates support for JSON as a serialization format. This is the only supported serialization.
cdmi_query	Indicates support for query queues
cdmi_query_regex	Indicates support for query with regular expressions
cdmi_query_contains	Indicates support for query with `“contains”` expressions
cdmi_query_tags	Indicates support for query with tag-matching expressions
cdmi_query_value	Indicates support for query of value fields
cdmi_notification	Indicates support for notification queues
cdmi_logging	Indicates support for logging queues
cdmi_object_move_from_local	(Not required for S3) Indicates support for moving CDMI objects from URIs within the same storage system
cdmi_object_copy_from_local	Indicates support for copying CDMI objects from URIs within the same storage system

Capability	Description
cdmi_object_copy_from_remote	(Not required for S3) Indicates support for copying CDMI objects from URIs within other CDMI storage systems
cdmi_security_access_control	Indicates support for ACLs for the purpose of access control
cdmi_references	Indicates support for references that return 302 Found (HTTP redirects)

Storage system metadata capabilities are found in the capabilities objects for domains, data objects, containers, and queues, as described in Table 3.

Table 3. Storage System Metadata (see CDMI 12.1.2)

Capability	Description
cdmi_acl	Indicates support for ACLs
cdmi_size	Indicates that the cloud storage system generates a `"cdmi_size"` metadata entry for each stored object
cdmi_ctime	Indicates that the cloud storage system generates a `"cdmi_ctime"` metadata entry for each stored object
cdmi_atime	Indicates that the cloud storage system generates a `"cdmi_atime"` metadata entry for each stored object
cdmi_mtime	Indicates that the cloud storage system generates a `"cdmi_mtime"` metadata entry for each stored object
cdmi_acount	Indicates that the cloud storage system generates a `"cdmi_acount"` metadata entry for each stored object
cdmi_mcount	Indicates that the cloud storage system generates a "cdmi_mcount" metadata entry for each stored object

Capabilities for data system metadata are found in the capabilities objects for data objects, domains, containers, and queues, i.e., in the same places as storage system metadata (see Table 4). The distinction between storage system metadata and data system metadata is therefore largely abstract.

Table 4. Capabilities for Data System Metadata (see CDMI 12.1.3)

Capability	Description
cdmi_data_autodelete cdmi_retention_autodelete	Indicates that the cloud storage system implements auto deletion after the retention period expires. (Note: Some versions of the spec erroneously call out both names in separate places. Check both names, and if either one exists and is set to `"true"`, the capability is available.)
cdmi_data_retention	Required to support cdmi_data_autodelete
cdmi_data_dispersion	Required to support geographic separation of data copies

Capability	Description
cdmi_geographic_placement	Required to support regional boundaries on data placement

Container capabilities are found in the capabilties objects on containers, as described in Table 5.

Table 5. Container Capabilities (see CDMI 12.1.5)

Capability	Description
cdmi_create_container (required on the top-level container only).	Indicates that the cloud storage system supports creating containers in this container
cdmi_delete_container	Indicates that the container can be deleted
cdmi_create_queue	Indicates that creating queues in the container is supported
cdmi_copy_queue	(Not required for S3) Indicates that a queue may be created in the container that is a copy of another queue
cdmi_move_queue	(Not required for S3) Indicates that a queue may be moved from another location into this container
cdmi_read_metadata	Indicates support for reading container metadata
cdmi_modify_metadata	Indicates support for creating and modifying container metadata
cdmi_list_children	Indicates support for listing children of an existing container
cdmi_list_children_range	(Not required for S3) Indicates support for partial listing of children of an existing container, as specified by a range description
cdmi_create_dataobject	Indicates that objects may be created in this container. One could reasonably ask how useful a container *without* this capability could be. Rather than checking for it up front, we recommend checking after an error is returned on PUT.
cdmi_post_dataobject	Indicates that ordinary HTTP POST requests are supported. In CDMI implementations based on FOSS web servers such as Apache, this should work as expected.
cdmi_create_container	(Not required for S3) Indicates support for nested containers
cdmi_delete_container	Indicates the container may be deleted
cdmi_create_reference	Indicates that references can be created in this container
cdmi_copy_dataobject	Indicates that the container supports creation of new objects by copying them from elsewhere in the system

Capability	Description
cdmi_move_dataobject	(Not required for S3) Indicates that the container supports moving objects from elsewhere into the container

Table 6 describes capabilities for data objects.

Table 6. Data Object Capabilities (see CDMI 12.1.5)

Capability	Description
cdmi_read_metadata	Indicates support for reading the object's metadata
cdmi_read_value	Indicates support for reading the object's value
cdmi_read_value_range	Indicates support for reading value ranges, e.g., `myobject?value:256-511` reads bytes 256 to 511
cdmi_modify_metadata	Indicates support for modifying the object's metadata
cdmi_modify_value	Indicates support for modifying the object's value
cdmi_delete_dataobject	Indicates support for deleting the data object

Table 7 describes capabilities for queue objects.

Table 7. Queue Object Capabilities (see CDMI 12.1.5)

Capability	Description
cdmi_read_metadata	Indicates support for reading the queue’s metadata
cdmi_read_value	Indicates support for reading the queue’s value
cdmi_modify_metadata	Indicates support for writing to the queue’s metadata
cdmi_delete_queue	Indicates support for deleting the queue
cdmi_modify_value	Indicates support for enqueuing objects to the queue

4 Operational Details

This section provides details on using CDMI to implement functionality similar to each of the S3 operations listed in section 3 Capabilities.

4.1 Authentication

CDMI 1.0.2 does not require any authentication beyond basic HTTP authentication. A ”curl” command like this will work on servers that support it. (Alternatively, ”jdoe:password” may be passed in using an Authorization header field (see RFC 2616 section 14.8).

```
% curl http://jdoe:password@10.100.3.5 GET /cs/classes/cs101/temp/ ...
```

But sending passwords in the clear is quite unsafe and should be avoided. CDMI servers are required to support TLS 1.0 at a minimum, and may support TLS 1.1 and 1.2. Therefore, the following is much preferable:

```
% curl https://jdoe:password@10.100.3.5 GET /cs/classes/cs101/temp/ ...
```

At present, the CDMI spec does not specify support for Kerberos, PKI, or other third-party or single signon technologies. Individual implementations may offer one or more of them; consult the documentation for the CDMI server you are using to determine whether these security features are available. You may, for example, be able to use an Authentication header field to pass in credentials that apply to these systems.

