Cloud Data Management Interface (CDMI™)

Version 2.0.0

ABSTRACT: This CDMI International Standard is intended for application developers who are implementing or using cloud storage. It documents how to access cloud storage and to manage the data stored there.

This document has been released and approved by the SNIA. The SNIA believes that the ideas, methodologies, and technologies described in this document accurately represent the SNIA goals and are appropriate for widespread distribution. Suggestion for revision should be directed to http://www.snia.org/feedback/.

SNIA Technical Position

April 1, 2020

USAGE

DISCLAIMER

The information contained in this publication is subject to change without notice. The SNIA makes no warranty of any kind with regard to this specification, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The SNIA shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this specification.

Suggestions for revisions should be directed to https://www.snia.org/feedback/.

Table of Contents:

List of Figures

List of Tables

Part I

CDMI Preamble

412 This Cloud Data Management Interface (CDMI™) International Standard is intended for application developers who are
413 implementing or using cloud storage. It documents how to access cloud storage and to manage the data stored there.

414 This document is organized as follows:

415

416 Table 1: Overview of this document

Clause 1	Scope	Defines the scope of this document
Clause 2	Normative references	Lists the normative references for this document
Clause 3	Terms	Provides terminology used in this document
Clause 4	Conventions	Describes the conventions used in presenting the interfaces and the typographical conventions used in this document
Clause 5	Overview of Cloud Storage	Provides a brief overview of cloud storage and details the philosophy behind this International Standard as a model for the operations
Clause 6	Data Object Resource Operations using HTTP	Provides the normative standard of data object resource operations using HTTP
Clause 7	Container Object Resource Operations using HTTP	Provides the normative standard of container object resource operations using HTTP
Clause 8	Data Object Resource Operations using CDMI	Provides the normative standard of data object resource operations using CDMI
Clause 9	Container Object Resource Operations using CDMI	Provides the normative standard of container object resource operations using CDMI
Clause 10	Domain Object Resource Operations using CDMI	Provides the normative standard of domain object resource operations using CDMI
Clause 11	Queue Object Resource Operations using CDMI	Provides the normative standard of queue object resource operations using CDMI
Clause 12	Capability Object Resource Operations using CDMI	Provides the normative standard of capability object resource operations using CDMI
Clause 13	Exported Protocols	Discusses how virtual machines in the cloud computing environment can use the exported protocols from CDMI containers
Clause 14	Snapshots	Discusses how snapshots are accessed under CDMI containers
Clause 15	Serialization/ Deserialization	Discusses serialization and deserialization, including import and export of serialized data under CDMI
Clause 16	Metadata	Provides the normative standard of the metadata used in the interface
Clause 18	Retention and Hold Management	Describes the optional retention management disciplines to be implemented into the system management functions
Clause 19	Scope Specification	Describes the structure of the scope specification for JSON objects
Clause 20	Results Specification	Provides a standardized mechanism to define subsets of CDMI object contents
Clause 21	Notification Queues	Describes how CDMI clients can efficiently discover what changes have occurred to the system
Clause 22	Query Queues	Describes how CDMI clients can efficiently discover what content matches a given set of metadata query criteria or full-content search criteria
Clause 23	Encrypted Objects	Describes how to work with transparently encrypted objects
Clause 24	Delegated Access Control	Describes how to delegate access control to external systems
Clause 25	Data Object Versions	Describes how to work with versioned data objects
Clause 26	Extensions	Provides informative vendor extensions. Each extension is added to the standard when at least two vendors implement the extension.

Clause 1

Scope

This CDMI™ International Standard specifies the interface to access cloud storage and to manage the data stored therein. This International Standard applies to developers who are implementing or using cloud storage.

Clause 2

Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

The provisions of the referenced specifications other than ISO/IEC, IEC, ISO, and ITU documents, as identified in this clause, are valid within the context of this International Standard. The reference to such a specification within this International Standard does not give it any further status within ISO/IEC. In particular, it does not give the referenced specifications the status of an International Standard.

ISO 3166:2013, *Codes for the representation of names of countries and their subdivisions (Parts 1, 2 and 3)* - see [35][36][37]

ISO 4217:2015, *Codes for the representation of currencies and funds* - see [38]

ISO 8601:2019, *Data elements and interchange formats – Information interchange – Representation of dates and times* - see [32][33]

ISO/IEC 9594-8:2017, *Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks* - see [43]

ISO 14721:2012, *Space data and information transfer systems – Open archival information system (OAIS) – Reference model* - see [34]

ISO/IEC 14776-414:2009, *SCSI Architecture Model - 4 (SAM-4)* - see [29]

ISO/IEC 17788:2014, *Information technology – Cloud computing – Overview and vocabulary* - see [31]

ISO/IEC 20648, *TLS specification for storage systems* - see [39]

ISO/IEC 27040:2015, *Information technology – Security techniques – Storage security* - see [30]

ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards, 6th edition, 2011* - see [28]

IEEE 1003.1-2017, *IEEE Standard for Information Technology–Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7* - see [41]

RFC 1867, *Form-based File Upload in HTML* - see [20]

RFC 2045, *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies* - see [9]

RFC 2046, *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types* - see [10]

RFC 2119, *Key Words for Use in RFCs to Indicate Requirement Levels* - see [3]

RFC 2578, *Structure of Management Information Version 2 (SMIv2)* - see [21]

RFC 2616, *Hypertext Transfer Protocol – HTTP/1.1* - see [23]

RFC 2617, *HTTP Authentication: Basic and Digest Access Authentication* - see [8]

RFC 3280, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile* - see [13]

RFC 3530, *Network File System (NFS) Version 4 Protocol* - see [1]

RFC 7143, *Internet Small Computer System Interface (iSCSI) Protocol (Consolidated)* - see [4]

RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax* - see [2]

457 RFC 4627, *The Application/JSON Media Type for JavaScript Object Notation (JSON)* - see [5]
458 RFC 4648, *The Base16, Base32, and Base64 Data Encodings* - see [19]
459 RFC 4918, *HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)* - see [6]
460 RFC 5246, *The Transport Layer Security (TLS) Protocol Version 1.2* - see [25]
461 RFC 6068, *The 'mailto' URI Scheme* - see [27]
462 RFC 5652, *Cryptographic Message Syntac (CMS)* - see [12]
463 RFC 6208, *Cloud Data Management Interface (CDMI) Media Types* - see [26]
464 RFC 6839, *Additional Media Type Structured Syntax Suffixes* - see [11]
465 RFC 7515, *JSON Web Signatures* - see [17]
466 RFC 7516, *JSON Web Encryption* - see [18]
467 RFC 7518, *JSON Web Algorithms* - see [15]
468 RFC 8446, *The Transport Layer Security (TLS) Protocol Version 1.3* - see [24]
469 SNIA TLS, *TLS Specification for Storage Systems, version 1.1.0* - see [42]

Clause 3

Terms, acronyms, and definitions

For the purposes of this document, the terms and definitions given in Rec. ITU-T Y.3500 | ISO/IEC 17788:2014 and the following apply.

3.1

Access Control List (ACL)
a persistent list, commonly composed of Access Control Entries (ACEs), that enumerates the rights of principals (users and groups) to access resources

3.2

API
Application Programming Interface

3.3

CDMI™
Cloud Data Management Interface

3.4

CDMI capabilities
an object that describes what operations are supported for a given cloud or cloud object

The mimetype for this object is `application/cdmi-capability`.

3.5

CDMI container
an object that stores zero or more children objects and associated metadata

The mimetype for this object is `application/cdmi-container`.

3.6

CDMI data object
an object that stores an array of bytes (value) and associated metadata

The mimetype for this object is `application/cdmi-object`.

3.7

CDMI domain
an object that stores zero or more children domains and associated metadata describing object administrative ownership

The mimetype for this object is `application/cdmi-domain`.

3.8

CDMI object
one of CDMI capabilities, CDMI container, CDMI data object, CDMI domain, or CDMI queue

3.9

CDMI queue
an object that stores a first-in, first-out set of values and associated metadata

The mimetype for this object is `application/cdmi-queue`.

3.10

CIFS
Common Internet File System (See SMB)

3.11

cloud storage
See Data storage as a Service

3.12

CRC
cyclic redundancy check

3.13

current data object version
the most recent version of a version-enabled data object

3.14

data object version
either the current data object version or an historical data object version

3.15

Data Storage as a Service (DSaaS)
delivery of appropriately configured virtual storage and related data services over a network, based on a request for a given service level

3.16

delegated access control (DAC)
the process of delegating an access control decision to a third party

3.17

delegated access control provider (DAC provider)
a third-party system that is capable of making access control decisions

3.18

delegated access control request (DAC request)
a request made to a DAC provider for an access control decision

3.19

delegated access control response (DAC response)
a response from a DAC provider indicating the result of a request for an access control decision

3.20

domain
a shared user authorization database that contains users, groups, and their security policies and associated accounting information

Each CDMI object belongs to a single domain, and each domain provides user mapping and accounting information.

3.21

eventual consistency
a behavior of transactional systems that does not provide immediate consistency guarantees to provide enhanced system availability and tolerance to network partitioning

3.22

FC
Fibre Channel

3.23

FCoE
Fibre Channel over Ethernet

3.24

historical data object version
a non-current state of a version-enabled data object

3.25

HTTP
HyperText Transfer Protocol

3.26

Infrastructure as a Service (IaaS)
delivery over a network of an appropriately configured virtual computing environment, based on a request for a given service level

Typically, IaaS is either self-provisioned or provisionless and is billed based on consumption.

3.27

intermediary CDMI server
a CDMI server that is capable of forwarding DAC requests and responses

3.28

iSCSI
Internet Small Computer Systems Interface (see RFC 7143 [4])

3.29

JOSE
JavaScript Object Signing and Encryption

3.30

JWA
JSON Web Algorithm

3.31

JWE
JSON Web Encryption

3.32

JWS
JSON Web Signing

3.33

JSON
JavaScript Object Notation

3.34

LDAP
Lightweight Directory Access Protocol

3.35

LUN
Logical Unit Number (see *ISO/IEC 14776-414*)

3.36

metadata
data about other data (see [34])

3.37

MIME
Multipurpose Internet Mail Extensions (see RFC 2045 [9])

3.38

NFS
Network File System (see RFC 3530 [1])

3.39

object
an entity that has an object ID, has a unique URI, and contains state

Types of CDMI objects include data objects, container objects, capability objects, domain objects, and queue objects.

3.40

object identifier
a globally-unique value assigned at creation time to identify an object

3.41

OCCI
Open Cloud Computing Interface (see [40])

3.42

Platform as a Service (PaaS)
delivery over a network of a virtualized programming environment, consisting of an application deployment stack based on a virtual computing environment

Typically, PaaS is based on IaaS, is either self-provisioned or provisionless, and is billed based on consumption.

3.43

POSIX
Portable Operating System Interface (see *IEEE Std 1003.1*)

3.44

private cloud
delivery of SaaS, PaaS, IaaS, and/or DaaS to a restricted set of customers, usually within a single organization

Private clouds are created due to issues of trust.

3.45

public cloud
delivery of SaaS, PaaS, IaaS, and/or DaaS to, in principle, a relatively unrestricted set of customers

3.46

Representational State Transfer (REST)
a specific set of principles for defining, addressing, and interacting with resources addressable by URIs (see [7])

3.47

RPO
recovery point objective

3.48

RTO
recovery time objective

3.49

service level
performance targets for a service

3.50

Server Message Block
A network file system access protocol designed primarily used by Windows clients to communicate file access requests to Windows servers. (Also see CIFS)

3.51

SNMP
Simple Network Management Protocol

3.52

Software as a Service (SaaS)
delivery over a network, on demand, of the use of an application

technology that allocates the physical capacity of a volume or file system as applications write data, rather than pre-allocating all the physical capacity at the time of provisioning.

3.53

Uniform Resource Identifier (URI)
compact sequence of characters that identifies an abstract or physical resource (see RFC 3986 [2])

3.54

version-enabled data object
a CDMI data object with versioning enabled

718 **3.55**

719 **virtualization**
720 presentation of resources as if they are physical, when in fact, they are decoupled from the underlying physical resources
721

722 **3.56**

723 **WebDAV**
724 Web Distributed Authoring and Versioning (see RFC 4918 [6])
725

Clause 4

Conventions

4.1 Interface format

Each interface description has nine components, as described in Table 2.

Table 2: Interface format

Component	Description
Synopsis	The GET, PUT, POST, PATCH, and DELETE semantics
Delayed completion	For long-running operations, a description of behavior when the operation does not immediately complete
Capabilities	A description of the supported operations
Request headers	The request headers, such as Accept, Authorization, Content-Length, Content-Type
Request message body	A description of the message body contents
Response headers	The response headers, such as Content-Length, Content-Type
Response message body	A description of the message body contents
Response Status	A list of HTTP status codes
Example	An example of the operation

4.2 Typographical conventions

All code text and HTTP status codes are shown in a fixed-width font.

API resquests also include a prefix to indicate the source of the request or reply:

- Client-initiated requests are prefixed with '–>'
- Server-initiated replies are prefixed with '<–'

An example is included below:

```
--> PUT /cdmi/2.0.0/MyContainer/MyDataObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-object
--> Content-Type: application/cdmi-object
-->
--> {
-->   "mimetype" : "text/plain",
-->   "metadata" : {
-->       ...
-->   },
-->  "value" : "This is the Value of this Data Object"
--> }

<-- HTTP/1.1 202 Accepted
<-- Location: https://cloud.example.com/cdmi/2.0.0/MyContainer/MyDataObject.txt
```

Similarly, HTTP status codes are shown in a fixed-width font, as in:

Requesting an optional field that is not present shall result in an HTTP status code of `404 Not Found`.

4.3 Request and response body requirements

In request and response body tables, the Requirement column contains one of the following three values:

- Mandatory. The value specified in this row shall be provided.
- Conditional. If the conditions specified in the Description cell is met, the value specified in this row shall be provided. Otherwise, it may be provided unless the Description specifically prohibits it, in which case it shall not be provided.
- Optional. The value specified in this row may be provided.

4.4 Key Word requirements

In this International Standard, the key words in Table 3 shall be interpreted as described in ISO/IEC Directives, Part 2.
Table 2 — Key word requirements

Table 3: Key word requirements

Key words	Denotes	Description	Equivalent expressions (for exception cases only)
shall	requirement	An action that is unconditionally required. • Do not use must as an alternative to shall. • To express a direct instruction, for example, when referring to steps to be taken in a test method, use the imperative mood in English. • EXAMPLE: Switch on the recorder.	• is to • is required to • it is required that • has to • only … is permitted • it is necessary
shall not	requirement	An action that is unconditionally prohibited. Do not use may not instead of shall not to express a prohibition.	• is not allowed [permitted] [acceptable] [permissible] • is required to be not • is required that … be not • is not to be
should	recommendation	An action that is recommended when choosing among several possibilities, or an action that is preferred but not necessarily required.	• it is recommended that • ought to
should not	recommendation	An action or certain possibility or course of action that is deprecated but not prohibited.	• it is not recommended that • ought not to
may	permission	An action that indicates what is allowed within the limits of the document. Do not use possible or can in this context. May signifies permission expressed by the document, whereas can refers to the ability of a user of the document or to a possibility open to him or her.	• is permitted • is allowed • is permissible
need not	permission	An action that indicates what is not required within the limits of the document. Do not use impossible in this context.	• it is not required that • no … is required

Clause 5

Overview of Cloud Storage

5.1 Overview

5.1.1 General Context

When discussing cloud storage and standards, it is important to distinguish the various resources that are being offered as services. These resources are exposed to clients as functional interfaces (i.e., data paths) and are managed by management interfaces (i.e., control paths). This International Standard explores the various types of interfaces that are part of cloud services today and shows how they are related. This International Standard defines a model for the interfaces that can be mapped to the various cloud services and a model that forms the basis for cloud storage interfaces into the future.

Another important concept in this International Standard is that of metadata. When managing large amounts of data with differing requirements, metadata is a convenient mechanism to express those requirements in such a way that underlying data services can differentiate their treatment of the data to meet those requirements.

The appeal of cloud storage is due to some of the same attributes that define other cloud services: pay as you go, the illusion of infinite capacity (elasticity), and the simplicity of use/management. It is therefore important that any interface for cloud storage support these attributes, while allowing for a multitude of business use cases.

5.1.2 What is Cloud Storage?

The use of the term cloud in describing these new models arose from architecture drawings that typically used a cloud as the icon for a network. The cloud represents any-to-any network connectivity in an abstract way. In this abstraction, the network connectivity in the cloud is represented without concern for how it is made to happen.

The cloud abstraction of complexity produces a simple base on which other features can be built. The general cloud model extends this base by adding a pool of resources. An important part of the cloud model is the concept of a pool of resources that is drawn from, on demand, in small increments. A relatively recent innovation that has made this possible is virtualization.

Thus, cloud storage is simply the delivery of virtualized storage on demand. The formal term that is used for this is Data storage as a Service (DaaS).

5.1.3 Data Storage as a Service

By abstracting data storage behind a set of service interfaces and delivering it on demand, a wide range of actual cloud services and implementations are possible. The only type of storage that is excluded from this definition is that which is delivered in fixed-capacity increments instead of that which is based on demand.

An important part of any DaaS system is the support of legacy clients. Support is accommodated with existing standard protocols such as iSCSI (and others) for block network storage and SMB/NFS or WebDAV for file network storage, as shown in Fig. 1.

The difference between purchasing a dedicated appliance or purchasing cloud storage is not the functional interface, but the fact that the storage is delivered on demand. Customers pay for either what they actually use or what they have

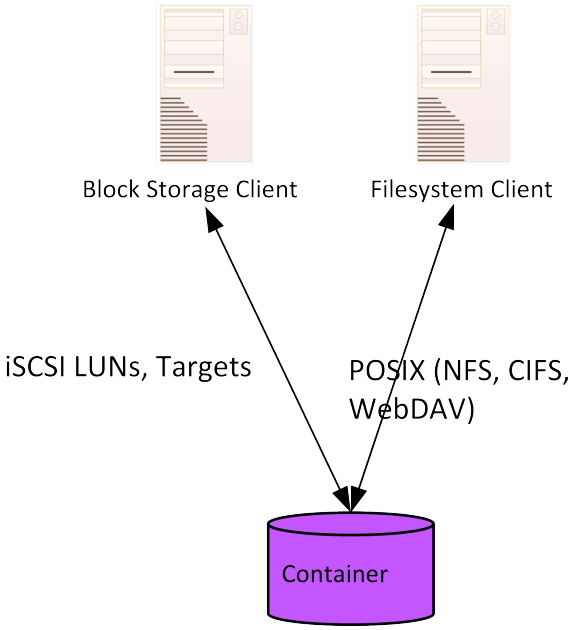


Fig. 1: Existing data storage interface standards

allocated for use. For block storage, a Logical Unit Number (LUN), or virtual volume, is the granularity of allocation. For file protocols, a file system is the unit of granularity. In either case, the actual storage space may be thin-provisioned and billed for based on actual usage. Data services, such as compression and deduplication, can be used to further reduce the actual space consumed.