4.2 Return Codes

CDMI uses the return codes in Table 8 in one or more of the operations used in this paper. Subsequent listings of return codes from operations will omit the details shown here.

Table 8. Return Codes from HTTP Operations

HTTP return codes	Response headers and body	Meaning
202 Accepted	<none>	The object is being created. Monitor the percentComplete and completionStatus fields to find out when to requery.
	X-CDMI-Specification-Version	The highest version supported by both client and server
	Content-Type	`application/cdmi-<type>`

HTTP return codes	Response headers and body	Meaning
204 No Content	<none>	The container or object was successfully created, updated, or deleted.
	X-CDMI-Specification-Version	The highest version supported by both client and server
	Content-Type	`application/cdmi-<type>`
302 Found	Possibly a note containing a URL. A "Location" header may also be present, and if so, it contains the URL to redirect to without caching the new location.	The URI given in the request is a reference to another object, but there is no guarantee that the object will stay there.
400 Bad Request	<none, or an error string for display>	The request contains invalid parameters or field names.
401 Unauthorized	<none, or an error string for display> A WWW-Authenticate header field (RFC 2616 section 14.47) containing a challenge applicable to the requested container will be present.	The authentication credentials are missing or invalid.
403 Forbidden	<none, or an error string for display> RFC 2616 says the server SHOULD indicate why the request failed. It also says that clients (meaning you) SHOULD NOT repeat the request.	The client lacks the proper authorization to perform this request.
404 Not Found	<none, or an error string for display>	The resource was not found at the specified URI.
406 Not Acceptable	<none, or an error string for display>	The server is unable to provide the object in the content type specified in the Accept header.

HTTP return codes	Response headers and body	Meaning
409 Conflict	<none, or an error string for display> From RFC 2616: The request could not be completed due to a conflict with the current state of the resource. This code is only allowed in situations where it is expected that the user might be able to resolve the conflict and resubmit the request. The response body SHOULD include enough information for the user to recognize the source of the conflict. Ideally, the response ... would include enough information for the user or user agent to fix the problem; however, that might not be possible and is not required.	The container or object may not be deleted.
410 Gone	<none, or an error string for display> This status code is recommended by RFC 2616 if the server knows that a resource that once existed has been permanently deleted or relocated. CDMI does not specify a mechanism for determining this, so this return code is HTTP specific.	The container has been deleted or moved without a forwarding address.

All operations may return a Location header if an object path is a CDMI reference (i.e., a soft link). Clients should use the URI in the Location header or message body as the real and cacheable address of the object (unless that is also a reference; caution must be exercised with references to detect cyclical references).

In the following detail tables, text in grey indicates content or fields that are outside the scope of CDMI but may be applicable based on HTTP semantics.

4.3 Operations on Containers

In CDMI, buckets are called *containers*. Unlike S3, most CDMI implementations allow containers to be nested, much as directories are in modern filesystems.

4.3.1 *DELETE Bucket*

Deleting a CDMI container deletes it, all of its metadata, and all of its contained objects. In S3, a DELETE operation on a bucket will fail if everything in it hasn't been deleted first, but in CDMI it should succeed, saving you the work of first deleting everything in the container. The appropriate headers and expectations for request and response bodies are shown below.

HTTP command	Example
DELETE *URI*	DELETE `/cs/classes/cs101/temp` HTTP/1.1

Header	Description
X-CDMI-Specification-Version	A comma-separated list of CDMI versions supported by the client, e.g., “`1.0.1, 1.0.2`”
Authorization	Authorization credentials supported by the server (see RFC 2616 section 14.8)

Request body
<none>

HTTP status codes	Response headers and body	Meaning
204 No Content 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 409 Conflict 410 Gone	See Table 8	See Table 8

4.3.2 *DELETE Bucket lifecycle*

S3 has a lifecycle control mechanism. One places a "lifecycle" on a bucket that specifies by which pattern to match objects and a time period that is equivalent to a retention period. Unlike enterprise retention systems, however, S3 allows one to update and delete buckets and objects in them that are under lifecycle control.

By contrast, CDMI uses a XAM-like[2] retention and compliance mechanism that is intended to be compliant with Sarbanes/Oxley and other compliance legislation.[3] CDMI servers will not permit a retention period to be deleted or reset to a shorter interval than the one that already exists on an object. This constraint is the same one that is used by enterprise retention systems.

There is, therefore, no direct equivalent to a lifecycle in CDMI 1.0.2. If you use retention to emulate it, CDMI will not allow you to delete the retention interval. Therefore, this operation is not supported directly by CDMI.

[2] XAM (the eXtensible Access Method) is a specification developed in SNIA, the Storage Networking Industry Organization. The spec has been retired, but its rich metadata schema has been incorporated, in large part, into CDMI.

[3] Due to the peculiarities of the law, it is not possible to affirm compliance. Instead, companies submit detailed compliance program plans incorporating various products and apply to have the program approved.

However, since CDMI supports nested containers, an alternative method can accomplish the same aims. Lifecycles are normally used to delete log files and other data that has a limited scope in time. By simply placing each day's files in a new container, it is possible to delete the files at the end of the lifecycle period by simply deleting the container.

Alternatively, if you use the method in section 4.3.19 PUT Bucket lifecycle, deleting the user metadata item “`s3_lifecycle`” from a container will have the desired effect, as the container will no longer match the daily query that you set up.