Managing this storage is typically done out of band for these standard data storage interfaces, either through an API, or more commonly, through an administrative browser-based user interface. This out-of-band interface can be used to invoke other data services as well (e.g., snapshots or cloning).

In this model, the underlying storage space that has been exposed by the out-of-band interfaces is abstracted and exposed using the notion of a container. A container is not only a useful abstraction for storage space, but also serves as a grouping of the data stored in it and a point of control for applying data services in the aggregate.

Each data object is created, retrieved, updated, and deleted as a separate resource. In this type of interface, a container, if used, is a simple grouping of data objects for convenience. Nothing prevents the concept of containers from being hierarchical, although any given implementation might support only a single level (see Fig. 2).

5.1.4 Data management for cloud storage

Many of the initial implementations of cloud storage focused on a kind of best effort quality of storage service and ignored most other types of data services. To address the needs of enterprise applications with cloud storage, however, there is an increasing need to offer better quality of service and to deploy additional data services.

Cloud storage can lose its abstraction and simplicity benefits if new data services that require complex management are added. Cloud storage customers are likely to resist new demands on their time (e.g., setting up backup schedules through dedicated interfaces, deploying data services individually for stored objects).

By supporting metadata in a cloud storage interface and prescribing how the storage system and data system metadata is interpreted to meet the requirements of the data, the simplicity required by the cloud storage model can be maintained while still addressing the requirements of enterprise applications and their data.

User metadata is retained by the cloud and can be used to find the data objects and containers by performing a query for specific metadata values. The schema for this metadata may be determined by each application, domain, or user. For more information on support for user metadata, see 16.5.

Storage system metadata is produced/interpreted by the cloud service provider and basic storage functions (e.g., modification and access statistics, access control). For more information on support for storage system metadata, see 16.2.

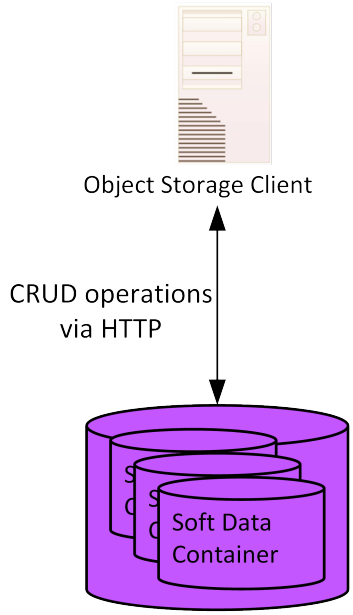


Fig. 2: Storage interfaces for object storage client data

Data system metadata is interpreted by the cloud service provider as data requirements that control the operation of underlying data services for that data. Depending on the level of granularity supported by the cloud, data system metadata may apply to an aggregation of data objects in a container or to individual data objects, if the cloud service provider supports this level of granularity. For more information on support for data system metadata, see 16.3.

5.1.5 Data and container management

There is no reason that managing data and managing containers should involve different interfaces. Therefore, the use of metadata is extended from applying to individual objects to applying to containers of objects as well. Thus, any data placed into a container inherits the data system metadata of the container into which it was placed. When creating a new container within an existing container, the new container would similarly inherit the metadata settings of its parent's data system metadata. After an object is created, the data system metadata may be overridden at the container or individual object level, as desired.

Even if the provided interface does not support setting metadata on individual objects, metadata can still be applied to the containers. In such a case, the interface does not provide a mechanism to override metadata that an individual object inherits from its parent container. For file-based interfaces that support extended attributes (e.g., SMB, NFSv4), these extended attributes may be used to specify the data system metadata to override that specified for the container.

5.2 Reference model for cloud storage interfaces

The cloud storage reference model is shown in Fig. 3.

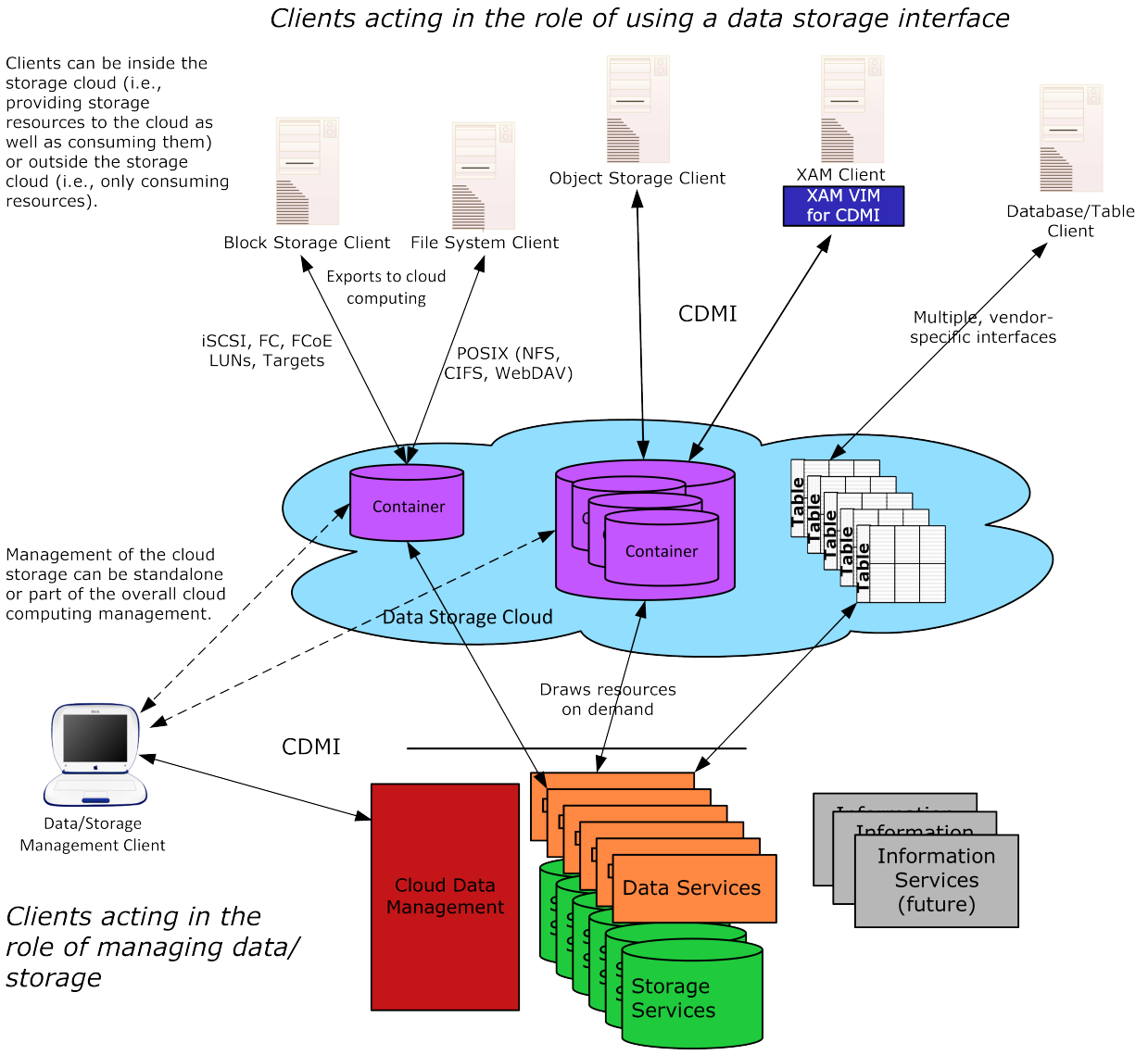


Fig. 3: Cloud storage reference model

This model shows multiple types of cloud data storage interfaces that are able to support both legacy and new applications. All of the interfaces allow storage to be provided on demand, drawn from a pool of resources. The storage capacity is drawn from a pool of storage capacity provided by storage services. The data services are applied to individual objects, as determined by the data system metadata. Metadata specifies the data requirements on the basis of individual objects or for groups of objects (containers).

5.3 Cloud data management interface

The Cloud Data Management Interface (CDMI™) shown in Fig. 3 may be used to create, retrieve, update, and delete objects in a cloud. The features of the CDMI include functions that:

- allow clients to discover the capabilities available by the cloud service provider,
- manage containers and the data that is placed in them, and
- allow metadata to be associated with containers and the objects they contain.

This International Standard divides operations into two types: those that use a CDMI content type in the HTTP body and those that do not. While much of the same data is available via both types, providing both allows for CDMI-aware clients and non-CDMI-aware clients to interact with a CDMI provider.

CDMI can also be used by administrative and management applications to manage containers, domains, security access, and monitoring/billing information, even for storage that is functionally accessible by legacy or proprietary protocols. The capabilities of the underlying storage and data services are exposed so that clients can understand what services the cloud service provider provides.

Conformant cloud service providers may support a subset of the CDMI, as long as they expose the limitations in the capabilities reported via the interface.

This International Standard uses RESTful principles in the interface design where possible (see [7]).

CDMI defines both a means to manage the data as well as a means to store and retrieve the data. The means by which the storage and retrieval of data is achieved is termed a data path. The means by which the data is managed is termed a control path. CDMI specifies both a data path and control path interface.

CDMI does not need to be used as the only data path and is able to manage cloud storage properties for any data path interface (e.g., standardized or vendor specific).

Container metadata is used to configure the data requirements of the storage provided through the exported protocol (e.g., block protocol or file protocol) that the container exposes. When an implementation is based on an underlying file system to store data for a block protocol (e.g., iSCSI), the CDMI container provides a useful abstraction for representing the data system metadata for the data and the structures that govern the exported protocols.

A cloud service may also support domains that allow administrative ownership to be associated with stored objects. Domains allow this International Standard to (among other things):

- determine how user credentials are mapped to principals used in an Access Control List (ACL),
- allow granting of special cloud-related privileges, and
- allow delegation to external user authorization systems (e.g., LDAP or Active Directory).

Domains may also be hierarchical, allowing for corporate domains with multiple children domains for departments or individuals. The domain concept is also used to aggregate usage data that is used to bill, meter, and monitor cloud use.

Finally, capabilities allow a client to discover the capabilities of a CDMI implementation. Requirements throughout this International Standard shall be understood in the context of CDMI capabilities. Mandatory requirements on functionality that is conditioned on a CDMI capability shall not be interpreted to require implementation of that capability, but rather shall be interpreted to apply only to implementations that support the functionality required by that capability.

For example, in 5.3.3, this International Standard states, “Every cloud storage system shall allow object ID-based access to stored objects.” This requirement shall be understood in the context that access by object ID is predicated on the presence of the `cdmi_object_access_by_ID` capability.

5.3.1 Object model for CDMI

The model for CDMI is shown in Fig. 4.

The five types of resources defined are shown in Table 4. The content type in any given operation is specific to each type of resource.

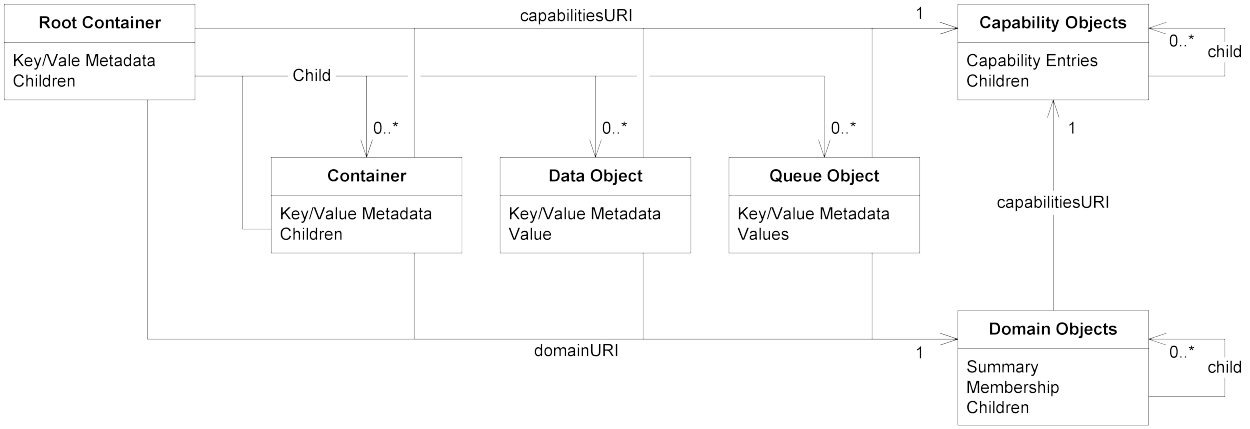


Fig. 4: CDMI object model

Table 4: Types of resources in the CDMI object model

Resource type	Description	Reference
Data objects	Data objects are used to store data and associated metadata, and provide functionality similar to files in a file system.	See clause 8.
Container objects	Container objects have zero or more children objects, and store metadata associated with the container as a whole. Container objects do not store data directly. They provide functionality similar to directories in a file system.	See clause 9.
Domain objects	Domain objects represent administrative groupings for user authentication and accounting purposes.	See clause 10.
Queue objects	Queue objects store zero or more pieces of data, and store metadata associated with the queue as a whole. Enqueued values are accessed in a first-in-first-out manner.	See clause 11.
Capability objects	Capability objects describe the functionality implemented by a CDMI server and are used by a client to discover supported functionality.	See clause 12.

For data storage operations, the client of the interface only needs to know about container objects and data objects. All data path implementations are required to support at least one level of containers (see 5.1.5). Using the CDMI object model (see Fig. 4), the client can send a PUT via CDMI (see Fig. 3) to the new container URI and create a new container with the specified name. Container metadata are optional and are expressed as a series of name-value pairs. After a container is created, a client can send a PUT to create a data object within the newly created container.

Queue objects are also defined (see Fig. 4) and provide in-order-first in-first-out access to enqueued objects. More information on queues can be found in clause 11.

CDMI defines two namespaces that can be used to access stored objects, a flat object ID namespace and a hierarchical path-based namespace. Support for objects accessed by object ID is indicated by the system-wide capability `cdmi_object_access_by_ID`, and support for objects accessed by hierarchical path is indicated by the container capability `cdmi_create_dataobject` found on the root container (and any subcontainers).

Objects are created by ID by performing an HTTP POST against a special URI, designated as `/cdmi_objectid/` (see 9.7). Subsequent to creation, objects are modified by performing PUTs using the object ID assigned by the CDMI server, using the "`/cdmi_objectid/`" URI (see 8.5). The same URI is used to retrieve and delete objects by ID.

Objects are created by name by performing an HTTP PUT to the desired path URI (see 8.3). Subsequent to creation, objects are modified by performing PUTs using the object path specified by the client (see 8.5). The same URI is used to retrieve and delete objects by path.

CDMI defines mechanisms so that objects having only an object ID can be assigned a path location within the hierarchical namespace, and so that objects having both an object ID and path can have their path dropped, such that the object only has an object ID. This function is accomplished by using a "move" modifier to a PUT or POST operation, as shown in Fig. 5.

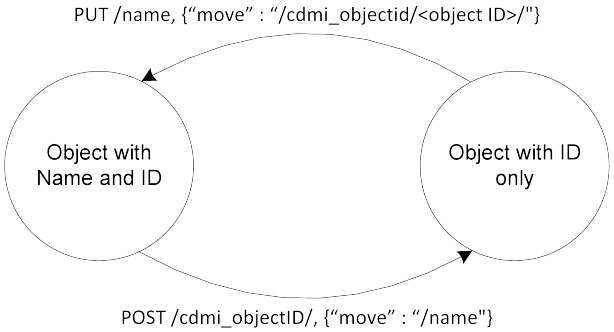


Fig. 5: Object transitions between named and ID-only

5.3.2 CDMI metadata

CDMI uses many different types of metadata, including HTTP metadata, data system metadata, user metadata, and storage system metadata.

HTTP metadata is metadata that is related to the use of the HTTP protocol (e.g., `Content-Length`, `Content-Type`, etc.). HTTP metadata is not specifically related to this International Standard but needs to be discussed to explain how CDMI uses the HTTP standard.

CDMI data system metadata, user metadata, and storage system metadata is defined in the form of name-value pairs. Vendor-defined data system metadata and storage system metadata names shall begin with the reverse domain name of the vendor.

Data system metadata is metadata that is specified by a CDMI client and is a component of objects. Data system metadata abstractly specifies the data requirements associated with data services that are deployed in the cloud storage system.

User metadata consists of client-defined JSON strings, arrays, and objects that are stored in the metadata field. The namespace used for user metadata names is self-administered (e.g., using the reverse domain name), and user metadata names shall not begin with the prefix “`cdmi_`”.

Storage system metadata is metadata that is generated by the storage services in the system (e.g., creation time, size) to provide useful information to a CDMI client.

The matrix of the creation and consumption of storage system metadata is shown in Table 5.

Table 5: Creation/consumption of storage system metadata

	Created by user	Created By system
Consumed by user	User metadata	Storage system metadata
Consumed by system	Data system metadata	N/A

5.3.3 CDMI object IDs

Every object stored within a CDMI-compliant system shall have a globally unique object identifier (ID) assigned at creation time. The CDMI object ID is a string with requirements for how it is generated and how it obtains its uniqueness. Each cloud service that implements CDMI shall generate these identifiers such that the probability of conflicting with identifiers generated by other CDMI Servers and the probability of generating an identifier that has already been used is effectively zero.