HTTP command	Example
PUT *URI* `?metadata:item`	PUT `/cs/classes/cs101/?metadata` HTTP/1.1

Header	Description
X-CDMI-Specification-Version	A comma-separated list of CDMI versions supported by the client, e.g., `"1.0.1, 1.0.2"`
Content-Type	"application/cdmi-object"
Authorization	Authorization credentials supported by the server (see RFC 2616 section 14.8)

Request body

```
{
    "s3_lifecycle": {}
}
```

HTTP status codes	Response headers and body	Meaning
204 No Content 302 Found 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 409 Conflict 410 Gone	See Table 8	See Table 8

4.3.3 *DELETE Bucket policy*

There is no support in CDMI 1.0.2 for bucket policies. CDMI uses capabilities and an access control mechanism modeled on NFSv4's access control. See section 4.3.5 on Bucket acls.

An extension to CDMI allowing use of the "scope" mechanism (section 19 of the spec) as an access policy mechanism may be a possibility for future work.

4.3.4 *DELETE Bucket website*

Adding a new DNS entry is outside of the scope of CDMI. However, a CDMI PUT can be used to create a CDMI reference. See Object References in section 7.2 of CDMI. Deleting the reference is then equivalent to deleting a bucket website.

HTTP commands	**Example**
DELETE *URI*	DELETE `/cs101` HTTP/1.1

Header	**Description**
Authorization	Authorization credentials supported by the server (see RFC 2616 section 14.8).

Request body
<none>

HTTP status codes	**Response headers and body**	**Meaning**
204 No Content 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 409 Conflict 410 Gone	See Table 8	See Table 8

4.3.5 *GET Bucket acl*

CDMI ACLs are a superset of S3 ACLs. In S3, only READ, WRITE, READACF, WRITEACF, and FULLCONTROL permissions are available. Furthermore, you can only set ACLs on buckets.

CDMI, on the other hand, supports a rich set of permissions on both objects and containers, following the design of NFSv4. An ACL (which means "Access Control List") is composed of ACEs ("Access Control Entries"). Each ACE carries the information described in Table 9.

Table 9. ACE Entry Definitions

Name of Property	Meaning
acetype	ALLOW, DENY or AUDIT
identifier	A user id, group id, or one of the following special identifiers: OWNER@, GROUP@, EVERYONE@, ANONYMOUS@, AUTHENTICATED@, ADMINISTRATOR@, ADMINUSERS@
aceflags	Any combination of NO_FLAGS, OBJECT_INHERIT, CONTAINER_INHERIT, NO_PROPAGATE, INHERIT_ONLY, IDENTIFIER_GROUP, INHERITED
acemask	Any combination of READ_OBJECT, LIST_CONTAINER, WRITE_OBJECT, ADD_OBJECT, APPEND_DATA, ADD_SUBCONTAINER, READ_METADATA, WRITE_METADATA, EXECUTE, DELETE_OBJECT, DELETE_SUBCONTAINER, READ_ATTRIBUTES, WRITE_ATTRIBUTES, WRITE_RETENTION, WRITE_RETENTION_HOLD, DELETE, READ_ACL, WRITE_ACL, WRITE_OWNER, SYNCHRONIZE

Table 10 displays the mapping between S3 and CDMI permissions.

Table 10. Mapping Between S3 and CDMI Permissions

S3 permission	CDMI permission flags and mask bits
READ	OBJECT_INHERIT, READ_OBJECT (object only), LIST_CONTAINER (bucket only), READ_METADATA, EXECUTE, READ_ATTRIBUTES
WRITE	OBJECT_INHERIT, WRITE_OBJECT (object only), ADD_OBJECT (bucket only), WRITE_METADATA, DELETE_OBJECT, WRITE_ATTRIBUTES
READACF	READ_ACL
WRITEACF	WRITE_ACL
FULLCONTROL	READ_OBJECT, LIST_CONTAINER, WRITE_OBJECT, ADD_OBJECT, APPEND_DATA, ADD_SUBCONTAINER, READ_METADATA, WRITE_METADATA, EXECUTE, DELETE_OBJECT, DELETE_SUBCONTAINER, READ_ATTRIBUTES, WRITE_ATTRIBUTES, WRITE_RETENTION, WRITE_RETENTION_HOLD, DELETE, READ_ACL, WRITE_ACL, WRITE_OWNER

Refer to section 16.1 of the 1.0.2 spec for more information.

ACLs can be "inherited" in CDMI. When a CDMI object is accessed, CDMI traverses up the container hierarchy to the root, collecting all ACLs that apply to the object and building a virtual ACL that represents the mathematical union of all the applicable ACLs in the hierarchy. In this case, the union is all of the ACEs in which the identifier is the user making the request or a group that contains the user or one of the groups. Because containers may be nested and ACLs set on both child objects and child containers, various flags can be set to control whether a child inherits a specific ACE from the parent (see the aceflags above).

ACLs are admittedly complex, but as CDMI largely copies the NFSv4 ACL mechanism and NFSv4 ACLs were patterned after Windows ACLs, a rich body of literature is available online regarding their use.

The following example GETs the ACL (which in this case happens to be the default ACL mandated by the spec) on a top-level container (the equivalent of a bucket). You must have READ_ACL permission on the container.

HTTP commands	Example
GET *URI* ?metadata:cdmi_acl	GET `/cs/?metadata:cdmi_acl` HTTP/1.1

Header	Description
X-CDMI-Specification-Version	A comma-separated list of CDMI versions supported by the client, e.g., `“1.0.1, 1.0.2”`.
Accept	"application/cdmi-object"
Authorization	Authorization credentials supported by the server (see RFC 2616 section 14.8).

Request body
<none>

HTTP status codes	Response headers and body	Meaning
200 Okay	X-CDMI-Specification-Version `1.0.2`	Highest version supported by both client and server
	Content-Type `"application/cdmi-object"`	

HTTP status codes	Response headers and body	Meaning
200 Okay	`{ "cdmi_acl": [ { "acetype": "ALLOW", "identifier": "OWNER@", "aceflags": "OBJECT_INHERIT, CONTAINER_INHERIT", "acemask": "ALL_PERMS" }, { "acetype": "ALLOW", "identifier": "AUTHENTICATED@", "aceflags": "OBJECT_INHERIT, CONTAINER_INHERIT", "acemask": "READ" } ] }`	
302 Found 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 406 Not Acceptable 410 Gone	See Table 8	See Table 8

4.3.6 *GET Bucket lifecycle*

There is no direct equivalent to a lifecycle in CDMI 1.0.2 (see section 4.3.2 DELETE Bucket lifecycle for more discussion).