Every cloud storage system shall allow object ID-based access to stored objects by allowing the object’s ID to be appended to the root URI (see 5.5.5). If the data object “`MyDataObject.txt`”, stored in the root container “`/`” with a root path of “`/cdmi/2.0.0/`”, has an object ID of “`00006FFD001001CCE3B2B4F602032653`”, the following pair of URIs access the same data object:

- `https://cloud.example.com/cdmi/2.0.0/MyDataObject.txt`
- `https://cloud.example.com/cdmi/2.0.0/cdmi_objectid/00006FFD001001CCE3B2B4F602032653`

If containers are supported, they shall also be accessible by object ID. If the container “MyContainer”, stored in the root container “/” with a root path of “/cdmi/2.0.0/”, has an object ID of “00006FFD0010AA33D8CEF9711E0835CA”, the following pairs of URIs access the same object:

- https://cloud.example.com/cdmi/2.0.0/MyContainer/
- https://cloud.example.com/cdmi/2.0.0/cdmi_objectid/00006FFD0010AA33D8CEF9711E0835CA/

- https://cloud.example.com/cdmi/2.0.0/MyContainer/MyDataObject.txt
- https://cloud.example.com/cdmi/2.0.0/cdmi_objectid/00006FFD0010AA33D8CEF9711E0835CA/MyDataObject.txt

5.3.4 CDMI object ID format

The CDMI Server shall create the object ID, which identifies an object. The object ID shall be globally unique and shall conform to the format defined in Table 6. The native format of an object ID is a variable-length byte sequence and shall be a maximum length of 40 bytes. A client should treat object IDs as opaque byte strings. However, the object ID format is defined such that its integrity may be validated, and independent CDMI Servers may assign unique object ID values independently.

Table 6: Object ID format

0	1	2	3	4	5	6	7	8	9	10	…	38	39
Reserved (zero)	Enterprise Number			Reserved (zero)	Length	CRC		Opaque Data					

The fields shown in Table 6 are defined as follows:

- The reserved bytes shall be set to zero.
- The Enterprise Number field shall be the SNMP enterprise number of the offering organization that developed the system that created the object ID, in network byte order. See RFC 2578 [21] and https://www.iana.org/assignments/enterprise-numbers. 0 is a reserved value.
- The byte at offset 5 shall contain the full length of the object ID, in bytes.
- The CRC field shall contain a 2-byte (16-bit) CRC in network byte order. The CRC field enables the object ID to be validated for integrity. The CRC field shall be generated by running the CRC algorithm across all bytes of the object ID, as defined by the Length field, with the CRC field set to zero. The CRC function shall have the following fields:
  - Name : “CRC-16”,
  - Width : 16,
  - Poly : 0x8005,
  - Init : 0x0000,
  - RefIn : True,
  - RefOut : True,
  - XorOut : 0x0000, and
  - Check : 0xBB3D.

  This function defines a 16-bit CRC with polynomial 0x8005, reflected input, and reflected output.
- Opaque data in each object ID shall be unique for a given Enterprise Number.

The native format for an object ID is binary. When necessary, such as when included in URIs and JSON strings, the object ID textual representation shall be encoded using Base16 encoding rules described in RFC 4648 [19] and shall be case insensitive.

5.4 Security

5.4.1 Security objectives

Security, in the context of CDMI, refers to the protective measures employed in managing and accessing data and storage. The specific objectives to be addressed by security include providing a mechanism that:

- assures that the communications between a CDMI client and server cannot be read or modified by a third party;
- allows CDMI clients and servers to assure their identity;
- allows control of the actions a CDMI client is permitted to perform on a CDMI server;
- allows records to be generated for actions performed by a CDMI client on a CDMI server;
- protects data at rest;
- eliminates data in a controlled manner; and
- discovers the security capabilities of of a particular implementation.

Security measures within CDMI are summarized as:

- transport security,
- user and entity authentication,
- authorization and access controls,
- data integrity,
- data and media sanitization,
- data retention,
- protections against malware,
- data at-rest encryption, and
- security capabilities.

With the exception of both the transport security and the security capabilities, which shall be implmented, the security measures can vary significantly from implementation to implementation.

When security is a concern, the CDMI client should begin with a series of security capability lookups (see 12.2.7 to determine the exact nature of the security features that are available. Based on the values of these capabilities, a risk-based decision should be made as to whether the CDMI server should be used. This is particularly true when the data to be stored in the cloud storage is sensitive or regulated in a way that requires stored data to be protected (e.g., encrypted) or handled in a particular manner (e.g., full accountability and traceability of management and access).

5.4.2 HTTP security

HTTP is the mandatory transport mechanism for this version of CDMI. It is important to note that HTTP, by itself, offers no confidentiality or integrity protections. As CDMI is built on top of HTTP, HTTP over Transport Layer Security (TLS) (i.e., HTTPS) is the mechanism that is used to secure the communications between CDMI clients and servers.

To ensure both security and interoperability, all CDMI implementations:

- shall implement the TLS protocol as described in the latest version of the “SNIA TLS Specification for Storage Systems” [42]; with a six-month transition period for implementations. The TLS specification is updated when new vulnerabilities are found, and CDMI implementations shall support the latest specification within six months of its publication announcement;
- shall support both HTTP over TLS and HTTP without TLS; and
- shall allow HTTP without TLS to be disabled.

When TLS is used to secure HTTP, the client and server typically perform some form of entity authentication. However, the specific nature of this entity authentication depends on the cipher suite negotiated; a cipher suite specifies the encryption algorithm and digest algorithm to use on a TLS connection. A very common scenario involves using server-side certificates, which the client trusts, as the basis for unidirectional entity authentication. It is possible that mutual authentication involving both client-side and server-side certificates are required.

5.4.3 Client Authentication

A CDMI client shall comply with all security requirements for HTTP that apply to clients.

CDMI clients shall be responsible for initiating user authentication for each CDMI operation that is performed. The CDMI server functions as the authenticator and receives and validates authentication credentials from the client.

RFC 2616 [23] and RFC 2617 [8] define requirements for HTTP authentication, which generally starts with an HTTP client request. If the client request does not include an `Authorization` header and authentication is required, the server responds with an HTTP status code of `401 Unauthorized` and a `WWW-Authenticate` response header. The HTTP client shall then respond with the appropriate `Authorization` header in a subsequent request. The format of the `WWW-Authenticate` and `Authorization` headers varies depending on the type of authentication required.

- HTTP basic authentication involves sending the user name and password in the clear, and it should only be used on a secure network or in conjunction with TLS.
- HTTP digest authentication sends a secure digest of the user name and password (and other information such as a nonce value), and can be used on an insecure network without TLS.
- HTTP status codes of `401 Unauthorized` should not include a choice of authentication.
- HTTP basic authentication and/or HTTP digest authentication should be implemented.
- Authentication credentials used with one type of HTTP authentication (i.e., basic or digest) should never be subsequently used with the other type of HTTP authentication.

Once a user is authenticated, the provided principal name shall be mapped by the CDMI domain to a domain user (or used directly as the ACE "`who`" if domains are not supported). This mapping is then used to determine authorization.

A CDMI server typically relies on an authentication service (local and/or external) to validate client credentials. Differing authentication schemes may be supported, including host-based authentication, Kerberos, PKI, or other; the authentication service is beyond the scope of this International Standard.

5.4.4 Use of TLS and HTTP

Recommendations for using HTTP and TLS are as follows:

- A client connecting to a CMDI server using TLS should use TCP port 443, and a client connecting without TLS should use TCP port 80.
- A client that fails to connect to a CDMI server on port 443 should retry without TLS on TCP port 80 if their security policy allows it.
- Servers may respond to HTTP requests on port 80 with an HTTP REDIRECT to the appropriate TLS URI (using port 443). Clients should honor such redirects in this situation.

5.4.5 Further information

For further information pertaining to storage security techniques, see the latest version of ISO 20648.

5.5 Required HTTP support

5.5.1 RFC 2616 support requirements

A conformant implementation of CDMI shall also be a conformant implementation of RFC 2616 [23] (i.e., HTTP 1.1). The subclauses below list the sections of RFC 2616 [23] that shall be supported; however, this list is not comprehensive.

5.5.2 Content-Type negotiation

For CDMI operations, media types for CDMI objects are used as defined in RFC 6208 [26]. All CDMI representations follow the rules established for `application/json` as defined in RFC 4627 [5]. The use of the CDMI media types with the `+json` suffix shall be supported as defined in RFC 6839 [11].

A client can optionally supply an HTTP `Accept` header, as per section 14.1 of RFC 2616 [23]. If a client is restricting the response to a specific CDMI media type, the corresponding media type shall be specified in the `Accept` header. Otherwise, the `Accept` header can contain "`*/*`" or a list of media types, or it may be omitted.

If a request body is present, the client shall include a Content-Type header, as per section 14.17 of RFC 2616 [23]. If the client does not provide a Content-Type header when required or provides a media type in the Content-Type header that does not match with the existing resource media type, the server shall return an HTTP status code of `400 Bad Request`.

If a response body is present, the server shall provide a `Content-Type` header.

This International Standard may further qualify content negotiation (e.g., in 9.4, the absence of a `Content-Type` header has a specific meaning).

5.5.3 Range support

The server shall support HTTP `Range` headers and partial content responses (see Section 14.16 of RFC 2616 [23]).

The values of the `childrange`, `valuerange` and `queuerange` fields are formatted based on the HTTP `byte-range-resp-spec`, as defined in clause 14.16 of RFC 2616 [23].

5.5.4 URI escaping

Percent escaping of reserved characters specified in RFC 3986 [2] shall be applied to all text strings used in HTTP request URIs and HTTP header URIs. This includes user-supplied field names, metadata names, data object names, container object names, queue object names, and domain object names when used in HTTP request URIs and HTTP header URIs.

Field names and values shall not be escaped when stored and when sent in request body and response bodies.

A client retrieving a metadata item named "@user" from a container object with the name of "@MyContainer" would perform the following request and reply:

```
--> GET /cdmi/2.0.0/%40MyContainer/?objectName&metadata=%40user HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-container

<-- HTTP/1.1 200 OK
<-- Content-Type: application/cdmi-container
<--
<-- {
<--   "objectName": "@MyContainer/",
<--   "metadata": {
<--     "@user": "test",
<--     ...
<--   }
<-- }
```

5.5.5 Use of URIs

The format and syntax of URIs are defined by RFC 3986 [2].

This International Standard splits the RFC 3986 path into two parts: The “root path” and the “CDMI path”, as shown in Fig. 6. The URI containing only the root path is called the “root URI”.

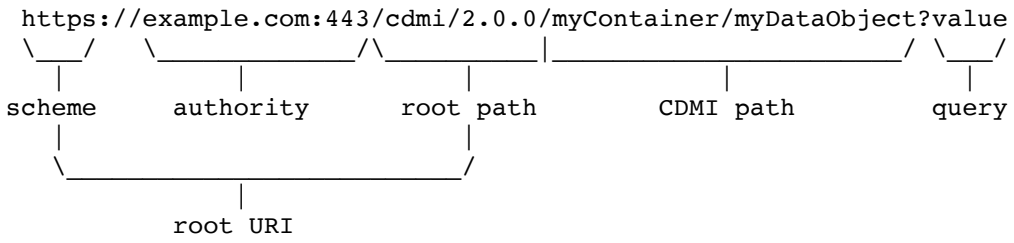


Fig. 6: CDMI URI Components

The container at the start of the CDMI path is the root container. For example, in Fig. 6, the root container is named “`myContainer`”.

All URIs in this International Standard are relative to the root URI unless otherwise noted. As a consequence, the algorithm used for calculating the resolved URI is as described in Section 5.2 of RFC 3986 [2]. Every CDMI client shall maintain one or more root URIs that each correspond to a root CDMI container on the CDMI server. Since all URIs to CDMI containers end in a trailing slash, all root URIs will end in a trailing slash.

This International Standard places no additional restrictions on root URIs beyond those specified for the “path component” in RFC 3986.

Industry conventions for RESTful APIs suggest root URIs end in “`/cdmi/<version>/`”, where <version> is in the form of “`<major>.<minor>.<micro>`”, where <major>, <minor> and <micro> are integers indicating the version of the CDMI interface specification. All examples in this specification use a root URI of `https://cloud.example.com/cdmi/2.0.0/`.

The properties of the root URI determine the `parentID` and `parentURI` fields of an root CDMI container:

- If the root path is “`/`”, the root container shall not include the `parentID` field and shall populate an empty string (“”) for the value of the `parentURI` field.
- If the root path is not “`/`” and the last entity in the root path is a CDMI container, the root container shall populate `parentID` field with the CDMI object ID of the CDMI container corresponding to the parent path entity, and shall populate the `parentURI` field with the URI of the parent path.
- If the root path is not “`/`” and the last entity in the root path is not a CDMI container, the root container shall not include the `parentID` field, and shall populate the `parentURI` field with the URI of the parent path.
- If the root path is not “`/`” and the last entry in the root path is not accessible via the scheme, root container may omit the `parentID` field and may populate `parentURI` field with an empty string (“”).

Table 7 shows how CDMI paths (relative URIs) are resolved with root URIs

1107
1108 Table 7: Relative URIs resolved against root URIs

Root URI	+ CDMI Path	=> Resolved URI
https://cloud.example.com/		https://cloud.example.com/
https://cloud.example.com/	/	https://cloud.example.com/
https://cloud.example.com/	myCDMIcontainer/testObject	https://cloud.example.com/myCDMIcontainer/testObject
https://cloud.example.com/	myCDMIcontainer/testObject	https://cloud.example.com/container/testObject
https://cloud.example.com/myNonCDMIentity/	myCDMIcontainer/testObject	https://cloud.example.com/myNonCDMIentity/myCDMIcontainer/testObject
https://cloud.example.com/myNonCDMIentity/	myCDMIcontainer/testObject	https://cloud.example.com/myCDMIcontainer/testObject
https://cloud.example.com/cdmi/2.0.0/	myCDMIcontainer/testObject	https://cloud.example.com/cdmi/2.0.0/myCDMIcontainer/testObject
https://cloud.example.com/cdmi/2.0.0/	myCDMIcontainer/testObject	https://cloud.example.com/myCDMIcontainer/testObject

1109 ### 5.5.6 Reserved characters

1110 The name of CDMI data objects, container objects, queue objects, domain objects and capability objects shall not
1111 contain the "/" or "?" characters, as these characters are reserved for delimiters.

5.6 Time representations

Unless otherwise specified, all date/time values are in the ISO 8601:2004 extended representation ("YYYY-MM-DDThh:mm:ss.ssssssZ"). The full precision shall be specified, the sub-second separator shall be a ".", the "Z" UTC zone indicator shall be included, and all timestamps shall be in UTC time zone. The "YYYY-MM-DDT24:00:00.000000Z" hour shall not be used, and instead, it shall be represented as "YYYY-MM-DDT00:00:00.000000Z".

Unless otherwise specified, all date/time intervals are in the ISO 8601:2004 start date/end date representation ("YYYY-MM-DDThh:mm:ss.ssssssZ/YYYY-MM-DDThh:mm:ss.ssssssZ"). The end date shall be equal to or later than the start date. The full precision shall be specified, the sub-second separator shall be a ".", the "Z" UTC zone indicator shall be included, and all timestamps shall be in UTC time zone. The "YYYY-MM-DDT24:00:00.000000Z hour shall not be used, and instead, it shall be represented as "YYYY-MM-DDT00:00:00.000000Z".

5.7 Backwards compatibility

CDMI client and server implementations shall implement the following measures to ensure backwards compability with earlier versions of this Interational Standard.

See the CDMI 1.1.1 Specification for details on backwards compatibility specific to the 1.x versions of CDMI.

5.7.1 Specification version detection

CDMI 2.x clients shall not include the `X-CDMI-Specification-Version` custom header. When a CDMI 2.x client performs an operation against a CDMI 1.x Server, the absence of this header shall result in an error response from the CDMI 1.x server. The client may use the presence of the `X-CDMI-Specification-Version` header in an error response as an indication to use CDMI 1.x (which mandates the use of this custom header), if supported.

CDMI 2.x servers may use the presence of the `X-CDMI-Specification-Version` custom header from a CDMI 1.x client as an indication to use CDMI 1.x, if supported.

5.7.2 JSON value transfer encoding

CDMI 2.x servers may support the “json” value transfer encoding. When a CDMI server supports both CDMI 2.x and CDMI 1.x, data objects with a value transfer encoding of json shall be made accessible to CDMI 1.x clients using a value transfer encoding of UTF-8, with the server adding in the required escaping.

5.8 Object references

Object references are URIs within the cloud storage namespace that redirect to another URI within the same or another cloud storage namespace. References are similar to soft links in a file system. The cloud does not guarantee that the referenced URI will be valid after the time of creation.

References are visible as children in a container and are distinguished from non-references in container children listings by the presence of a trailing "?" character added to the reference name. Performing an operation (with the exception of create or delete) to a reference URI will result in an HTTP status code of `302 Found`, with the HTTP `Location` header containing the absolute redirect destination URI that was specified at the time the reference was created. The reference's destination URI shall not be changed after a reference has been created.

To continue, when CDMI clients receive an HTTP status code of `302 Found`, they should retry the operation using the URI contained within the " Location" header.

A delete operation on a reference URI shall delete the reference. References cannot be updated. To update the destination of a redirect, the client shall first delete the reference and then create a new reference to the desired destination.

EXAMPLE 1: GET to a URI, where the URI is a reference:

```
--> GET /cdmi/2.0.0/MyContainer/MyDataObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-object

<-- HTTP/1.1 302 Found
<-- Location: https://cloud.example.com/cdmi/2.0.0/MyContainer/MyOtherDataObject.txt
```

References by object ID shall always redirect to a URI that ends with the same object ID as the request URI.

EXAMPLE 2: GET to an object ID URI, where the URI is a reference:

```
--> GET /cdmi/2.0.0/cdmi_objectid/00006FFD0010AA33D8CEF9711E0835CA HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-object

<-- HTTP/1.1 302 Found
<-- Location: https://archive.example.com/cdmi/2.0.0/cdmi_objectid/
↪00006FFD0010AA33D8CEF9711E0835CA
```

EXAMPLE 3: PUT to create a reference:

```
--> PUT /cdmi/2.0.0/MyContainer/MyDataObject.txt HTTP/1.1
--> Host: cloud.example.com Accept: application/cdmi-object
--> Content-Type: application/cdmi-object
-->
--> {
--> "reference": "https://cloud.example.com/cdmi/2.0.0/MyContainer/MyOtherDataObject.
↪txt"
--> }

<-- HTTP/1.1 201 Created
```

1156 EXAMPLE 4: POST to create a reference:

```
--> POST /cdmi/2.0.0/cdmi_objectid/ HTTP/1.1
--> Host: cloud.example.com Accept: application/cdmi-object
--> Content-Type: application/cdmi-object
-->
--> {
--> "reference": "https://cloud.example.com/cdmi/2.0.0/MyContainer/MyOtherDataObject.
↪txt"</P>
--> }

<-- HTTP/1.1 201 Created
<-- Location: https://cloud.example.com/cdmi/2.0.0/cdmi_objectid/
↪00007ED90010DF417BAD70A0C7F5CDDA
```

1157 EXAMPLE 5: DELETE to delete a reference:

```
--> DELETE /cdmi/2.0.0/MyContainer/MyDataObject.txt HTTP/1.1
--> Host: cloud.example.com

<-- HTTP/1.1 204 No Content
```

Part II

Basic Cloud Storage

Clause 6

Data Object Resource Operations using HTTP

6.1 Overview

Data objects are the fundamental storage components within CDMI™, and is analogous to files in a file system.

As CDMI builds on top of, and is compatible with, the HTTP standard (RFC 2616 [23]), this allows unmodified HTTP clients to communicate with a CDMI server. This also allows CDMI operations to coexist with other HTTP-based storage protocols, such as WebDAV, S3, and OpenStack Swift.

A CDMI server differentiates between HTTP and CDMI operations using the standard Content-Type and Accept headers. When CDMI MIME types defined in RFC 6208 [26] are used in these headers, this indicates that CDMI behaviors, as described in clause 8, are used in addition to the standard HTTP behaviors.

In CDMI 1.0.2, basic HTTP operations were described as “Non-CDMI” operations to distinguish them from operations using CDMI MIME types.

A CDMI implementation that supports data objects shall include support for basic data object HTTP operations corresponding with the CDMI capabilities that are published by the implementation. Capabilities allow a client to discover which operations (such as create, update, delete, etc.) are supported and are described in clause 9.

Ciphertext representation of encrypted objects are created, accessed, and updated by explicitelyspecifying a MIME type “`application/cms`” or “`application/jose+json`”. Otherwise, a plaintext representation is created, accessed, and updated. For more details on encrypted updates, see clause 23.

6.2 Create a data object using HTTP

6.2.1 Synopsis

The following HTTP PUT operation creates a new data object in the specified container:

- `PUT <root URI>/<ContainerName>/<DataObjectName>`

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<ContainerName>` is zero or more intermediate containers that already exist, with one slash (i.e., `"/"`) between each pair of container names.
- `<DataObjectName>` is the name specified for the data object to be created.

After it is created, the data object shall also be accessible at `<root URI>/cdmi_objectid/<objectID>`.

6.2.2 Capabilities

Capabilities that indicate which operations are supported are shown in Table 8.

Table 8: Capabilities - Create a CDMI data object using HTTP

Capability	Location	Description
`cdmi_create_dataobject`	Parent Container	Ability to create a new data object
`cdmi_create_value_range`	System Wide Capability	Ability to create a data object using a specified byte range

6.2.3 Request headers

The HTTP request headers for creating a CDMI data object using HTTP are shown in Table 9.

Table 9: Request headers - Create a CDMI data object using HTTP

Header	Type	Description	Requirement
`Content-Type`	Header string	The content type of the data to be stored as a data object. The value specified in this header shall be converted to lower case and stored in the `mimetype` field of the CDMI data object. • If the `Content-Type` header includes the charset parameter as defined in RFC 2616 [23] of "`utf-8` (e.g., "`;charset=utf-8`"), the `valuetransferencoding` field of the CDMI data object shall be set to "`utf-8`". Otherwise, the `valuetransferencoding` field of the CDMI data object shall be set to "`base64`". • If not specified, the `mimetype` field shall be set to "`application/octet-stream`".	Optional
`X-CDMI-Partial`	Header String	Indicates that the newly created object is part of a series of writes and has not yet been fully created. When set to "`true`", the `completionStatus` field shall be set to "`Processing`". `X-CDMI-Partial` works across CDMI and non-CDMI operations.	Optional
`Content-Range`	Header String	A valid ranges-specifier (see RFC 2616 [23] Section 14.35.1)	Optional

6.2.4 Request message body

The request message body contains the data to be stored in the value of the data object.

6.2.5 Response headers

No response headers are specified.

6.2.6 Response message body

No response message body fields are specified.

6.2.7 Response status

The HTTP status codes that occur when creating a data object using HTTP are described in Table 10.

Table 10: HTTP status codes - Create a data object using HTTP

HTTP Status	Description
`201 Created`	The new data object was created.
`400 Bad Request`	The request contains invalid parameters or field names.
`401 Unauthorized`	The authentication credentials are missing or invalid.
`403 Forbidden`	The client lacks the proper authorization to perform this request.
`404 Not Found`	The resource was not found at the specified URI.
`409 Conflict`	The operation conflicts with a non-CDMI access protocol lock or has caused a state transition error on the server.

6.2.8 Examples

EXAMPLE 1: PUT to the container URI the data object name and contents.

```
--> PUT /cdmi/2.0.0/MyContainer/MyDataObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: text/plain;charset=utf-8
--> Content-Length: 37
-->
--> This is the Value of this Data Object


 <-- HTTP/1.1 201 Created
```

EXAMPLE 2: Put to the container URI to create an encrypted object:

```
--> PUT /cdmi/2.0.0/MyContainer/MyEncryptedObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cms
--> Content-Length: 1425
-->
--> <CMS Encrypted Object>

<-- HTTP/1.1 201 Created
```

EXAMPLE 3: PUT to the container URI to create an encrypted object:

6.3 Read a data object using HTTP

6.3.1 Synopsis

The following HTTP GET operations read from an existing data object at the specified URI:

- `GET <root URI>/<ContainerName>/<DataObjectName>`
- `GET <root URI>/cdmi_objectid/<DataObjectID>`

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<ContainerName>` is zero or more intermediate containers.
- `<DataObjectName>` is the name of the data object to be read from.
- `<DataObjectID>` is the ID of the data object to be read from.

6.3.2 Capabilities

Capabilities that indicate which operations are supported are shown in Table 11.

Table 11: Capabilities - Read a CDMI data object using HTTP

Capability	Location	Description
`cdmi_read_value`	Data Object	Ability to read the value of an existing data object
`cdmi_read_value_range`	Data Object	Ability to read a sub-range of the value of an existing data object
`cdmi_object_access_by_ID`	System Wide Capability	Ability to access the object by ID

6.3.3 Request header

The HTTP request header for reading a CDMI data object using HTTP is shown in Table 12.

Table 12: Request header - Read a CDMI data object using HTTP

Header	Type	Description	Requirement
`Range`	Header string	A valid ranges-specifier (see RFC 2616 [23] Section 14.35.1)	Optional
`Accept`	Header string	"*/*" or a value as described in: 5.5.2. • If the object has a mimetype of "`application/cms`" or "`application/jose+json`", and the mimetype "`application/cms`" or "`application/jose+json`" is included in the Accept header mimetype, the CDMI server shall return the CMS or JOSE value in the response message body. • Otherwise, the decrypted plaintext shall be returned in the response message body, along with the encapsulated mimetype in the Content-Type response header. If decryption is not possible, an error result code shall be returned. (See clause 23 – Encrypted Objects) • If the Accept header mimetype list includes "`\*/\*`" before "`application/cms`" and/or "`application/jose+json`", the server will first try to return the decrypted plaintext, and shall return the CMS or JOSE value when decryption fails. • If the Accept header mimetype list excludes "`\*/\*`", decrypted plaintext shall only be returned if the encapsulated mimetype is included in the Accept header mimetype list.	Optional

6.3.4 Request message body

A request body shall not be provided.

6.3.5 Response headers

The HTTP response headers for reading a data object using HTTP are shown in Table 13.

Table 13: Response headers - Read a CDMI Data Object using HTTP

Header	Type	Description	Requirement
`Content-Type`	Header string	The content type returned shall be the mimetype field in the data object.	Mandatory
`Location`	Header string	The server shall respond with the URI that the reference redirects to if the object is a reference.	Conditional

6.3.6 Response message body

When reading a data object using HTTP, the following applies:

- The response message body shall be the contents of the data object's value field.
- When reading a value, zeros shall be returned for any gaps resulting from non-contiguous writes.

6.3.7 Response status

The HTTP status codes that occur when reading a data object using HTTP are described in Table 14.

1238
1239 Table 14: HTTP status codes - Read a CDMI data object using HTTP

HTTP Status	Description
200 OK	The data object content was returned in the response.
206 Partial Content	A requested range of the data object content was returned in the response.
302 Found	The resource is a reference to another resource.
400 Bad Request	The request contains invalid parameters or field names.
401 Unauthorized	The authentication credentials are missing or invalid.
403 Forbidden	The client lacks the proper authorization to perform this request.
404 Not Found	The resource was not found at the specified URI, or a requested field within the resource was not found.

1240 ### 6.3.8 Examples

1241 EXAMPLE 1: GET to the data object URI to read the value of the data object:

```
--> GET /cdmi/2.0.0/MyContainer/MyDataObject.txt HTTP/1.1
--> Host: cloud.example.com

<-- HTTP/1.1 200 OK
<-- Content-Type: text/plain
<-- Content-Length: 37
<--
<-- This is the value of this data object
```

1242 EXAMPLE 2: GET to the data object URI to read the first 11 bytes of the value of the data object:

```
--> GET /cdmi/2.0.0/MyContainer/MyDataObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Range: bytes=0-10

<-- HTTP/1.1 206 Partial Content
<-- Content-Type: text/plain
<-- Content-Range: bytes 0-10/37
<-- Content-Length: 11
<--
<-- This is the value of this data object
```

1243 EXAMPLE 3: GET to the data object URI to always return the ciphertext of an encrypted object:

```
--> GET /cdmi/2.0.0/MyContainer/MyEncryptedObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cms, application/jose+json

<-- HTTP/1.1 200 OK
<-- Content-Type: application/cms
<-- Content-Length: 1425
<--
<-- <CMS Encrypted Object>
```

1244 EXAMPLE 4: GET to the data object URI to read the plaintext of an encrypted object, if possible; otherwise, get the
1245 ciphertext:

```
--> GET /cdmi/2.0.0/MyContainer/MyEncryptedObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Accept: */*, application/cms, application/jose+json
--> <Header credentials used to authenticate and access the decryptionkey>

<-- HTTP/1.1 200 OK
<-- Content-Type: text/plain
<-- Content-Length: 252
<--
<-- <Decrypted contents of Encrypted Value>
```

1246 EXAMPLE 5: GET to the data object URI to read the plaintext of an encrypted object:

```
--> GET /cdmi/2.0.0/MyContainer/MyEncryptedObject.txt HTTP/1.1
--> Host: cloud.example.com
--> <Header credentials used to authenticate and access the decryption key>

<-- HTTP/1.1 200 OK
<-- Content-Type: text/plain
<-- Content-Length: 252
<--
<-- <Decrypted contents of Encrypted Value>
```

6.4 Update a data object using HTTP

6.4.1 Synopsis

The following HTTP PATCH operation updates an existing data object at the specified URI:

- `PATCH <root URI>/<ContainerName>/<DataObjectName>`
- `PATCH <root URI>/cdmi_objectid/<DataObjectID>`

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<ContainerName>` is zero or more intermediate containers.
- `<DataObjectName>` is the name of the data object to be updated.
- `<DataObjectID>` is the ID of the data object to be read from.

6.4.2 Capabilities

Capabilities that indicate which operations are supported are shown in Table 15.

Table 15: Capabilities - Update a CDMI data object using HTTP

Capability	Location	Description
`cdmi_modify_value`	Data Object	Ability to modify the value of an existing data object
`cdmi_modify_value_range`	Data Object	Ability to modify a sub-range of the value of an existing data object
`cdmi_object_access_by_ID`	System Wide Capability	Ability to access the object by ID

6.4.3 Request headers

The HTTP request headers for updating a CDMI data object using HTTP are shown in Table 16.

Table 16: Request headers - Update a CDMI data object using HTTP

Header	Type	Description	Requirement
`Content-Type`	Header string	The content type of the data to be stored as a data object. The value specified in this header shall be converted to lower case and stored in the `mimetype` field of the CDMI data object. • If the `Content-Type` header includes the charset parameter as defined in RFC 2616 [23] of "`utf-8` (e.g., "`;charset=utf-8`"), the `valuetransferencoding` field of the CDMI data object shall be set to "`utf-8`". Otherwise, the `valuetransferencoding` field of the CDMI data object shall be set to "`base64`". • If not specified, the existing `mimetype` field value shall be preserved.	Optional
`Content-Range`	Header string	A valid ranges-specifier (see RFC 2616 [23] Section 14.35.1)	Optional

Continued on next page

Table 16 – continued from previous page

Header	Type	Description	Requirement
`X-CDMI-Partial`	Header string	Indicates that the operation is part of a series of updates and has not yet been fully created. When set to "`true`", the `completionStatus` field shall be set to "`Processing`". `X-CDMI-Partial` works across CDMI and non-CDMI operations. If the `completionStatus` field had previously been set to "`Processing`" by including this header in a create or update, the next update without this field shall change the `completionStatus` field back to "`Complete`".	Optional

6.4.4 Request message body

The request message body contains the data to be stored in the value of the data object.

6.4.5 Response header

The HTTP response header for updating a data object using HTTP is shown in Table 17.

Table 17: Response header - Update a CDMI data object using HTTP

Header	Type	Description	Requirement
`Location`	Header string	The server shall respond with the URI to which the reference redirects if the object is a reference.	Conditional

6.4.6 Response message body

A response body may be provided as per RFC 2616 [23].

6.4.7 Response status

The HTTP status codes that occur when updating a data object using HTTP are described in Table 18.

Table 18: HTTP status codes - Update a CDMI data object using HTTP

HTTP Status	Description
`204 No Content`	The data object content was returned in the response.
`302 Found`	The resource is a reference to another resource.
`400 Bad Request`	The request contains invalid parameters or field names.
`401 Unauthorized`	The authentication credentials are missing or invalid.
`403 Forbidden`	The client lacks the proper authorization to perform this request.
`404 Not Found`	The resource was not found at the specified URI.
`409 Conflict`	The operation conflicts with a non-CDMI access protocol lock or has caused a state transition error on the server.

6.4.8 Examples

EXAMPLE 1: PATCH to the data object URI to update the value of the data object:

```
--> PATCH /cdmi/2.0.0/MyContainer/MyDataObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: text/plain
--> Content-Length: 37
-->
--> This is the value of this data object

<-- HTTP/1.1 204 No Content
```

EXAMPLE 2: PATCH to the data object URI to update four bytes within the value of the data object:

```
--> PATCH /cdmi/2.0.0/MyContainer/MyDataObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Content-Range: bytes 21-24/37
--> Content-Type: text/plain
--> Content-Length: 4
-->
--> that

<-- HTTP/1.1 204 No Content
```

6.5 Delete a data object using HTTP

6.5.1 Synopsis

The following HTTP DELETE operations delete an existing data object at the specified URI:

- `DELETE <root URI>/<ContainerName>/<DataObjectName>`
- `DELETE <root URI>/cdmi_objectid/<DataObjectID>`

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<ContainerName>` is zero or more intermediate containers.
- `<DataObjectName>` is the name of the data object to be deleted.
- `<DataObjectID>` is the ID of the data object to be deleted.

6.5.2 Capability

Capabilities that indicate which operations are supported are shown in Table 19.

Table 19: Capabilities - Delete a CDMI data object using HTTP

Capability	Location	Description
`cdmi_delete_dataobject`	Data Object	Ability to delete an existing data object
`cdmi_object_access_by_ID`	System Wide Capability	Ability to access the object by ID

6.5.3 Request headers

Request headers may be provided as per RFC 2616 [23].

6.5.4 Request message body

A request body may be provided as per RFC 2616 [23].

6.5.5 Response headers

Response headers may be provided as per RFC 2616 [23].

6.5.6 Response message body

A response body may be provided as per RFC 2616 [23].

6.5.7 Response status

Table 20 describes the HTTP status codes that occur when deleting a data object using HTTP.

Table 20: HTTP status codes - Delete a CDMI data object using HTTP

HTTP Status	Description
`204 No Content`	The data object was successfully deleted.
`400 Bad Request`	The request contains invalid parameters or field names.
`401 Unauthorized`	The authentication credentials are missing or invalid.
`403 Forbidden`	The client lacks the proper authorization to perform this request.
`404 Not Found`	The resource was not found at the specified URI.
`409 Conflict`	The operation conflicts with a non-CDMI access protocol lock, has caused a state transition error on the server, or the data object cannot be deleted.

6.5.8 Example

EXAMPLE 1: DELETE to the data object URI:

```
--> DELETE /cdmi/2.0.0/MyContainer/MyDataObject.txt HTTP/1.1
--> Host: cloud.example.com

<-- HTTP/1.1 204 No Content
```

Clause 7

Container Object Resource Operations using HTTP

7.1 Overview

Container objects is the fundamental grouping mechanism for stored data within CDMI, and is analogous to directories in a file system. Each container object has zero or more child objects.

Following the URI conventions for hierarchical paths, container URIs shall consist of one or more container names that are separated by forward slashes (“/”) and that end with a forward slash (“/”).

As basic HTTP operations do not use the CDMI MIME types that distinguish data object operations from container object operations, a CDMI implementation shall use the presence or absence of a forward slash at the end of a URI to distinguish between a container object create or a data object create, respectively.

If a basic HTTP read, update, or delete operation is performed against an existing container resource and the trailing slash at the end of the URI is omitted, the server shall respond with an HTTP status code of `301 Moved Permanently`. In addition, a Location header containing the URI with the trailing slash added shall be returned.

A CDMI server differentiates between HTTP and CDMI operations using the standard Content-Type and Accept headers. When CDMI MIME types defined in RFC 6208 [26] are used in these headers, this indicates that CDMI behaviors, as described in Clause 9 are used in addition to the standard HTTP behaviors.

A CDMI implementation that supports container objects shall include support for basic container object HTTP operations corresponding with the CDMI capabilities that are published by the implementation. Capabilities allow a client to discover which operations (such as create, update, delete, etc.) are supported and are described in Clause 12.

7.2 Create a container object using HTTP

7.2.1 Synopsis

To create a new container object, the following request shall be performed:

- `PUT <root URI>/<ContainerName>/<ContainerObjectName>/`

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<ContainerName>` is zero or more intermediate container objects that already exist, with one slash (i.e., "/") between each pair of container object names.
- `<ContainerObjectName>` is the name specified for the container object to be created.

After it is created, the container object shall also be accessible at `<root URI>/cdmi_objectid/<objectID>/`.

The presence of a trailing slash at the end of the HTTP PUT URI indicates that a container object is being created and distinguishes it from a request to create a data object.

7.2.2 Capabilities

Capabilities that indicate which operations are supported are shown in Table 21.

Table 21: Capabilities - Create a CDMI container object using HTTP

Capability	Location	Description
`cdmi_create_container`	Parent Container	Ability to create a new data object

7.2.3 Request headers

Request headers can be provided as per RFC 2616 [23].