If you use the method in section 4.3.19 PUT Bucket lifecycle, fetching the user metadata item “s3_lifecycle” from a container will show the lifecycle that has been placed on the container. Remember that the server does not automatically delete anything marked with this method. You are responsible for setting up a daily batch job to do that.

The following request will fetch the lifecycle.

HTTP command	Example
GET *URI* ?metadata:item	GET `/cs/classes/cs101/?metadata:s3_lifecycle` HTTP/1.1

Header	Description
X-CDMI-Specification-Version	A comma-separated list of CDMI versions supported by the client, e.g., `"1.0.1, 1.0.2"`.
Content-Type	`"application/cdmi-object"`

Header	Description
Authorization	Authorization credentials supported by the server (see RFC 2616 section 14.8).

Request body
<none>

HTTP status codes	Response headers and body	Meaning
302 Found 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 409 Conflict 410 Gone	See Table 8	See Table 8
200 Okay	X-CDMI-Specification-Version `1.0.2`	Highest version supported by both client and server
	Content-Type `"application/cdmi-object"`	

	`{`
	    `"metadata" : {`
	        `"s3_lifecycle" : "30"`
	    `}`
	`}`	

4.3.7 *GET Bucket policy*

There is no support in CDMI 1.0.2 for bucket policies. CDMI uses capabilities and an access control mechanism modeled on NFSv4's access control. See section 4.3.5 on GET Bucket acls.

An extension to CDMI allowing use of the "scope" mechanism (section 19 of the spec) as an access policy mechanism may be a possibility for future work.

4.3.8 *GET Bucket location*

If the server supports geographic placement, as indicated by the cdmi_geographic_placement capability, the following request will return the geographic placements previously requested. This is not quite the same as an indication of where the data actually is sitting—cloud systems need to be able to manage that to meet shifting load and other constraints. Instead, indicates which locations are acceptable and which are not. See Table 118 in the spec.

Note that CDMI has the ability to specify geographic regions in which data should *not* be stored.

HTTP command	Example
GET *URI* ?metadata:item	GET `/cs/classes/cs101/?metadata:cdmi_geographic_placement` HTTP/1.1

Header	Description
X-CDMI-Specification-Version	A comma-separated list of CDMI versions supported by the client, e.g., `"1.0.1, 1.0.2"`.
Content-Type	`"application/cdmi-object"`
Authorization	Authorization credentials supported by the server (see RFC 2616 section 14.8).

Request body
<none>

HTTP status codes	Response headers and body	Meaning
302 Found 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 409 Conflict 410 Gone	See Table 8	See Table 8
200 Okay	X-CDMI-Specification-Version `1.0.2`	Highest version supported by both client and server
	Content-Type `"application/cdmi-object"`	
	`{ "metadata" : { "cdmi_geographic_placement" : [ "CA", "US" ] } }`	

4.3.9 *GET Bucket logging*

CDMI supports logging in some depth, using logging queues, and a full discussion of it would be a whitepaper in its own right. To summarize, logging queues can be instantiated on object operations, security events, and data management events. The entries in logging queues contain more information than the entries in notification queues. Please refer to chapter 20 of the CDMI spec.

HTTP command	Example
GET *URI* ?metadata:item	GET `/cs/classes/cs101/log?metadata` HTTP/1.1

Header	Description
X-CDMI-Specification-Version	A comma-separated list of CDMI versions supported by the client, e.g., `"1.0.1, 1.0.2".`
Content-Type	`"application/cdmi-queue"`
Authorization	Authorization credentials supported by the server (see RFC 2616 section 14.8).

Request body
<none>

HTTP status codes	Response headers and body	Meaning
302 Found 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 409 Conflict 410 Gone	See Table 8	See Table 8
200 Okay	X-CDMI-Specification-Version `1.0.2`	Highest version supported by both client and server
	Content-Type `"application/cdmi-object"`	

HTTP status codes	Response headers and body	Meaning

	{
	    "metadata" : {
	        "cdmi_queue_type" : "cdmi_logging_queue",
	        "cdmi_logging_class" : [
	            "cdmi_object_logging",
	            "cdmi_security_logging"
	        ],
	        "cdmi_scope_specification" : [
	            {
	                "domainURI" : "==
	/cdmi_domains/cs101/"
	            }
	        ]
	    }
	}	

4.3.10 *GET Bucket notification*

CDMI supports notification queues, which are similar to logging queues but do not contain as much information. They are intended for management clients that need to know when objects in the system change. People familiar with CIM or SMI-S would call them lifecycle indications. See chapter 21 in the spec.

S3, however, only supports one notification, which is that data loss has occurred. There is no support in CDMI for this idea.

4.3.11 *GET Bucket Object versions*

CDMI 1.02 does not support versioning. There is a versioning extension under consideration in the working group.

4.3.12 *GET Bucket requestPayment*

CDMI does not support the concept of a third party responsible for payment of fees. This is determined out of band with the CDMI service provider.

4.3.13 *GET Bucket versioning*

CDMI 1.02 does not support versioning. There is a versioning extension under consideration in the working group.

4.3.14 *GET Bucket website*

Manipulating DNS entries is outside of the scope of CDMI. However, a CDMI PUT can be used to create a CDMI Reference. See Object References in section 7.2 of the CDMI spec. Accessing the reference provides a similar kind of redirection to a bucket website.

HTTP commands	Example
GET URI	GET `/cs101/` HTTP/1.1

Header	Description
Authorization	Authorization credentials supported by the server (see RFC 2616 section 14.8).

Request body
<none>

HTTP status codes	Response headers and body	Meaning
200 Okay	X-CDMI-Specification-Version `1.0.2`	Highest version supported by both client and server
	Content-Type `"application/cdmi-object"`	
	`{ "reference" : "cs/classes/cs101/" }`	
400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 409 Conflict 410 Gone	See Table 8	See Table 8

4.3.15 *HEAD Bucket*

CDMI does not support the HEAD operation, but it can be easily emulated by simply fetching a container's metadata.