7.2.4 Request message body

A request body shall not be provided.

7.2.5 Response headers

Response headers can be provided as per RFC 2616 [23].

7.2.6 Response message body

A response body can be provided as per RFC 2616 [23].

7.2.7 Response status

Table 22 describes the HTTP status codes that occur when creating a container object using HTTP.

Table 22: HTTP status codes - Create a container object using HTTP

HTTP Status	Description
`201 Created`	The new container object was created.
`400 Bad Request`	The request contains invalid parameters or field names.
`401 Unauthorized`	The authentication credentials are missing or invalid.
`403 Forbidden`	The client lacks the proper authorization to perform this request.
`404 Not Found`	The resource was not found at the specified URI.
`409 Conflict`	The operation conflicts with a non-CDMI access protocol lock or has caused a state transition error on the server.

7.2.8 Example

EXAMPLE 1: PUT to the URI the container object name:

```
--> PUT /cdmi/2.0.0/MyContainer/ HTTP/1.1
--> Host: cloud.example.com

<-- HTTP/1.1 201 Created
```

7.3 Read a container object using HTTP

Reading a container object using HTTP is not defined by this version of this International Standard. A server is allowed to implement responses such as an Apache directory listing or an S3-style bucket listing.

To read a container object using CDMI, see 9.4.

7.4 Update a container object using HTTP

Updating a container object using HTTP is not defined by this version of this International Standard.

To update a container object using CDMI, see 9.5.

7.5 Delete a container object using HTTP

7.5.1 Synopsis

The following HTTP DELETE operations delete an existing container object at the specified URI, including all contained children and snapshots:

- `DELETE <root URI>/<ContainerName>/<ContainerObjectName>/`
- `DELETE <root URI>/cdmi_objectid/<ContainerObjectID>`

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<ContainerName>` is zero or more intermediate container objects.
- `<ContainerObjectName>` is the name of the container object to be deleted.
- `<ContainerObjectID>` is the ID of the container object to be deleted.

7.5.2 Capabilities

Capabilities that indicate which operations are supported are shown in Table 23.

Table 23: Capabilities - Delete a CDMI container object using HTTP

Capability	Location	Description
`cdmi_delete_container`	Parent Container	Ability to delete an existing container object
`cdmi_object_access_by_ID`	System Wide Capability	Ability to access the object by ID

7.5.3 Request headers

Request headers can be provided as per RFC 2616 [23].

7.5.4 Request message body

A request body can be provided as per RFC 2616 [23].

7.5.5 Response headers

Response headers can be provided as per RFC 2616 [23].

7.5.6 Response message body

A response body can be provided as per RFC 2616 [23].

7.5.7 Response status

Table 24 describes the HTTP status codes that occur when deleting a container object using HTTP.

1380 Table 24: HTTP status codes - Delete a CDMI container object using HTTP

1381

HTTP Status	Description
`204 No Content`	The container object was successfully deleted.
`400 Bad Request`	The request contains invalid parameters or field names.
`401 Unauthorized`	The authentication credentials are missing or invalid.
`403 Forbidden`	The client lacks the proper authorization to perform this request.
`404 Not Found`	The resource was not found at the specified URI.
`409 Conflict`	The operation conflicts with a non-CDMI access protocol lock or has caused a state transition error on the server.

1382 ### 7.5.8 Example

1383 EXAMPLE 1: DELETE to the container object URI:

```
--> DELETE /cdmi/2.0.0/MyContainer/ HTTP/1.1
--> Host: cloud.example.com


<-- HTTP/1.1 204 No Content
```

7.6 Create (POST) a new data object using HTTP

7.6.1 Synopsis

To create a new data object in a specified container where the name of the data object is a server-assigned object identifier, the following request shall be performed:

```
POST <root URI>/<ContainerName>/
```

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<ContainerName>` is zero or more intermediate container objects that already exist, with one slash (i.e., "/") between each pair of container object names.

The data object shall be accessible as a child of the container with a server-assigned name and shall also be accessible at `<root URI>/cdmi_objectid/<objectID>`.

HTTP POST to a container is used to enable CDMI servers to support RFC 1867 [20] form-based file uploading. When implementing RFC 1867 [20], the CDMI server-assigned name may be set to, or derived from, the user-provided file name.

7.6.2 Capabilities

Capabilities that indicate which operations are supported are shown in Table 25.

Table 25: Capabilities - Create a CDMI data object using HTTP POST

Capability	Location	Description
`cdmi_create_dataobject` `cdmi_post_dataobject`	Parent Container	Ability to create a new data object
`cdmi_post_dataobject_by_ID`	System Wide Capability	Ability to create a data object in `"/cdmi_objectid/"`
`cdmi_create_value_range`	System Wide Capability	Ability to create a data object using a specified byte range
`cdmi_create_value_range_by_ID`	System Wide Capability	Ability to create a data object in `"/cdmi_objectid/"` using a specified byte range
`cdmi_multipart_mime`	System Wide Capability	Ability to create a data object using multi-part MIME

7.6.3 Request headers

The HTTP request header for creating a new CDMI data object using HTTP is shown in Table 26.

Table 26: Request header - Create a new data object using HTTP

Header	Type	Description	Requirement
`Content-Type`	Header String	The content type of the data to be stored as a data object. The value specified here shall be converted to lower case and stored in the mimetype field of the CDMI data object. • If the content type includes the charset parameter as defined in RFC 2616 [23] of "`utf-8` (e.g., "`;charset=utf-8`"), the valuetransferencoding field of the CDMI data object shall be set to "`utf-8`". Otherwise, the valuetransferencoding field of the CDMI data object shall be set to "`base64`". • If not specified, the mimetype field shall be set to "`application/octet-stream`".	Optional
`X-CDMI-Partial`	Header String	Indicates that the newly created object is part of a series of writes and has not yet been fully created. When set to "`true`", the `completionStatus` field shall be set to "`Processing`". `X-CDMI-Partial` works across CDMI and non-CDMI operations.	Optional

7.6.4 Request message body

The message body shall contain the contents (value) of the data object to be created.

7.6.5 Response headers

The HTTP response header for creating a new CDMI data object using HTTP is shown in Table 27.

Table 27: Response header - Create a new data object using HTTP

Header	Type	Description	Requirement
`Location`	Header string	The unique absolute URI for the new data object as assigned by the system. In the absence of file name information from the client, the system shall assign the URI in the form: `http://host:port/<root URI>/<ContainerName>/<ObjectID>` or `https://host:port/<root URI>/<ContainerName>/<ObjectID>`.	Mandatory

7.6.6 Response message body

A response body can be provided as per RFC 2616 [23].

7.6.7 Response status

Table 28 describes the HTTP status codes that occur when creating a new data object using HTTP.

Table 28: HTTP status codes - Create a new data object using HTTP

HTTP Status	Description
`201 Created`	The new data object was created.
`400 Bad Request`	The request contains invalid parameters or field names.
`401 Unauthorized`	The authentication credentials are missing or invalid.
`403 Forbidden`	The client lacks the proper authorization to perform this request.
`404 Not Found`	The resource was not found at the specified URI.

7.6.8 Examples

EXAMPLE 1: POST to the container object URI the data object contents:

```
--> POST /cdmi/2.0.0/MyContainer/ HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: text/plain;charset=utf-8
-->
--> <object contents>

<-- HTTP/1.1 201 Created
<-- Location: https://cloud.example.com/cdmi/2.0.0/MyContainer/
↪00007ED900104E1D14771DC67C27BF8B
```

Part III

CDMI Core

Clause 8

Data Object Resource Operations using CDMI

8.1 Overview

Data objects are the fundamental storage component within CDMI™ and are analogous to files within a file system. Each data object has a set of well-defined fields that include:

- a mandatory value,
- mandatory fields generated by the cloud storage system,
- mandatory metadata items generated by the cloud storage system,
- optional metadata generated by the cloud storage system; and
- optional metadata specified by the cloud user.

All cloud storage systems shall support data objects, but the ability to create a data object is determiend by the presence or absence of the `cdmi_create_dataobject` and `cdmi_post_dataobject` capabilities in the parent container, and by the `cdmi_post_dataobject_by_ID` system-wide capability for creation by ID.

Each CDMI data object is represented as a JSON object, containing one or more "fields". For example, the "`metadata`" field contains metadata items.

EXAMPLE 1: CDMI Data Object

```
{
    "objectType" : "application/cdmi-object",
    "objectID" : "00007ED90010D891022876A8DE0BC0FD",
    "objectName" : "MyDataObject.txt",
    "parentURI" : "/MyContainer/",
    "parentID" : "00007E7F00102E230ED82694DAA975D2",
    "domainURI" : "/cdmi_domains/MyDomain/",
    "capabilitiesURI" : "/cdmi_capabilities/dataobject/",
    "completionStatus" : "Complete",
    "mimetype" : "text/plain",
    "metadata" : {
        "cdmi_size" : "37"
    },
    "valuetransferencoding" : "utf-8",
    "valuerange" : "0-36",
    "value" : "This is the Value of this Data Object"
}
```

The meaning, use, and permitted values of each field is described in each operation that creates, modifies or retreives CDMI data objects.

8.2 Data object details

8.2.1 Data object addressing

Data objects are addressed in CDMI in two ways:

- by name (e.g. `https://cloud.example.com/cdmi/2.0.0/dataobject`); and
- by ID (e.g. `https://cloud.example.com/cdmi/2.0.0/cdmi_objectid/00007ED90010D891022876A8DE0BC0FD`).

Every data object has a single, globally-unique object identifier (ID) that remains constant for the life of the object. Each data object shall have one or more URI addresses that allow the object to be accessed.

8.2.2 Data object fields

Individual fields within a data object can be accessed by specifying the field name after a question mark "`?`" that is appended to the end of the data object URI.

EXAMPLE 2: The following URI returns the value field in the response body:

```
https://cloud.example.com/cdmi/2.0.0/dataobject?value
```

A list of unique fields, separated by an ampersand "`&`" can be specified, allowing multiple fields to be accessed in a single request.

EXAMPLE 3: The following URI returns the value and metadata fields in the response body:

```
https://cloud.example.com/cdmi/2.0.0/dataobject?value&metadata
```

When a client provides fields that are not defined in this International Standard or deserializes an object containing fields that are not defined in this International Standard, these fields shall be persisted, but shall not be interpreted.

8.2.3 Data Object Value

The encoding of the data transported in the data object value field depends on the data object `valuetransferencoding` field.

- If the value transfer encoding of the object is set to "`utf-8`", the data stored in the value of the data object shall be a valid UTF-8 string and shall be transported as a UTF-8 string in the value field.
- If the value transfer encoding of the object is set to "`base64`", the data stored in the value of the data object can contain arbitrary binary sequences, and it shall be transported as a base 64-encoded string in the value field.
- If the value transfer encoding of the object is set to "`json`", the data stored in the value of the data object shall contain a valid JSON object, and the value field shall contain a valid JSON object. The JSON stored and returned shall be semantically equivalent but may not be syntactically identical. For example, whitespace outside of JSON-quoted strings may be removed or added by either client libraries or by the server. This means that the number of bytes sent may not be the same as the number of bytes stored.

Specific ranges of the value of a data object can be accessed by specifying a byte range after the value field name.

EXAMPLE 4: The following URI returns the first thousand bytes in the value field:

```
https://cloud.example.com/cdmi/2.0.0/dataobject?value=0-999
```

Because a byte range of a UTF-8 string is often not a valid UTF-8 string, the response to a range request shall always be transported in the value field as a base 64-encoded string. Likewise, when updating a range of bytes within the value of a data object, the contents of the value field shall be transported as a base 64-encoded string.

Byte ranges are specified as single inclusive byte ranges as per Section 14.35.1 of RFC 2616 [23].

The value of a data object can also be specified and retrieved using multipart MIME, where the CDMI JSON is transferred in the first MIME part, and the raw object value is transferred in the second MIME part. Each MIME part, including any header fields, shall conform to RFC 2045 [9], RFC 2046 [10], and RFC 2047 [22]. The length of each part can optionally be specified by a `Content-Length` header in addition to the MIME boundary delimiter.

Multiple non-overlapping ranges of the value of a data object can also be accessed or updated in a multipart MIME operation by transferring one MIME part for each range of the value. The byte ranges for these operations shall be specified as per Section 14.35.1 of RFC 2616 [23].

Multipart MIME enables the efficient transfer of binary data alongside CDMI object metadata without incurring the overhead of the UTF-8 or Base64 encoding and validation required to represent binary data in JSON.

8.2.4 Data object metadata

Data object metadata can also include arbitrary user-supplied metadata, storage system metadata, and data system metadata, as specified in clause 16. Metadata shall be stored as a valid UTF-8 string. Binary data stored in user metadata shall be first encoded such that it can be contained in a UTF-8 string, with the use of base 64 encoding recommended.

Every data object has a parent object from which the data object inherits data system metadata that is not explicitly specified in the data object itself.

EXAMPLE 5: The “budget.xls” data object stored at the following URI would inherit data system metadata from its parent container, “finance”:

```
https://cloud.example.com/cdmi/2.0.0/finance/budget.xls
```

8.2.5 Data object access control

If read access to any of the requested fields is not permitted by the object ACL, only the permitted fields shall be returned. If no requested fields are permitted to be read, an HTTP status code of `403 Forbidden` shall be returned to the client.

If write access to any of the requested fields is not permitted by the object ACL, no updates shall be performed, and an HTTP status code of `403 Forbidden` shall be returned to the client.

8.2.6 Data object consistency

Writing to a data object is an atomic operation.

- If a client reads a data object simultaneously with a write to that same data object, the reading client shall get either the old version or the new version, but not a mixture of both.
- If a write is terminated due to errors, the contents of the data object shall be as if the write never occurred (i.e., writes are atomic in the face of errors).

Create and update timestamps that are returned in response to multiple client writes to a given object can indicate that a specific write is the newest (i.e., the write whose data is expected to be returned to subsequent reads until another write is processed). However, there is no guarantee that the write with the latest timestamp is the one whose data is returned on subsequent reads.

Range writes can result in a gap in an object value that have had no data written to them. Reading from a gap in a data object value shall return zero for each byte read.

Implementations of this International Standard shall provide the atomicity features described in this subclause for data objects that are accessed via CDMI. The atomicity properties of data objects that are accessed by protocols other than CDMI are outside the scope of this International Standard.

8.2.7 Data object representations

The representations in this clause are shown using JSON notation. Both clients and servers shall support UTF-8 JSON representation. The request and response body JSON fields may be specified or returned in any order, with the exception that, if present, for data objects, the “`valuerange`” and “`value`” fields shall appear last and in that order.

8.2.8 Encrypted objects

CDMI data object operations only permit management operations and access to the ciphertext of encrypted objects. For more details on encrypted objects, see clause 23.

8.3 Create a data object using CDMI

8.3.1 Synopsis

To create a new data object, the following request shall be performed:

- `PUT <root URI>/<ContainerName>/<DataObjectName>`

To create a new data object by ID, see 9.7.

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<ContainerName>` is zero or more intermediate containers that already exist, with one slash (i.e., "`/`") between each pair of container names.
- `<DataObjectName>` is the name specified for the data object to be created.

After it is created, the data object shall also be accessible at `<root URI>/cdmi_objectid/<objectID>`.

8.3.2 Delayed completion of create

In response to a create operation for a data object, the server may return an HTTP status code of `202 Accepted` to indicate that the object is in the process of being created. This response is useful for long-running operations (e.g., copying a large data object from a source URI). Such a response has the following implications.

- The server shall return a `Location` header with an absolute URI to the object to be created along with an HTTP status code of `202 Accepted`.
- With an HTTP status code of `202 Accepted`, the server implies that the following checks have passed:
  - user authorization for creating the object;
  - user authorization for read access to any source object for move, copy, serialize, or deserialize; and
  - availability of space to create the object or at least enough space to create a URI to report an error.
- A client might not be able to immediately access the created object, e.g., due to delays resulting from the implementation's use of eventual consistency.

The client performs GET operations to the URI to track the progress of the operation. In response, the server returns two fields in its response body to indicate progress.

- A mandatory `completionStatus` text field contains either "`Processing`", "`Complete`", or an error string starting with the value "`Error`".
- An optional `percentComplete` field contains the percentage of the operation that has completed (0 to 100).

GET shall not return any value for the data object when `completionStatus` is not "`Complete`". If the final result of the create operation is an error, the URI is created with the `completionStatus` field set to the error message. It is the client's responsibility to delete the URI after the error has been noted.

8.3.3 Capabilities

Capabilities that indicate which operations are supported are shown in Table 29.

Table 29: Capabilities - Create a CDMI data object using CDMI

Capability	Location	Description
cdmi_create_dataobject	Parent Container	Ability to create a new data object
cdmi_create_reference	Parent Container	Ability to create a new reference
cdmi_copy_dataobject	Parent Container	Ability to create a data object that is a copy of another data object
cdmi_move_dataobject	Parent Container	Ability to move a data object from another container
cdmi_deserialize_dataobject	Parent Container	Ability to create a data object that is deserialized from the contents of the PUT or the contents of another data object
cdmi_serialize_dataobject cdmi_serialize_container cdmi_serialize_domain cdmi_serialize_queue	Parent Container	Ability to create a data object that contains a serialized representation of an existing data object, container, domain or queue
cdmi_create_value_range	Parent Container	Ability to create a data object using a specified byte range
cdmi_multipart_mime	System Wide Capability	Ability to create a data object using multi-part MIME

8.3.4 Request headers

The HTTP request headers for creating a CDMI data object using CDMI are shown in Table 30.

Table 30: Request headers - Create a CDMI data object using CDMI

Header	Type	Description	Requirement
Accept	Header string	"application/cdmi-object" or a consistent value defined in 5.5.2	Optional
Content-Type	Header string	"application/cdmi-object" or "multipart/mixed" • If "multipart/mixed" is specified, the body shall consist of at least two MIME parts, where the first part shall contain a body of content-type "application/cdmi-object", and the second and subsequent parts shall contain one or more byte ranges of the value. • If multiple byte ranges are included and the Content-Range header is omitted for a part, the data in the part shall be appended to the data in the preceding part, with the first part having a byte offset of zero.	Mandatory
X-CDMI-Partial	Header string	Indicates that the newly created object is part of a series of writes and has not yet been fully created. When set to "true", the completionStatus field shall be set to "Processing". X-CDMI-Partial works across CDMI and non-CDMI operations.	Optional

8.3.5 Request message body

The request message body fields for creating a data object using CDMI are shown in Table 31.

Table 31: Request message body - Create a data object using CDMI

Field Name	Type	Description	Requirement
`mimetype`	JSON string	MIME type of the data contained within the value field of the data object • This field may be included when creating by value or when deserializing, serializing, copying, and moving a data object. • If this field is not included and multi-part MIME is not being used, the value of "`text/plain`" shall be assigned as the field value. • If this field is not included and multi-part MIME is being used, the value of the `Content-Type` header of the second MIME part shall be assigned as the field value. • This field field value shall be converted to lower case before being stored.	Optional
`metadata`	JSON object	Metadata for the data object • If this field is included, the contents of the JSON object provided in this field shall be used as data object metadata. • If this field is included when deserializing, serializing, copying, or moving a data object, the contents of the JSON object provided in this field shall be used as object metadata instead of the metadata from the source URI. • If this field is not included, no user-specified metadata shall be added to the object. • If this field is not included when deserializing, serializing, copying, or moving a data object, metadata from the source URI shall be used. • This field shall not be included when creating a reference to a data object.	Optional
`domainURI`	JSON string	URI of the owning domain • If different from the parent domain, the user shall have the "`cross-domain`" privilege (see `cdmi_member_privileges` in Table 80 . • If not specified, the domain of the parent container shall be used.	Optional
`deserialize`	JSON string	URI of a CDMI data object with a value that contains a data object serialized as specified in clause 15. The serialized data object shall be deserialized to create the new data object.	Optional[1]
`serialize`	JSON String	URI of a CDMI object that shall be serialized into the new data object	Optional[1]

Continued on next page

Table 31 – continued from previous page

Field Name	Type	Description	Requirement
`copy`	JSON string	URI of a source CDMI data object or queue object that shall be copied into the new destination data object. • If the destination data object URI and the copy source object URI both do not specify individual fields, the destination data object shall be a complete copy of the source data object. • If the destination data object URI or the copy source object URI specifies individual fields, only the fields specified shall be used to create the destination data object. If specified fields are not present in the source, default field values shall be used. • If the destination data object URI and the copy source object URI both specify fields, an HTTP status code of `400 Bad Request` shall be returned to the client. • If the copy source object URI points to a queue object, as part of the copy operation, multiple queue values shall be concatenated into a single data object value. • If the copy source object URI points to one or more queue object values, as part of the copy operation, the specified queue values shall be concatenated into a single data object value. • If there are insufficient permissions to read the data object at the source URI or create the data object at the destination URI, or if the read operation fails, the copy shall return an HTTP status code of `400 Bad Request`, and the destination object shall not be created.	Optional[1]
`move`	JSON string	URI of an existing local or remote CDMI data object (source URI) that shall be relocated to the URI specified in the PUT. The contents of the object, including the object ID, shall be preserved by a move, and the data object at the source URI shall be removed after the data object at the destination has been successfully created. If there are insufficient permissions to read the data object at the source URI, write the data object at the destination URI, or delete the data object at the source URI, or if any of these operations fail, the move shall return an HTTP status code of `400 Bad Request`, and the source and destination are left unchanged.	Optional[1]
`reference`	JSON string	URI of a CDMI data object that shall be redirected to by a reference. If any other fields are supplied when creating a reference, the server shall respond with an HTTP status code of `400 Bad Request`.	Optional[1]
`deserializevalue`	JSON string	A data object serialized as specified in clause 15 and encoded using base 64 encoding rules described in RFC 4648 [19], that shall be deserialized to create the new data object. • If multi-part MIME is being used and this field contains the value of the MIME boundary parameter, the contents of the second MIME part shall be assigned as the field value. • If the serialized data object in the second MIME part does not include a value field, the contents of the third MIME part shall be assigned as the field value of the value field.	Optional[1]

Continued on next page

Table 31 – continued from previous page

Field Name	Type	Description	Requirement
`valuetransferencoding`	JSON string	The value transfer encoding used for the data object value. Three value transfer encodings are defined. • "`utf-8`" indicates that the data object contains a valid UTF-8 string, and it shall be transported as a UTF-8 string in the value field. • "`base64`" indicates that the data object may contain arbitrary binary sequences, and it shall be transported as a base 64-encoded string in the value field. Setting the contents of the data object value field to any value other than a valid base 64 string shall result in an HTTP status code of `400 Bad Request` being returned to the client. • "`json`" indicates that the data object contains a valid JSON object, and the `value` field shall be a JSON object containing valid JSON data. If the contents of the `value` field are set to any value other than a valid JSON object, an HTTP status code of `400 Bad Request` shall be returned to the client. • This field shall only be included when creating a data object by value. • If this field is not included and multi-part MIME is not being used, the value of "`utf-8`" shall be assigned as the field value. • If this field is not included and multi-part MIME is being used, the value of "`utf-8`" shall be assigned as the field value if the `Content-Type` header of the second and all MIME parts includes the charset parameter as defined in RFC 2046 of "`utf-8`" (e.g., "`;charset=utf-8`"). Otherwise, the value of "`base64`" shall be assigned as the field value. This field applies only to the encoding of the value when represented in CDMI; the `Content-Transfer-Encoding` header of the part specifies the encoding of the value within a multi-part MIME request, as defined in RFC 2045 [9].	Optional[1]
`value`	JSON string	The data object value • If this field is not included and multi-part MIME is not being used, an empty JSON String (i.e., "") shall be assigned as the field value. • If this field is not included and multi-part MIME is being used, the contents of the second MIME part shall be assigned as the field value. • If the `valuetransferencoding` field indicates UTF-8 encoding, the value shall be a UTF-8 string escaped using the JSON escaping rules described in RFC 4627 [5]. • If the `valuetransferencoding` field indicates base 64 encoding, the value shall be first encoded using the base 64 encoding rules described in RFC 4648 [19]. • If the `valuetransferencoding` field indicates JSON encoding, the value shall contain a valid JSON object.	Optional[1]

[1] Only one of these fields shall be specified in any given operation. Except for value, these fields shall not be stored. If more than one of these fields is supplied, the server shall respond with an HTTP status code of `400 Bad Request`.

8.3.6 Response headers

The HTTP response headers for creating a data object using CDMI are shown in Table 32.

Table 32: Response headers - Create a data object using CDMI

Header	Type	Description	Requirement
`Content-Type`	Header string	"`application/cdmi-object`"	Mandatory
`Location`	Header string	When an HTTP status code of `202 Accepted` is returned, the server shall respond with the absolute URL of the object that is in the process of being created.	Conditional

8.3.7 Response message body

The response message body fields for creating a data object using CDMI are shown in Table 33.

Table 33: Response message body - Create a data object using CDMI

Field Name	Type	Description	Requirement
`objectType`	JSON string	"`application/cdmi-object`"	Mandatory
`objectID`	JSON string	Object ID of the object	Mandatory
`objectName`	JSON string	Name of the object	Mandatory
`parentURI`	JSON string	URI for the parent object. Appending the objectName to the parentURI shall always produce a valid URI for the object.	Mandatory
`parentID`	JSON string	Object ID of the parent container object	Mandatory
`domainURI`	JSON string	URI of the owning domain	Mandatory
`capabilitiesURI`	JSON string	URI to the capabilities for the object	Mandatory
`completionStatus`	JSON string	A string indicating if the object is still in the process of being created or updated by another operation, and after that operation is complete, indicates if it was successfully created or updated or if an error occurred. The value shall be the string "`Processing`", the string "`Complete`", or an error string starting with the value "`Error`".	Mandatory
`percentComplete`	JSON string	A string indicating the percentage of completion if the object is still in the process of being created or updated by another operation. • When the value of `completionStatus` is "`Processing`", this field, if provided, shall indicate the percentage of completion as a numeric integer value from "`0`" through "`100`". • When the value of `completionStatus` is "`Complete`", this field, if provided, shall contain the value "`100`". • When the value of `completionStatus` is "`Error`", this field, if provided, may contain any integer value from "`0`" through "`100`".	Optional
`mimetype`	JSON string	MIME type of the value of the data object	Mandatory

Continued on next page

Table 33 – continued from previous page

Field Name	Type	Description	Requirement
metadata	JSON object	Metadata for the data object. This field includes any user and data system metadata specified in the request body metadata field, along with storage system metadata generated by the cloud storage system. See clause 16 for a further description of metadata.	Mandatory

8.3.8 Response status

The HTTP status codes that occur when creating a data object using CDMI are described in Table 34.

Table 34: HTTP status codes - Create a data object using CDMI

HTTP Status	Description
201 Created	The new data object was created.
202 Accepted	The data object is in the process of being created. The CDMI client should monitor the completionStatus and percentComplete fields to determine the current status of the operation.
400 Bad Request	The request contains invalid parameters or field names.
401 Unauthorized	The authentication credentials are missing or invalid.
403 Forbidden	The client lacks the proper authorization to perform this request.
404 Not Found	The resource was not found at the specified URI.
409 Conflict	The operation conflicts with a non-CDMI access protocol lock or has caused a state transition error on the server.

8.3.9 Examples

EXAMPLE 1: PUT to the container URI the data object name and contents:

```
--> PUT /cdmi/2.0.0/MyContainer/MyDataObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-object
--> Content-Type: application/cdmi-object
-->
--> {
--> "mimetype" : "text/plain",
--> "metadata" : {
-->
--> },
--> "value" : "This is the Value of this Data Object"
--> }

<-- HTTP/1.1 201 Created
<-- Content-Type: application/cdmi-object
<--
<-- {
<--   "objectType" : "application/cdmi-object",
<--   "objectID" : "00007ED90010D891022876A8DE0BC0FD",
<--   "objectName" : "MyDataObject.txt",
<--   "parentURI" : "/MyContainer/",
<--   "parentID" : "00007E7F00102E230ED82694DAA975D2",
<--   "domainURI" : "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI" : "/cdmi_capabilities/dataobject/",
<--   "completionStatus" : "Complete",
<--   "mimetype" : "text/plain",
<--   "metadata" : {
<--     "cdmi_size" : "37"
<--   }
<-- }
```

EXAMPLE 2: PUT to the container URI the data object name and binary contents:

```
--> PUT /cdmi/2.0.0/MyContainer/MyDataObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-object
--> Content-Type: application/cdmi-object
-->
--> {
--> "mimetype" : "text/plain",
--> "metadata" : { },
--> "valuetransferencoding" : "base64"
--> "value" : "VGhpcyBpcyB0aGUgVmFsdWUgb2YgdGhpcyBEYXRhIE9iamVjdA=="
--> }

<-- HTTP/1.1 201 Created
<-- Content-Type: application/cdmi-object
<--
<-- {
<--   "objectType": "application/cdmi-object",
<--   "objectID": "00007ED9001008C174ABCE6AC3287E5F",
<--   "objectName": "MyDataObject.txt",
<--   "parentURI": "/MyContainer/",
<--   "parentID" : "00007E7F00102E230ED82694DAA975D2",
<--   "domainURI": "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI": "/cdmi_capabilities/dataobject/",
<--   "completionStatus": "Complete",
<--   "mimetype": "text/plain",
<--   "metadata": {
<--     "cdmi_size": "37"
<--   }
<-- }
```

1573 EXAMPLE 3: PUT to the container URI the data object name and binary contents using multi-part MIME:

```
--> PUT /cdmi/2.0.0/MyContainer/MyDataObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-object
--> Content-Type: multipart/mixed; boundary=gc0p4Jq0M2Yt08j34c0p
-->
--> --gc0p4Jq0M2Yt08j34c0p
--> Content-Type: application/cdmi-object
-->
--> {
-->   "domainURI": "/cdmi_domains/MyDomain/",
-->   "metadata": {
-->     "colour": "blue"
-->   }
--> }
-->
--> --gc0p4Jq0M2Yt08j34c0p
--> Content-Type: application/octet-stream
--> Content-Transfer-Encoding: binary
-->
--> <37 bytes of binary data>
-->
--> --gc0p4Jq0M2Yt08j34c0p--

<-- HTTP/1.1 201 Created
<-- Content-Type: application/cdmi-object
<--
<-- {
<--   "objectType": "application/cdmi-object",
<--   "objectID": "00007ED900103ADE9DE3A8D1CF5436A3",
<--   "objectName": "MyDataObject.txt",
<--   "parentURI": "/MyContainer/",
<--   "parentID" : "00007E7F00102E230ED82694DAA975D2",
<--   "domainURI": "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI": "/cdmi_capabilities/dataobject/",
<--   "completionStatus": "Complete",
<--   "mimetype": "application/octet-stream",
<--   "metadata": {
```

(continues on next page)

(continued from previous page)

```
<--     "cdmi_size": "37",
<--     "colour": "blue",
<--     ...
<--   }
<-- }
```

EXAMPLE 4: PUT to the container URI the data object name and binary contents using multi-part MIME with optional content-lengths for the parts:

```
--> PUT /cdmi/2.0.0/MyContainer/MyDataObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-object
--> Content-Type: multipart/mixed; boundary=gc0p4Jq0M2Yt08j34c0
-->
--> --gc0p4Jq0M2Yt08j34c0p
--> Content-Type: application/cdmi-object
--> Content-Length: 82
-->
--> {
-->   "domainURI": "/cdmi_domains/MyDomain/",
-->   "metadata": {
-->     "colour": "blue"
-->   }
--> }
-->
--> --gc0p4Jq0M2Yt08j34c0p
--> Content-Type: application/octet-stream
--> Content-Transfer-Encoding: binary
--> Content-Length: 37
-->
--> <37 bytes of binary data>
-->
--> --gc0p4Jq0M2Yt08j34c0p--

<-- HTTP/1.1 201 Created
<-- Content-Type: application/cdmi-object
<--
<-- {
<--   "objectType": "application/cdmi-object",
<--   "objectID": "00007ED900103ADE9DE3A8D1CF5436A3",
<--   "objectName": "MyDataObject.txt",
<--   "parentURI": "/MyContainer/",
<--   "parentID" : "00007E7F00102E230ED82694DAA975D2",
<--   "domainURI": "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI": "/cdmi_capabilities/dataobject/",
<--   "completionStatus": "Complete",
<--   "mimetype": "application/octet-stream",
<--   "metadata": {
<--     "cdmi_size": "37",
<--     "colour": "blue",
<--   }
<-- }
```

EXAMPLE 5: PUT to the container URI the data object name and JSON contents:

```
--> PUT /cdmi/2.0.0/MyContainer/MyDataObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-object
--> Content-Type: application/cdmi-object
-->
--> {
-->   "mimetype" : "text/plain",
-->   "metadata" : { },
-->   "valuetransferencoding" : "json"
-->   "value" : {
-->     "test" : "value"
-->   }
--> }
```

(continues on next page)

(continued from previous page)

```
<-- HTTP/1.1 201 Created
<-- Content-Type: application/cdmi-object
<--
<-- {
<--   "objectType": "application/cdmi-object",
<--   "objectID": "0000706D0010374085EF1A5C7018D774",
<--   "objectName": "MyDataObject.txt",
<--   "parentURI": "/MyContainer/",
<--   "parentID" : "00007ED90010067404EDED32860C086A",
<--   "domainURI": "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI": "/cdmi_capabilities/dataobject/",
<--   "completionStatus": "Complete",
<--   "mimetype": "text/plain",
<--   "metadata": {
<--     "cdmi_size": "21"
<--   }
<-- }
```

1577 EXAMPLE 6: PUT to the container URI to create an encrypted object:

```
--> PUT /cdmi/2.0.0/MyContainer/MyEncryptedObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-object
-->
--> {
-->   "mimetype" : "application/cms",
-->   "metadata" : {
-->     "cdmi_enc_key_id" : "testkey"
-->   },
-->   "valuetransferencoding" : "base64"
-->   "value" : "<CMS Encrypted Object in Base64>"
--> }

<-- HTTP/1.1 201 Created
```

1578 EXAMPLE 7: PUT to the container URI to create an encrypted object:

```
--> PUT /cdmi/2.0.0/MyContainer/MyEncryptedObject2.txt HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-object
-->
--> {
-->   "mimetype" : "application/jose+json",
-->   "metadata" : {
-->     "cdmi_enc_key_id" : "77c7e2b8-6e13-45cf-8672-617b5b45243a"
-->   },
-->   "valuetransferencoding" : "json",
-->   "value" : {
-->     "protected": "eyJhbGciOiJkaXIiLCJraWQiOiI3N2M3ZTJi
-->       OC02ZTEzLTQ1Y2YtODY3Mi02MTdiNWI0NTI0
-->       M2EiLCJlbmMiOiJBMTI4R0NNIn0",
-->     "iv": "refa467QzzKx6QAB",
-->     "ciphertext": "JW_i_f52hww_ELQPGaYyeAB6HYGcR559l9T
-->       YnSovc23XJoBcW29rHP8yZOZG7YhLpT1bjF
-->       uvZPjQS-m0IFtVcXkZXdH_lr_FrdYt9HRUY
-->       kshtrMmIUAyGmUnd9zMDB2n0cRDIHAzFVeJ
-->       UDxkUwVAE7_YGRPdcqMyiBoCO-FBdE-Nceb
-->       4h3-FtBP-c_BIwCPTjb9o0SbdcdREEMJMyZ
-->       BH8ySWMVi1gPD9yxi-aQpGbSv_F9N4IZAxs
-->       cj5g-NJsUPbjk29-s7LJAGb15wEBtXphVCg
-->       yy53CoIKLHHeJHXex45Uz9aKZSRSInZI-wj
-->       sY0yu3cT4_aQ3i1o-tiE-F8Ios61EKgyIQ4
-->       CWao8PFMj8TTnp",
-->     "tag": "vbb32Xvllea2OtmHAdccRQ",
-->     "cty": "text/plain"
-->   }
--> }
```

(continues on next page)

(continued from previous page)

```
--> HTTP/1.1 201 Created
```

8.4 Read a data object using CDMI

8.4.1 Synopsis

To read an existing data object, the following requests shall be performed:

- `GET <root URI>/<ContainerName>/<DataObjectName>`
- `GET <root URI>/<ContainerName>/<DataObjectName>?<fieldname>&<fieldname>&...`
- `GET <root URI>/<ContainerName>/<DataObjectName>?value=<range>&...`
- `GET <root URI>/<ContainerName>/<DataObjectName>?metadata=<prefix>&...`
- `GET <root URI>/cdmi_objectid/<DataObjectID>`
- `GET <root URI>/cdmi_objectid/<DataObjectID>?<fieldname>&<fieldname>&...`
- `GET <root URI>/cdmi_objectid/<DataObjectID>?value=<range>&...`
- `GET <root URI>/cdmi_objectid/<DataObjectID>?metadata=<prefix>&...`

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<ContainerName>` is zero or more intermediate containers.
- `<DataObjectName>` is the name of the data object to be read from.
- `<fieldname>` is the name of a field.
- `<range>` is a byte range of the data object value to be returned in the value field.
- `<prefix>` is a matching prefix that returns all metadata items that start with the prefix value.
- `<DataObjectID>` is the ID of the data object to be read from.