HTTP commands	Example
GET *URI*?metadata	GET `/cs/classes/cs101/?metadata` HTTP/1.1

Header	Description
Authorization	Authorization credentials supported by the server (see RFC 2616 section 14.8).

Request body
<none>

HTTP status codes	Response headers and body	Meaning
204 No Content 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 409 Conflict 410 Gone	See Table 8	See Table 8
200 Okay	X-CDMI-Specification-Version `1.0.2`	Highest version supported by both client and server
	Content-Type `"application/cdmi-object"`	

	`{`
	`  "metadata" : {`
	`     <normally a long list>`
	`  }`
	`}`
		All metadata the user has permission to access. If, for example, the user does have permission to read metadata, but doesn't have permission to read the ACL, a partial list not containing the ACL will be returned.

4.3.16 *List Multipart Uploads*

CDMI supports multipart uploads via the X-CDMI-Partial header and the RFC 2616 Range and other headers. There is, at present, no support for upload identifiers such as those that S3 returns. The client is responsible for tracking which partial uploads have been acknowledged as successful by the server.

4.3.17 *PUT Bucket*

In S3, an ACL can be specified at the time a bucket is created.

When a container is created in CDMI, it inherits the ACL of the parent container. To modify the ACL, follow the directions in the next section.

The create operation is a simple PUT with no request body. The container name MUST end with a forward slash ("/").

HTTP commands	Example
PUT URI	PUT `/cs/classes/cs101/lesson1/` HTTP/1.1

Header	Description
Authorization	Authorization credentials supported by the server (see RFC 2616 section 14.8).

Request body
<none>

HTTP status codes	Response headers and body	Meaning
204 No Content 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 409 Conflict 410 Gone	See Table 8	See Table 8

4.3.18 *PUT Bucket ACL*

To put an ACL on a bucket you must have knowledge of user and group names. You can get these by querying the domain in which you are working. You may need to contact a cloud administrator if you don't know this.

To place an ACL on a container, construct a JSON object containing an array of ACEs per section 16.1 of the spec. In this example, "OWNER@" and "AUTHENTICATED@" are special wildcards, meaning the owner of the object and any authenticated user, respectively.

The order of ACEs in an ACL is important. Normally DENY ACEs are placed first, and ALLOW ACEs after. There is one exception: suppose `jdoe` is a member of the group `birthday` and you want to allow her access to a secret birthday party planning document and prevent others in the group from seeing it. In that case, an ALLOW ACE for `jdoe` should come before a DENY ACE for the group `birthdays`.

In general, good practice is to place any DENY ACEs you wish to add at the head of the ACL list and any ALLOW ACEs at the end of the list, preserving the existing order of entries intact.

HTTP commands	Example
PUT *URI*?metadata:cdmi_acl	PUT `/cs/classes/cs101/lesson1/?metadata:cdmi_acl` HTTP/1.1

Header	Description
Authorization	Authorization credentials supported by the server (see RFC 2616 section 14.8).

Request body

```
{
  "cdmi_acl":
  [
    {
      "acetype": "ALLOW",
      "identifier": "OWNER@",
      "aceflags": "OBJECT_INHERIT,
                   CONTAINER_INHERIT",
      "acemask": "ALL_PERMS"
    },
    {
      "acetype": "ALLOW",
      "identifier": "AUTHENTICATED@",
      "aceflags": "OBJECT_INHERIT,
                   CONTAINER_INHERIT",
      "acemask": "READ"
    }
  ]
}
```

HTTP status codes	Response headers and body	Meaning
204 No Content 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 409 Conflict 410 Gone	See Table 8	See Table 8

4.3.19 *PUT Bucket lifecycle*

There is no direct equivalent to a lifecycle in CDMI 1.0.2 (see section 4.3.2). To do the equivalent, first determine whether the objects you wish to place under lifecycle control will ever need to be updated. If they are video surveillance files or some other kind of log, it may actually be advantageous to use CDMI retention, as that will actively prevent changes to them—a desirable feature in a log store.

Alternatively, for logs and other things that can be deleted after a fixed period of time, it may be convenient to simply rename their container each day and create a new one. CDMI deletes all the objects in a container when it is deleted, so a single operation at expiration time deletes many log files.

If you need more than that, you can put a user-defined metadata item on the files on which you want to place a lifecycle. You'll need to create a query queue that has a scope that matches the objects on which you want to put a lifecycle and then dequeue and perform the following operation on each object in the queue. In addition, you'll need to set up a cron job or the equivalent to run once a day, recreate the query queue, and examine the s3_lifecycle metadata item and delete the object if it is time (see sections 11.2 of the spec and 4.4.6 and 4.4.1 in this document).

HTTP commands	Example
PUT *URI*?metadata:s3_lifecycle	PUT `/cs/classes/cs101/lesson1/?metadata:s3_lifecycle` HTTP/1.1

Header	Description
Authorization	Authorization credentials supported by the server (see RFC 2616 section 14.8).

Request body
`{ "s3_lifecycle" : 30 }`

```
{
    "s3_lifecycle" : 30
}
```

HTTP status codes	Response headers and body	Meaning
204 No Content 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 409 Conflict 410 Gone	See Table 8	See Table 8

4.3.20 *PUT Bucket policy*

There is no support in CDMI 1.0.2 for bucket policies. CDMI uses capabilities and an access control mechanism modeled on NFSv4's access control. See section 4.3.5 on Bucket acls.

An extension to CDMI allowing use of the "scope" mechanism (section 19 of the spec) as an access policy mechanism may be a possibility for future work.

4.3.21 *PUT Bucket logging*

CDMI supports a richer set of logging and notification operations than S3; it does not, however, document the concept of a data loss notification (possibly because its architects are from the enterprise storage space where data loss is not an option). The possibilities are encapsulated in the queue object type.

HTTP commands	Example of creating a notification queue
PUT URI/queuename	PUT `/cs/classes/cs101/lesson1/questions` HTTP/1.1

Header	Description
Accept	`"application/cdmi-queue"`
Content-Type	`"application/cdmi-queue"`
X-CDMI-Specification-Version	`"1.0.1, 1.0.2"`
Authorization	Authorization credentials supported by the server (see RFC 2616 section 14.8).