8.4.2 Capabilities

Capabilities that indicate which operations are supported are shown in Table 35.

Table 35: Capabilities - Read a CDMI data object using CDMI

Capability	Location	Description
`cdmi_read_metadata`	Data Object	Ability to read the metadata of an existing data object
`cdmi_read_value`	Data Object	Ability to read the value of an existing data object
`cdmi_read_value_range`	Data Object	Ability to read a sub-range of the value of an existing data object
`cdmi_multipart_mime`	System Wide Capability	Ability to read a data object using multi-part MIME
`cdmi_object_access_by_ID`	System Wide Capability	Ability to access the object by ID

8.4.3 Request headers

The HTTP request headers for reading a CDMI data object using CDMI are shown in Table 36.

Table 36: Request headers - Read a CDMI data object using CDMI

Header	Type	Description	Requirement
`Accept`	Header string	“`application/cdmi-object`”, “`multipart/mixed`”, or a consistent value defined in 5.5.2	Optional

8.4.4 Request message body

A request body shall not be provided.

8.4.5 Response headers

The HTTP response headers for reading a data object using CDMI are shown in Table 37.

Table 37: Response headers - Read a CDMI data object using CDMI

Header	Type	Description	Requirement
`Content-Type`	Header string	"`application/cdmi-object`" or "`multipart/mixed`" • If "`multipart/mixed`", the body shall consist of at least two MIME parts, where the first part shall contain a body of content-type "`application/cdmi-object`" and the second and subsequent parts shall contain the requested byte ranges of the value. • If multiple byte ranges are included and the Content-Range header is omitted for a part, the data in the part shall be appended to the data in the preceding part, with the first part having a byte offset of zero.	Mandatory
`Location`	Header string	The server shall respond with the URI that the reference redirects to if the object is a reference.	Conditional

8.4.6 Response message body

The response message body fields for reading a CDMI data object using CDMI are shown in Table 38.

Table 38: Response message body - Read a CDMI data object using CDMI

Field Name	Type	Description	Requirement
`objectType`	JSON string	"`application/cdmi-object`"	Mandatory
`objectID`	JSON string	Object ID of the object	Mandatory
`objectName`	JSON string	Name of the object • For objects in a container, the objectName field shall be returned. • For objects not in a container (objects that are only accessible by ID), the "`objectName`" field does not exist and shall not be returned.	Conditional
`parentURI`	JSON string	URI for the parent object • For objects in a container, the parentURI field shall be returned. • For objects not in a container (objects that are only accessible by ID), the "`parentURI`" field does not exist and shall not be returned. Appending the "`objectName`" to the "`parentURI`" shall always produce a valid URI for the object.	Conditional

Continued on next page

Table 38 – continued from previous page

Field Name	Type	Description	Requirement
parentID	JSON string	Object ID of the parent container object • For objects in a container, the "parentID" field shall be returned. • For objects not in a container (objects that are only accessible by ID), the "parentID" field does not exist and shall not be returned.	Conditional
domainURI	JSON string	URI of the owning domain	Mandatory
capabilitiesURI	JSON string	URI to the capabilities for the object	Mandatory
completionStatus	JSON string	A string indicating if the object is still in the process of being created or updated by another operation, and after that operation is complete, indicates if it was successfully created or updated or if an error occurred. The value shall be the string "Processing", the string "Complete", or an error string starting with the value "Error".	Mandatory
percentComplete	JSON string	A string indicating the percentage of completion if the object is still in the process of bewing created or updated by another operation. • When the value of completionStatus is "Processing", this field, if provided, shall indicate the percentage of completion as a numeric integer value from 0 through 100. • When the value of completionStatus is "Complete", this field, if provided, shall contain the value "100". • When the value of completionStatus is "Error", this field, if provided, may contain any integer value from "0" through "100".	Optional
mimetype	JSON string	MIME type of the value of the data object	Mandatory
metadata	JSON object	Metadata for the data object. This field includes any user and data system metadata specified in the request body metadata field, along with storage system metadata generated by the cloud storage system. See clause 16 for a further description of metadata.	Mandatory
valuerange	JSON string	The range of bytes of the data object to be returned in the value field • If a specific value range has been requested, the valuerange field shall correspond to the bytes requested. If the request extends beyond the end of the value, the valuerange field shall indicate the smaller byte range returned. • If the object value has gaps (due to PUTs with non-contiguous value ranges), the value range will indicate the range to the first gap in the object value. • The cdmi_size storage system metadata of the data object shall always indicate the complete size of the object, including zero-filled gaps.	Mandatory

Continued on next page

Table 38 – continued from previous page

Field Name	Type	Description	Requirement
valuetransferencoding	JSON string	The value transfer encoding used for the data object value. Three value transfer encodings are defined: • "utf-8" indicates that the data object contains a valid UTF-8 string, and it shall be transported as a UTF-8 string in the value field. • "base64" indicates that the data object may contain arbitrary binary sequences, and it shall be transported as a base 64-encoded string in the value field. • "json" indicates that the data object contains a valid JSON object, and the value field shall contain a valid JSON object.	Mandatory
value	JSON string	The data object value • If the valuetransferencoding field indicates UTF-8 encoding, the value field shall contain a UTF-8 string using JSON escaping rules described in RFC 4627 [5]. • If the valuetransferencoding field indicates base 64 encoding, the value field shall contain a base 64-encoded string as described in RFC 4648 [19]. • If the valuetransferencoding field indicates JSON encoding, the value field shall contain a valid JSON object. • The value field shall not be provided when using multi-part MIME. • The value field shall only be provided when the completionStatus field contains "Complete". • When reading a value, zeros shall be returned for any gaps resulting from non-contiguous writes.	Conditional

If individual fields are specified in the GET request, only these fields are returned in the result body. Optional fields that are requested but do not exist are omitted from the result body.

8.4.7 Response status

The HTTP status codes that occur when reading a data object using CDMI are described in Table 39.

Table 39: HTTP status codes - Read a CDMI data object using CDMI

HTTP Status	Description
200 OK	The data object content was returned in the response.
202 Accepted	The data object is in the process of being created. The CDMI client should monitor the completionStatus and percentComplete fields to determine the current status of the operation.
302 Found	The resource is a reference to another resource.
400 Bad Request	The request contains invalid parameters or field names.
401 Unauthorized	The authentication credentials are missing or invalid.
403 Forbidden	The client lacks the proper authorization to perform this request.
404 Not Found	The resource was not found at the specified URI.
406 Not Acceptable	The server is unable to provide the object in the specified in the Accept header.

8.4.8 Examples

EXAMPLE 1: GET to the data object URI to read all fields of the data object:

```
--> GET /cdmi/2.0.0/MyContainer/MyDataObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-object

<-- HTTP/1.1 200 OK
<-- Content-Type: application/cdmi-object
<--
<-- {
<--   "objectType" : "application/cdmi-object",
<--   "objectID" : "00007ED90010D891022876A8DE0BC0FD",
<--   "objectName" : "MyDataObject.txt",
<--   "parentURI" : "/MyContainer/",
<--   "parentID" : "00007E7F00102E230ED82694DAA975D2",
<--   "domainURI" : "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI" : "/cdmi_capabilities/dataobject/",
<--   "completionStatus" : "Complete",
<--   "mimetype" : "text/plain",
<--   "metadata" : {
<--     "cdmi_size" : "37"
<--   },
<--   "valuerange" : "0-36",
<--   "valuetransferencoding" : "utf-8",
<--   "value" : "This is the Value of this Data Object"
<-- }
```

EXAMPLE 2: GET to the data object URI by ID to read all fields of the data object:

```
--> GET /cdmi/2.0.0/cdmi_objectid/00007ED90010D891022876A8DE0BC0FD HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-object

<-- HTTP/1.1 200 OK
<-- Content-Type: application/cdmi-object
<--
<-- {
<--   "objectType" : "application/cdmi-object",
<--   "objectID" : "00007ED90010D891022876A8DE0BC0FD",
<--   "objectName" : "MyDataObject.txt",
<--   "parentURI" : "/MyContainer/",
<--   "parentID" : "00007E7F00102E230ED82694DAA975D2",
<--   "domainURI" : "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI" : "/cdmi_capabilities/dataobject/",
<--   "completionStatus" : "Complete",
<--   "mimetype" : "text/plain",
<--   "metadata" : {
<--     "cdmi_size" : "37"
<--   },
<--   "valuetransferencoding" : "utf-8",
<--   "valuerange" : "0-36",
<--   "value" : "This is the Value of this Data Object"
<-- }
```

EXAMPLE 3: GET to the data object URI to read the value and mimetype fields of the data object:

```
--> GET /cdmi/2.0.0/MyContainer/MyDataObject.txt?value&mimetype HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-object

<-- HTTP/1.1 200 OK
<-- Content-Type: application/cdmi-object
<--
<-- {
<--   "value" : "This is the Value of this Data Object",
<--   "mimetype" : "text/plain"
<-- }
```

1622 EXAMPLE 4: GET to the data object URI to read the first 11 bytes of the value of the data object:

```
--> GET /cdmi/2.0.0/MyContainer/MyDataObject.txt?valuerange&value=0-10 HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-object

<-- HTTP/1.1 200 OK
<-- Content-Type: application/cdmi-object
<--
<-- {
<--   "valuerange" : "0-10",
<--   "value" : "VGhpcyBpcyB0aGU="
<-- }
```

1623 EXAMPLE 5: GET to the data object URI to read the data object using multi-part MIME:

```
--> GET /cdmi/2.0.0/MyContainer/MyDataObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Accept: multipart/mixed

<-- HTTP/1.1 200 OK
<-- Content-Type: multipart/mixed; boundary=gc0p4Jq0M2Yt08j34c0p
<--
<-- --gc0p4Jq0M2Yt08j34c0p
<-- Content-Type: application/cdmi-object
<--
<-- {
<--   "objectType": "application/cdmi-object",
<--   "objectID": "00007ED90010C2414303B5C6D4F83170",
<--   "objectName": "MyDataObject.txt",
<--   "parentURI": "/MyContainer/",
<--   "parentID" : "00007E7F00102E230ED82694DAA975D2",
<--   "domainURI": "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI": "/cdmi_capabilities/dataobject/",
<--   "completionStatus": "Complete",
<--   "mimetype": "application/octet-stream",
<--   "metadata": {
<--     "cdmi_size": "37",
<--     "colour": "blue",
<--       ...
<--   },
<--   "valuerange": "0-36",
<--   "valuetransferencoding": "base64"
<-- }
<--
<-- --gc0p4Jq0M2Yt08j34c0p
<-- Content-Type: application/octet-stream
<-- Content-Transfer-Encoding: binary
<--
<-- <37 bytes of binary data>
<--
<-- --gc0p4Jq0M2Yt08j34c0p—
```

1624 EXAMPLE 6: GET to the data object URI to read the data object using multi-part MIME, with optional content-lengths
1625 for the parts:

```
--> GET /cdmi/2.0.0/MyContainer/MyDataObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Accept: multipart/mixed

<-- HTTP/1.1 200 OK
<-- Content-Type: multipart/mixed; boundary=gc0p4Jq0M2Yt08j34c0p
<--
<-- --gc0p4Jq0M2Yt08j34c0p
<-- Content-Type: application/cdmi-object
<-- Content-Length: 505
<--
<-- {
<--   "objectType": "application/cdmi-object",
<--   "objectID": "00007ED90010C2414303B5C6D4F83170",
```

(continues on next page)

(continued from previous page)

```
<--   "objectName": "MyDataObject.txt",
<--   "parentURI": "/MyContainer/",
<--   "parentID" : "00007E7F00102E230ED82694DAA975D2",
<--   "domainURI": "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI": "/cdmi_capabilities/dataobject/",
<--   "completionStatus": "Complete",
<--   "mimetype": "application/octet-stream",
<--   "metadata": {
<--     "cdmi_size": "37",
<--     "colour": "blue",
<--     ...
<--   },
<--   "valuerange": "0-36",
<--   "valuetransferencoding": "base64"
<-- }
<--
<-- --gc0p4Jq0M2Yt08j34c0p
<-- Content-Type: application/octet-stream
<-- Content-Transfer-Encoding: binary
<-- Content-Length: 37
<--
<-- <37 bytes of binary data>
<--
<-- --gc0p4Jq0M2Yt08j34c0p—
```

EXAMPLE 7: GET to the data object URI to read the metadata and multiple byte ranges of the binary contents using multi-part MIME:

```
--> GET /cdmi/2.0.0/MyContainer/MyDataObject.txt?metadata&value=0-10&value=21-24
↪HTTP/1.1
--> Host: cloud.example.com
--> Accept: multipart/mixed

<-- HTTP/1.1 200 OK
<-- Content-Type: multipart/mixed; boundary=gc0p4Jq0M2Yt08j34c0p
<--
<-- --gc0p4Jq0M2Yt08j34c0p
<-- Content-Type: application/cdmi-object
<--
<-- {
<--   "metadata": {
<--     "cdmi_size": "37",
<--     "colour": "blue",
<--     ...
<--   }
<-- }
<--
<-- --gc0p4Jq0M2Yt08j34c0p
<-- Content-Type: application/octet-stream
<-- Content-Transfer-Encoding: binary
<-- Content-Range: bytes 0-10/37
<--
<-- <11 bytes of binary data>
<--
<-- --gc0p4Jq0M2Yt08j34c0p
<-- Content-Type: application/octet-stream
<-- Content-Transfer-Encoding: binary
<-- Content-Range: bytes 21-24/37
<--
<-- <4 bytes of binary data>
<--
<-- --gc0p4Jq0M2Yt08j34c0p--
```

EXAMPLE 8: GET to the data object URI to read the value and valuetransferencoding fields of a data object storing JSON data:

```
--> GET /cdmi/2.0.0/cdmi_objectid/0000706D0010374085EF1A5C7018D774?
↪valuetransferencoding&value HTTP/1.1
```

(continues on next page)

(continued from previous page)

```
--> Host: cloud.example.com
--> Accept: application/cdmi-object

<-- Content-Type: application/cdmi-object
<--
<-- {
<--   "valuetransferencoding" : "json"
<--   "value" : {
<--     "test" : "value"
<--   }
<-- }
```

1630 EXAMPLE 9: GET to the data object URI to read a newly-created data object with a current version:

```
--> GET /cdmi/2.0.0/MyContainer/MyVersionedDataObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-object

<-- Content-Type: application/cdmi-object
<--
<-- {
<--
<--   "objectType" : "application/cdmi-object",
<--   "objectID" : "00007ED900100DA32EC94351F8970400",
<--   "objectName" : "MyVersionedDataObject.txt",
<--   "parentURI" : "/MyContainer/",
<--   "parentID" : "00007E7F00102E230ED82694DAA975D2",
<--   "domainURI" : "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI" : "/cdmi_capabilities/dataobject/",
<--   "completionStatus" : "Complete",
<--   "mimetype" : "text/plain",
<--   "metadata" : {
<--     "cdmi_size" : "33",
<--     "cdmi_versioning" : "user",
<--     "cdmi_version_object" : "/cdmi_objectid/00007ED900100DA32EC94351F8970400",
<--     "cdmi_version_current" : "/cdmi_objectid/00007ED90010512EB55A9304EAC5D4AA",
<--     "cdmi_version_oldest" : [
<--       "/cdmi_objectid/00007ED90010512EB55A9304EAC5D4AA"
<--     ],
<--     ...
<--   },
<--   "valuerange" : "0-32",
<--   "valuetransferencoding" : "utf-8",
<--   "value" : "First version of this Data Object"
<-- }
```

1631 EXAMPLE 10: GET to the data object URI to read a data object with two historical versions:

```
--> GET /cdmi/2.0.0/MyContainer/MyVersionedDataObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-object

<-- Content-Type: application/cdmi-object
<--
<-- {
<--
<--   "objectType" : "application/cdmi-object",
<--   "objectID" : "00007ED900100DA32EC94351F8970400",
<--   "objectName" : "MyDataObject.txt",
<--   "parentURI" : "/MyContainer/",
<--   "parentID" : "00007E7F00102E230ED82694DAA975D2",
<--   "domainURI" : "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI" : "/cdmi_capabilities/dataobject/",
<--   "completionStatus" : "Complete",
<--   "mimetype" : "text/plain",
<--   "metadata" : {
<--     "cdmi_size" : "33",
<--     "cdmi_versioning" : "user",
```

(continues on next page)

(continued from previous page)

```
<--     "cdmi_version_object" : "/cdmi_objectid/00007ED900100DA32EC94351F8970400",
<--     "cdmi_version_current" : "/cdmi_objectid/00007ED90010F077F4EB1C99C87524CC",
<--     "cdmi_version_oldest" : [
<--       "/cdmi_objectid/00007ED90010512EB55A9304EAC5D4AA"
<--     ],
<--     ...
<--   },
<--   "valuerange" : "0-32",
<--   "valuetransferencoding" : "utf-8",
<--   "value" : "Third version of this Data Object"
<-- }
```

1632 EXAMPLE 11: GET to the URI of a data object version:

```
--> GET /cdmi/2.0.0/cdmi_objectid/00007ED9001005192891EEBE599D94BB HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-object

<-- Content-Type: application/cdmi-object
<--
<-- {
<--   "objectType" : "application/cdmi-object",
<--   "objectID" : "00007ED9001005192891EEBE599D94BB",
<--   "objectName" : "MyVersionedDataObject.txt",
<--   "parentURI" : "/MyContainer/",
<--   "parentID" : "00007E7F00102E230ED82694DAA975D2",
<--   "domainURI" : "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI" : "/cdmi_capabilities/dataobject/dataobject_version/",
<--   "completionStatus" : "Complete",
<--   "mimetype" : "text/plain",
<--   "metadata" : {
<--     "cdmi_size" : "34",
<--     "cdmi_version_object" : "/cdmi_objectid/00007ED900100DA32EC94351F8970400",
<--     "cdmi_version_current" : "/cdmi_objectid/00007ED90010F077F4EB1C99C87524CC",
<--     "cdmi_version_oldest" : [
<--       "/cdmi_objectid/00007ED90010512EB55A9304EAC5D4AA"
<--     ],
<--     "cdmi_version_parent" : "/cdmi_objectid/00007ED90010512EB55A9304EAC5D4AA",
<--     "cdmi_version_children" : [
<--       "/cdmi_objectid/00007ED90010F077F4EB1C99C87524CC"
<--     ],
<--     ...
<--   },
<--   "valuerange" : "0-33",
<--   "valuetransferencoding" : "utf-8",
<--   "value" : "Second version of this Data Object"
<-- }
```

8.5 Update a data object using CDMI

8.5.1 Synopsis

To update part or all of an existing data object, the following requests shall be performed:

- `PATCH <root URI>/<ContainerName>/<DataObjectName>`
- `PATCH <root URI>/<ContainerName>/<DataObjectName>?value=<range>`
- `PATCH <root URI>/<ContainerName>/<DataObjectName>?metadata=<metadataname>&....`
- `PATCH <root URI>/cdmi_objectid/<DataObjectID>`
- `PATCH <root URI>/cdmi_objectid/<DataObjectID>?value=<range>`
- `PATCH <root URI>/cdmi_objectid/<DataObjectID>?metadata=<metadataname>&....`

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<ContainerName>` is zero or more intermediate containers.
- `<DataObjectName>` is the name of the data object to be updated.
- `<range>` is a byte range for the data object value to be updated.
- `<DataObjectID>` is the ID of the data object to be updated.