Request body	

`{`
`    "metadata" : { <metadata> },`
`    "domainURI" : <URI>,`
`    "copy" : <URI>,`
`    "move" : <URI>,`
`    "cdmi_queue_type" :`
`"cdmi_logging_queue",`
`    "cdmi_logging_class" : [`
`        "cdmi_object_logging",`
`        "cdmi_security_logging"`
`    ],`
`    "cdmi_scope_specification" : [`
`        {`
`            "parentURI" : "==`
`/cs/classes/cs101/"]`
`        }`
`    ]`
`}`
	Optional – see spec Table 121
	Optional – see spec
	Optional – copy from URI
	Optional – move from URI
	Optional – specify type of queue
	
	See spec section 18.
	This example asks for notifications for all object and security events in the cs101 container

HTTP status codes	Response headers and body	Meaning

201 Created	Content-Type
	`"application/cdmi-queue"`
	X-CDMI-Specification-Version
	`"1.0.2"`
	`{`
	`  "objectType": "application/cdmi-queue",`
	`  "objectID": "00007E7F00104BE66AB53A9572F9F51E",`
	`  "objectName" : "questions",`
	`  "parentURI " : "/cs/classes/cs101/lesson1/",`
	`  "parentID" : "0000706D0010B84FAD185C425D8B537E",`
	`  "domainURI" : "/",`
	`  "capabilitiesURI" : "/cdmi_capabilities/queue/",`
	`  "completionStatus" : "Complete",`
	`  "queueValues" : "",`
	`  "metadata" : { <metadata> }`
	`}`	

202 Accepted	Content-Type
	`"application/cdmi-queue"`
	X-CDMI-Specification-Version
	`"1.0.2"`
	Location
	`/cs/classes/cs101/lesson1/questions`
	`{`
	`  "objectType": "application/cdmi-queue",`
	`  "objectID": "00007E7F00104BE66AB53A9572F9F51E",`
	`  "objectName" : "questions",`
	`  "parentURI " : "/cs/classes/cs101/lesson1/",`
	`  "parentID" : "0000706D0010B84FAD185C425D8B537E",`
	`  "domainURI" : "/",`
	`  "capabilitiesURI" : "/cdmi_capabilities/queue/",`
	`  "completionStatus" : "Processing",`
	`  "percentComplete" : "21",`
	`  "queueValues" : "",`
	`}`
		The operation has been authorized and is in process. There may be access delays. Delays typically occur when a "create" is a copy or a move and the queue has many entries.

204 No Content 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 409 Conflict 410 Gone	See Table 8	See Table 8

This has only been one example of creating a logging queue. For more discussion, see sections 18 and 20 of the CDMI spec.

4.3.22 *PUT Bucket notification*

CDMI supports notification queues, which are similar to logging queues but do not contain as much information. They are intended for management clients that need to know when objects in the system change. People familiar with CIM or SMI-S would call them lifecycle indications. See chapter 21 in the spec.

S3, however, only supports one notification, which is that data loss has occurred. There is no support in CDMI for this idea.

However, CDMI supports a rich set of notification functionality. Notifications are mainly different from logging in that you are able to specify exactly what information gets sent in the notification. Please see sections 18 and 21 of the spec for more information.

4.3.23 *PUT Bucket requestPayment*

CDMI does not support the concept of a third party that is responsible for payment of fees. This is determined out of band with the CDMI service provider.

4.3.24 *PUT Bucket versioning*

CDMI 1.02 does not support versioning in the version 1 spec. There is a versioning extension under consideration in the working group.

4.3.25 *PUT Bucket website*

Adding a new DNS entry is outside of the scope of CDMI and should be done using Dynamic DNS (DDNS). However, a CDMI PUT to an available address on the server, ending in "?", will create a CDMI Reference.

HTTP commands	Example
PUT URI	PUT `/cs101?` HTTP/1.1

Header	Description
Authorization	Authorization credentials supported by the server (see RFC 2616 section 14.8).

Request body
`{ "reference" : "cs/classes/cs101/" }`

HTTP status codes	Response headers and body	Meaning
204 No Content 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 409 Conflict 410 Gone	See Table 8	See Table 8

4.4 Operations on Objects

4.4.1 *DELETE Object*

Deleting an object requires DELETE_OBJECT privilege on its parent container and WRITE_METADATA privilege on the object (see Table 116 – ACE Bit Masks – in the 1.0.2 spec).

HTTP command	Example
DELETE *URI*	DELETE `/cs/classes/cs101/temp` HTTP/1.1

Header	Description
X-CDMI-Specification-Version	A comma-separated list of CDMI versions supported by the client, e.g., `“1.0.1, 1.0.2”.`
Authorization	Authorization credentials supported by the server (see RFC 2616 section 14.8).

Request body
<none>

HTTP status codes	Response headers and body	Meaning
204 No Content 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 409 Conflict 410 Gone	See Table 8	See Table 8

4.4.2 *Delete Multiple Objects*

Deletion of multiple objects must be done one by one in CDMI unless they are grouped in a container. In that case, successfully deleting the container will delete the objects in the container as well.

This fact may encourage you to create containers for grouping data that you know will be deleted at some point. You can still access objects via object id.

A jobs extension under consideration in the Working Group provides another method for managing bulk deletion.

4.4.3 *GET Object*

GETting and object via CDMI returns the object and all the metadata that the principal making the query has privilege to access. To receive just the object's "body," query the "value" component of the object.

HTTP commands	**Example**
GET *URI*	`GET /cs/cs101/lesson1/dates?value` HTTP/1.1

Header	**Description**
X-CDMI-Specification-Version	A comma-separated list of CDMI versions supported by the client, e.g., `“1.0.1, 1.0.2”.`
Accept	`"application/cdmi-object"`
Authorization	Authorization credentials supported by the server (see RFC 2616 section 14.8).

Request body
<none>

HTTP status codes	**Response headers and body**	**Meaning**
200 Okay	X-CDMI-Specification-Version 1.0.2	Highest version supported by both client and server
	Content-Type `"application/cdmi-object"`	

	`{`
	`    "value" :`
	`         "This assignment is due Sep 15"`
	`}`	

HTTP status codes	Response headers and body	Meaning
202 Accepted 302 Found 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 406 Not Acceptable 410 Gone	See Table 8 and section 4.3.21.	See Table 8

GETting an object without request headers is equivalent to a non-CDMI GET and returns the "value" component of the object.

You can get portions of an object. See spec section 8.4.8 for examples.

4.4.4 *GET Object ACL*

To read an ACL, simply specify the "metadata:cdmi_acl" location in the query

HTTP commands	Example
GET *URI*	GET `/cs/cs101/lesson1/dates?metadata:cdmi_acl` HTTP/1.1

Header	Description
X-CDMI-Specification-Version	A comma-separated list of CDMI versions supported by the client, e.g., `“1.0.1, 1.0.2”.`
Accept	`"application/cdmi-object"`
Authorization	Authorization credentials supported by the server (see RFC 2616 section 14.8).

Request body
<none>

HTTP status codes	Response headers and body	Meaning
200 Okay	X-CDMI-Specification-Version 1.0.2	Highest version supported by both client and server
	Content-Type `"application/cdmi-object"`	

HTTP status codes	Response headers and body	Meaning

	`{`
	`  "cdmi_acl":`
	`  [`
	`    {`
	`      "acetype": "ALLOW",`
	`      "identifier": "OWNER@",`
	`      "acemask": "ALL_PERMS"`
	`    },`
	`    {`
	`      "acetype": "ALLOW",`
	`      "identifier":`
	`"AUTHENTICATED@",`
	`      "acemask": "READ"`
	`    }`
	`  ]`
	`}`	

202 Accepted 302 Found 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 406 Not Acceptable 410 Gone	See Table 8 and section 4.3.21.	See Table 8

4.4.5 *GET Object torrent*

CDMI does not support BitTorrent downloads. A vendor wishing to support them would very likely store the URI of the associated torrent in a field in the object's metadata:torrentURI field. In this case, a GET to <objectpath>?metadata:torrentURI would return the URI to use for the download.

4.4.6 *HEAD Object*

CDMI does not directly support the HEAD operation, but the equivalent can be done by just GETting the object's metadata.

HTTP commands	Example
GET *URI*?metadata	GET `/cs/classes/cs101/dates?metadata` HTTP/1.1

Header	Description
X-CDMI-Specification-Version	A comma-separated list of CDMI versions supported by the client, e.g., `“1.0.1, 1.0.2”.`

Header	Description
Accept	`"application/cdmi-object"`
Authorization	Authorization credentials supported by the server (see RFC 2616 section 14.8)

Request body
<none>

HTTP status codes	Response headers and body	Meaning
200 Okay 202 Accepted	X-CDMI-Specification-Version 1.0.2	Highest version supported by both client and server
	Content-Type `"application/cdmi-object"`	
202 Accepted 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 409 Conflict 410 Gone	See Table 8 and section 4.3.21	See Table 8

200 Okay	`{`
	`  "metadata" : {`
	`        <normally a long list>`
	`    }`
	`}`
		All metadata the user has permission to access. If, for example, the user does have permission to read metadata, but doesn't have permission to read the ACL, a partial list not containing the ACL will be returned. An empty JSON object normally means that the user doesn't have privilege to read any metadata.

4.4.7 *POST Object*

CDMI uses the POST operation differently from S3. In CDMI, it is used to create objects that do not have paths (just object IDs in a flat namespace). S3 uses POST in the more traditional sense, allowing HTTP headers to be passed as form fields from browserbased applications. There is no support for this as a CDMI operation, but a non-CDMI POST should work as expected, providing the cdmi_post_dataobject capability is present.

4.4.8 *PUT Object*

CDMI offers support for object updates (by specifying a byte range, e.g., `myObject?value:1048576-1081344`). S3 does not have this capability. So, when you PUT an object to S3, it is effectively always a new copy unless you are using versioning.

Other than that, the two systems operate similarly.

HTTP commands	Example
PUT *URI*	PUT `/cs/classes/cs101/lesson1/lesson1.html` HTTP/1.1

Header	Description
X-CDMI-Specification-Version	A comma-separated list of CDMI versions supported by the client, e.g., `“1.0.1, 1.0.2”.`
Accept	`"application/cdmi-object"`
Content-Type	`"application/cdmi-object"`
Authorization	Authorization credentials supported by the server (see RFC 2616 section 14.8).

Request body (optional fields are in grey)	

`{`
`    "mimetype" : "text/html",`
`    "metadata" : { <user metadata> },`
`    "domainURI" : "/cs/",`
`    "copy" : "/cs/prof/mrc/cs101/1.htm",`
`    "move" : "/cs/prof/mrc/cs101/tmp.htm",`
`    "value" : "<escaped utf-8 string>",`
`    "valuetransferencoding" : [ "utf-8" ]`
`}`
	Notes: mimetype is set to `"text/plain"` if not supplied. The domain of the parent container is used by default. Only one of copy, move, or value can be specified.
	The order of the fields does not matter.

HTTP status codes	Response headers and body	Meaning
201 Created 202 Accepted	X-CDMI-Specification-Version 1.0.2	Highest version supported by both client and server
	Content-Type `"application/cdmi-object"`	

HTTP status codes	Response headers and body	Meaning
202 Accepted 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 409 Conflict 410 Gone	See Table 8	See Table 8

201 Created	`{`
	`  "objectType" : "application/cdmi-object",`
	`  "objectID" : "000706D0010B84FAD185C425D8B537E",`
	`  "objectName" : "lesson1.html",`
	`  "parentURI" : "/cs/classes/cs101/lesson1/",`
	`  "parentID" : "00007E7F00102E230ED82694DAA975D2",`
	`  "domainURI" : "/cs/",`
	`  "capabilitiesURI" : "cdmi_capabilities/dataobject/",`
	`  "completionStatus" : "Complete",`
	`  "mimetype" : "text/html",`
	`  "metadata" : { <long list> }`
	`}`	

204 No Content	When an object is updated by overwriting it, as S3 requires one to do, CDMI returns No Content on writes subsequent to the initial create.	

4.4.9 *PUT Object acl*

S3 allows you to set ACLs using request headers. CDMI does not support this, but it does support a complete NFSv4- and Windows-compatible ACL structure. See section 4.4.9 for more detail.

HTTP commands	Example
PUT *URI*?metadata:cdmi_acl	PUT `/cs/classes/cs101/lesson2/sample1?metadata:cdmi_acl` HTTP/1.1

Header	Description
Authorization	Authorization credentials supported by the server (see RFC 2616 section 14.8).

Request body

`{`
`  "cdmi_acl":`
`  [`
`    {`
`      "acetype": "ALLOW",`
`      "identifier": "OWNER@",`
`      "acemask": "ALL_PERMS"`
`    },`
`    {`
`      "acetype": "ALLOW",`
`      "identifier": "cs101",`
`      "acemask": "READ"`
`    }`
`  ]`
`}`

HTTP status codes	Response headers and body	Meaning
204 No Content 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 409 Conflict 410 Gone	See Table 8	See Table 8

4.4.10 *PUT Object - Copy*

To copy an object from one address to another in CDMI, use the "copy" field in the request body, as implied in section 4.4.8.

HTTP commands	Example
PUT *URI*	PUT `/cs/classes/cs101/lesson1/lesson1.html` HTTP/1.1

Header	Description
X-CDMI-Specification-Version	A comma-separated list of CDMI versions supported by the client, e.g., “`1.0.1, 1.0.2`”.
Accept	`"application/cdmi-object"`
Content-Type	`"application/cdmi-object"`

Header	Description
Authorization	Authorization credentials supported by the server (see RFC 2616 section 14.8).

Request body (optional fields are in grey)	

`{`
`    "mimetype" : "text/html",`
`    "metadata" : { <user metadata> },`
`    "domainURI" : "/cs/",`
`    "copy" : "/cs/prof/mrc/cs101/1.htm",`
`    "valuetransferencoding" : [ "utf-8" ]`
`}`
	Notes: mimetype is set to `"text/plain"` if not supplied. The domain of the parent container is used by default.
	The order of the fields does not matter.

HTTP status codes	Response headers and body	Meaning
201 Created 202 Accepted	X-CDMI-Specification-Version 1.0.2	Highest version supported by both client and server
	Content-Type `"application/cdmi-object"`	
202 Accepted 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 409 Conflict 410 Gone	See Table 8	See Table 8

201 Created	`{`
	`  "objectType" : "application/cdmi-object",`
	`  "objectID" :`
	`       "000706D0010B84FAD185C425D8B537E",`
	`  "objectName" : "lesson1.html",`
	`  "parentURI" : "/cs/classes/cs101/lesson1/",`
	`  "parentID" :`
	`       "00007E7F00102E230ED82694DAA975D2",`
	`  "domainURI" : "/cs/",`
	`  "capabilitiesURI" :`
	`       "cdmi_capabilities/dataobject/",`
	`  "completionStatus" : "Complete",`
	`  "mimetype" : "text/html",`
	`  "metadata" : { <long list> }`
	`}`	

HTTP status codes	Response headers and body	Meaning
204 No Content	When an object is updated by overwriting it, as S3 requires one to do, CDMI returns No Content on writes subsequent to the initial create.	

4.4.11 *Initiate Multipart Upload*

CDMI supports multipart upload using the standard mechanisms of HTTP for doing so, with an additional header ("X-CDMI-Partial") indicating to CDMI whether the upload is complete.

Suppose we need to upload a file in three parts, with byte ranges 0-99, 100-199 and 200-299. (These are small byte ranges in the example; real ranges would likely be much larger). The following three queries accomplish this:

```
PUT /cs/cs101/lesson1/mpfile.txt HTTP/1.1
X-CDMI-Specification-Version "1.0.1, 1.0.2"
Accept "application/cdmi-object"
Content-Type "application/cdmi-object"
X-CDMI-Partial "false"
Range 0-99
{
    "value" : "<100 bytes of data"
}

PUT /cs/cs101/lesson1/mpfile.txt HTTP/1.1
X-CDMI-Specification-Version "1.0.1, 1.0.2"
Accept "application/cdmi-object"
Content-Type "application/cdmi-object"
X-CDMI-Partial "false"
Range 100-199
{
    "value" : "<100 bytes of data"
}

PUT /cs/cs101/lesson1/mpfile.txt HTTP/1.1
X-CDMI-Specification-Version "1.0.1, 1.0.2"
Accept "application/cdmi-object"
Content-Type "application/cdmi-object"
X-CDMI-Partial "true"
Range 200-299
{
    "value" : "<100 bytes of data"
}
```

The response codes, headers, and body are identical to those of an ordinary PUT as documented in section 4.4.8.

4.4.12 *Upload Part*

To upload any part except the last one, specify the location of the part in the object using the HTTP Range header and an X-CDMI-Partial header set to "false". The example in section 4.4.11 shows this. Note that the ranges are not required to be sent or received in byte range order.

4.4.13 *Upload Part - Copy*

CDMI v1.0.2 does not support partial object copy. The ”Copy Range” extension provides this functionality.

4.4.14 *Complete Multipart Upload*

During a multipart upload, the object's existence and metadata are visible, but the value is not. To tell the server that the upload is complete and expose the value, set the X-CDMI-Partial header to "true". This can be seen in the example in section 4.4.11.

4.4.15 *Abort Multipart Upload*

To abort a multipart upload in CDMI, DELETE the object. There is no "rollback" mechanism for "undoing" the uploads made so far. The versioning mechanism, if offered by the CDMI vendor, could be used to provide a rollback.

4.4.16 *List Parts*

CDMI 1.0.2 does not support listing of partial uploads. The client is responsible for tracking them and ensuring that normal responses are received for all parts.

5 Acknowledgements

Thanks go to many members of the SNIA Green Technical Working Group and to the following individuals and companies in particular:

- David Slik – NetApp
- Mark Carlson – Oracle
- Doug Davis – IBM
- Tong Li – IBM