8.5.2 Capabilities

Capabilities that indicate which operations are supported are shown in Table 40.

Table 40: Capabilities - Update a CDMI data object using CDMI

Capability	Location	Description
`cdmi_modify_metadata`	Data Object	Ability to modify the metadata of an existing data object
`cdmi_modify_value`	Data Object	Ability to modify the value of an existing data object
`cdmi_modify_value_range`	Data Object	Ability to modify a sub-range of the value of an existing data object
`cdmi_multipart_mime`	System Wide Capability	Ability to modify a data object using multi-part MIME
`cdmi_object_access_by_ID`	System Wide Capability	Ability to access the object by ID

8.5.3 Request headers

The HTTP request headers for updating a CDMI data object using CDMI are shown in Table 41.

Table 41: Request headers - Update a CDMI data object using CDMI

Header	Type	Description	Requirement
`Content-Type`	Header string	"`application/cdmi-object`" or "`multipart/mixed`" • If "`multipart/mixed`" is specified, the body shall consist of at least two MIME parts, where the first part shall contain a body of content-type "`application/cdmi-object`", and the second and subsequent parts shall contain one or more byte ranges of the value. • If multiple byte ranges are included and the `Content-Range` header is omitted for a part, the data in the part shall be appended to the data in the preceding part, with the first part having a byte offset of zero.	Mandatory
`X-CDMI-Partial`	Header string	Indicates that the newly created object is part of a series of writes and has not yet been fully created. When set to "`true`", the `completionStatus` field shall be set to "`Processing`". `X-CDMI-Partial` works across CDMI and non-CDMI operations. If the `completionStatus` field had previously been set to "`Processing`" by including this header in a create or update, the next update without this field shall change the `completionStatus` field back to "`Complete`".	Optional

8.5.4 Request message body

The request message body fields for updating a data object using CDMI are shown in Table 42.

Table 42: Request message body - Update a CDMI data object using CDMI

Field Name	Type	Description	Requirement
`mimetype`	JSON string	MIME type of the data contained within the value field of the data object. If present, this value replaces the existing mimetype field value. • This field may be included when updating by value, deserializing, and copying a data object. • If this field is not included, the existing value of the mimetype field shall be left unchanged. • This field field value shall be converted to lower case before being stored. If this field is set to "`application/cms`" or "`application/jose+json`", the CDMI server shall encrypt or reencrypt the value of the object in place, using the key specified by the "`cdmi_enc_key_id`" metadata item. If the "`cdmi_enc_key_id`" metadata item is not present, the object ID shall be used as the key identifier. The mimetype of the plaintext shall be stored in the CMS or JWE JSON representation. If a "`cdmi_enc_value_sign_id`" metadata item is present, the encrypted object shall also be signed. If this field is changed from "`application/cms`" or "`application/jose+json`" to any other mimetype, the CDMI server shall decrypt the value of the object in place, replacing the specified mimetype with the mimetype of the encrypted object, if stored as part of the encrypted object. For more details on encrypted objects, see clause 23.	Optional
`metadata`	JSON object	Metadata for the data object. If present, the new metadata specified replaces the existing object metadata. If individual metadata items are specified in the URI, only those items are replaced; other items are preserved. See clause 16 for a further description of metadata.	Optional
`domainURI`	JSON string	URI of the owning domain • If different from the parent domain, the user shall have the "`cross-domain`" privilege (see `cdmi_member_privileges` in Table 80). • If not specified, the existing domain shall be preserved.	Optional
`deserialize`	JSON string	URI of a CDMI data object with a value that contains a data object serialized as specified in clause 15. The serialized data object shall be deserialized to update the existing data object. The object ID of the serialized data object shall match the object ID of the destination data object. Otherwise, the server shall return an HTTP status code of `400 Bad Request`.	Optional[1]

Continued on next page

Table 42 – continued from previous page

Field Name	Type	Description	Requirement
`copy`	JSON string	URI of a source CDMI data object or queue object that shall be copied into an existing destination data object. • If the destination data object URI and the copy source object URI both do not specify individual fields, the destination data object shall be replaced with the source data object. • If the destination data object URI or the copy source object URI specifies individual fields, only the fields specified shall be used to update the destination data object. If specified fields are not present in the source, these fields shall be ignored. • If the destination data object URI and the copy source object URI both specify fields, an HTTP status code of `400 Bad Request` shall be returned to the client. If the copy source object URI points to a queue object, as part of the copy operation, multiple queue values shall be concatenated into a single data object value. If there are insufficient permissions to read the data object at the source URI, update the data object at the destination URI, or if the read operation fails, the copy shall return an HTTP status code of `400 Bad Request`, and the destination shall be left unchanged.	Optional[1]
`deserializevalue`	JSON string	A data object serialized as specified in clause 15 and encoded using base 64 encoding rules described in RFC 4648 [19], that shall be deserialized to update the existing data object. The object ID of the serialized data object shall match the object ID of the destination data object. Otherwise, the server shall return an HTTP status code of `400 Bad Request`.	Optional[1]

Continued on next page

Table 42 – continued from previous page

Field Name	Type	Description	Requirement
`valuetransferencoding`	JSON string	The value transfer encoding used for the data object value. Three value transfer encodings are defined: • "`utf-8`" indicates that the data object contains a valid UTF-8 string and shall be transported as a UTF-8 string in the value field. If the contents of the data object value field are set or updated to any value other than a valid UTF-8 string, an HTTP status code of 400 Bad Request shall be returned to the client. • "`base64`" indicates that the data object may contain arbitrary binary sequence and shall be transported as a base 64 encoded string in the value field. Setting the contents of the data object value field to any value other than a valid base 64 string shall result in an HTTP status code of `400 Bad Request` being returned to the client. • "`json`" indicates that the data object contains a valid JSON object and shall be transported as a JSON object in the value field. If the contents of the data object value field are set or updated to any value other than a valid JSON object, an HTTP status code of 400 Bad Request shall be returned to the client. This field shall only be included when updating a data object by value. • If this field is not included and multi-part MIME is not being used, the existing value of "`valuetransferencoding`" shall be left unchanged. • If this field is not included and multi-part MIME is being used, the value of "`utf-8`" shall be assigned as the field value if the "`Content-Type`" header of the second and all subsequent MIME parts includes the charset parameter as defined in RFC 2046 of "`utf-8`" (e.g., "`;charset=utf-8`"). Otherwise, the value of "`base64`" shall be assigned as the field value. This field applies only to the encoding of the value when represented in JSON; the "`Content-Transfer-Encoding`" header of the part specifies the encoding of the value within a multi-part MIME request, as defined in RFC 2045.	Optional

Continued on next page

Table 42 – continued from previous page

Field Name	Type	Description	Requirement
`value`	JSON string	This field contains the new data for the object. If present, this value replaces the existing value. • If this field is not included and multi-part MIME is being used, the contents of the second and subsequent MIME parts shall be assigned to the corresponding byte ranges of the field value. • If the valuetransferencoding field indicates UTF-8 encoding, the value shall be a UTF-8 string escaped using the JSON escaping rules described in RFC 4627 [5]. • If the valuetransferencoding field indicates base 64 encoding, the value shall be first encoded using the base 64 encoding rules described in RFC 4648 [19]. • If the valuetransferencoding field indicates JSON encoding, the value field shall contain a valid JSON object. • If a value range was specified in the request, the new data shall be inserted at the location specified by the range. Any resulting gaps between ranges shall be treated as if zeros had been written and shall be included when calculating the size of the value. When storing a range, the value shall be encoded using base 64, and the valuetransferencoding field shall be set to "`base64`".	Optional[1]

8.5.5 Response header

The HTTP response header for updating a data object using CDMI is shown in Table 43.

Table 43: Response header - Update a CDMI data object using CDMI

Header	Type	Description	Requirement
`Location`	Header string	The server shall respond with the URI to which the reference redirects if the object is a reference.	Conditional

8.5.6 Response message body

A response body can be provided as per RFC 2616 [23].

8.5.7 Response status

The HTTP status codes that occur when updating a data object using CDMI are described in Table 44.

[1] Only one of these fields shall be specified in any given operation. Except for value, these fields shall not be stored. If more than one of these fields is supplied, the server shall respond with an HTTP status code of `400 Bad Request`.

Table 44: HTTP status codes - Update a CDMI data object using CDMI

HTTP Status	Description
`204 No Content`	The data object content was returned in the response.
`302 Found`	The resource is a reference to another resource.
`400 Bad Request`	The request contains invalid parameters or field names.
`401 Unauthorized`	The authentication credentials are missing or invalid.
`403 Forbidden`	The client lacks the proper authorization to perform this request.
`404 Not Found`	The resource was not found at the specified URI.
`409 Conflict`	The operation conflicts with a non-CDMI access protocol lock or has caused a state transition error on the server.

8.5.8 Examples

EXAMPLE 1: PATCH to the data object URI to set new field values:

```
--> PATCH /cdmi/2.0.0/MyContainer/MyDataObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-object
-->
--> {
-->   "mimetype" : "text/plain",
-->   "metadata" : {
-->     "colour" : "blue",
-->     "length" : "10"
-->   },
-->   "value" : "This is the Value of this Data Object"
--> }

<-- HTTP/1.1 204 No Content
```

EXAMPLE 2: PATCH to the data object URI to set a new MIME type:

```
--> PATCH /cdmi/2.0.0/MyContainer/MyDataObject.txt?mimetype HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-object
-->
--> {
-->   "mimetype" : "text/plain"
--> }

<-- HTTP/1.1 204 No Content
```

EXAMPLE 3: PATCH to the data object URI to update a range of the value:

```
-->  PATCH /cdmi/2.0.0/MyContainer/MyDataObject.txt?value=21-24 HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-object
-->
--> {
-->   "value" : "dGhhdA=="
--> }

<-- HTTP/1.1 204 No Content
```

When updating a value without specifying a value transfer encoding, the client must be aware of the current value transfer encoding of the object.

- If a client sends a value containing a UTF-8 string that is not a valid base 64 string to update an existing object with a value transfer encoding of "`base64`", the server shall return an error.
- If a client sends a value containing a base 64 string to update an existing object with a value transfer encoding of "`utf-8`", the server shall not return an error. Instead, the server shall store the literal base 64 character sequence in the data object instead of the data encoded in the base 64 string.

EXAMPLE 4: PATCH to the data object URI to replace all metadata with new metadata:

```
--> PATCH /cdmi/2.0.0/MyContainer/MyDataObject.txt?metadata HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-object
-->
--> {
-->   "metadata" : {
-->     "colour" : "red",
-->     "number" : "7"
-->   }
--> }

<-- HTTP/1.1 204 No Content
```

EXAMPLE 5: PATCH to the data object URI to add a new metadata item while preserving existing metadata:

```
--> PATCH /cdmi/2.0.0/MyContainer/MyDataObject.txt?metadata=shape HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-object
-->
--> {
-->  "metadata" : {
-->   "shape" : "round"
-->  }
--> }

<-- HTTP/1.1 204 No Content
```

EXAMPLE 6: PATCH to the data object URI to replace just one metadata item with a new value:

```
--> PATCH /cdmi/2.0.0/MyContainer/MyDataObject.txt?metadata=colour HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-object
-->
--> {
-->   "metadata" : {
-->     "colour" : "green"
-->   }
--> }

<-- HTTP/1.1 204 No Content
```

EXAMPLE 7: Delete a single metadata item:

```
--> PATCH /cdmi/2.0.0/MyContainer/MyDataObject.txt?metadata=colour HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-object
-->
--> {
-->   "metadata": {}
--> }

<-- HTTP/1.1 204 No Content
```

EXAMPLE 8: Add, update, and delete metadata items. Assume a starting condition where the object has a metadata item “colour” with value “green” and a metadata item “shape” with value “round” and does not have a metadata item “size”. After the update, “colour” has value “red”, “shape” is deleted, and “size” has been added with value “10”.

```
--> PATCH /cdmi/2.0.0/MyContainer/MyDataObject.txt?metadata=colour&metadata=shape&
↪metadata=size HTTP/1.1
-->
--> Host: cloud.example.com
--> Content-Type: application/cdmi-object
-->
--> {
-->   "metadata": {
-->     "colour": "red",
-->     "size": "10"
-->   }
--> }
```

(continues on next page)

(continued from previous page)

```
<-- HTTP/1.1 204 No Content
```

EXAMPLE 9: PATCH to the data object URI to set new field values and the binary contents using multi-part MIME:

```
--> PATCH /cdmi/2.0.0/MyContainer/MyDataObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: multipart/mixed; boundary=gc0p4Jq0M2Yt08j34c0p
-->
--> --gc0p4Jq0M2Yt08j34c0p
--> Content-Type: application/cdmi-object
-->
--> {
--> "metadata": {
--> "colour": "red",
--> "number": "7"
--> }
--> }
-->
--> --gc0p4Jq0M2Yt08j34c0p
--> Content-Type: application/octet-stream
--> Content-Transfer-Encoding: binary
-->
--> <37 bytes of binary data>
-->
--> --gc0p4Jq0M2Yt08j34c0p--

<-- HTTP/1.1 204 No Content
```

EXAMPLE 10: PATCH to the data object URI to replace just one metadata item and update multiple byte ranges within the binary contents of the data object using multi-part MIME:

```
--> PATCH /cdmi/2.0.0/MyContainer/BinaryObject.txt?metadata=colour HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: multipart/mixed; boundary=gc0p4Jq0M2Yt08j34c0p
-->
--> --gc0p4Jq0M2Yt08j34c0p
--> Content-Type: application/cdmi-object
-->
--> {
--> "metadata": {
--> "colour": "green"
--> }
--> }
-->
--> --gc0p4Jq0M2Yt08j34c0p
--> Content-Type: application/octet-stream
--> Content-Range: bytes 0-10/37
-->
--> <11 bytes of binary data>
-->
--> --gc0p4Jq0M2Yt08j34c0p
--> Content-Type: application/octet-stream
--> Content-Range: bytes 21-24/37
-->
--> <4 bytes of binary data>
-->
--> --gc0p4Jq0M2Yt08j34c0p--

<-- HTTP/1.1 204 No Content
```

EXAMPLE 11: PATCH to the data object URI to encrypt an existing object:

```
--> PATCH /cdmi/2.0.0/MyContainer/MyDataObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-object
-->
--> {
```

(continues on next page)

(continued from previous page)

```
--> "mimetype" : "application/cms",
--> "metadata" : {
--> "cdmi_enc_key_id" : "testkey"
--> }
--> }

<-- HTTP/1.1 204 No Content
```

1686 EXAMPLE 12: PATCH to the data object URI to decrypt an existing encrypted object:

1687

8.6 Delete a data object using CDMI

8.6.1 Synopsis

To delete an existing data object, the following requests shall be performed:

- `DELETE <root URI>/<ContainerName>/<DataObjectName>`
- `DELETE <root URI>/cdmi_objectid/<DataObjectID>`

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<ContainerName>` is zero or more intermediate containers.
- `<DataObjectName>` is the name of the data object to be deleted.
- `<DataObjectID>` is the ID of the data object to be deleted.

8.6.2 Capabilities

Capabilities that indicate which operations are supported are shown in Table 45.

Table 45: Capabilities - Delete a CDMI data object using CDMI

Capability	Location	Description
`cdmi_delete_dataobject`	Data Object	Ability to delete an existing data object
`cdmi_object_access_by_ID`	System Wide Capability	Ability to access the object by ID

8.6.3 Request headers

Request headers can be provided as per RFC 2616 [23].

8.6.4 Request message body

A request body can be provided as per RFC 2616 [23].

8.6.5 Response headers

Response headers can be provided as per RFC 2616 [23].

8.6.6 Response message body

A response body can be provided as per RFC 2616 [23].

8.6.7 Response status

Table 46 describes the HTTP status codes that occur when deleting a data object using CDMI.

1710
1711 Table 46: HTTP status codes - Delete a CDMI data object using CDMI

HTTP Status	Description
204 No Content	The data object was successfully deleted.
400 Bad Request	The request contains invalid parameters or field names.
401 Unauthorized	The authentication credentials are missing or invalid.
403 Forbidden	The client lacks the proper authorization to perform this request.
404 Not Found	The resource was not found at the specified URI.
409 Conflict	The operation conflicts with a non-CDMI access protocol lock or has caused a state transition error on the server or the data object cannot be deleted.

1712 ### 8.6.8 Example

1713 EXAMPLE 1: DELETE by data object URI:

```
--> DELETE /cdmi/2.0.0/MyContainer/MyDataObject.txt HTTP/1.1
--> Host: cloud.example.com
 `
<-- HTTP/1.1 204 No Content
```

1714 EXAMPLE 2: DELETE by data object ID:

```
--> DELETE /cdmi/2.0.0/cdmi_objectid/00007ED90010D891022876A8DE0BC0FD HTTP/1.1
--> Host: cloud.example.com

<-- HTTP/1.1 204 No Content
```

Clause 9

Container Object Resource Operations using CDMI

9.1 Overview

Container objects are the fundamental grouping of stored data within CDMI™ and are analogous to directories within a file system. Each container object has a set of well-defined fields that include:

- zero or more child objects,
- mandatory fields generated by the cloud storage system,
- mandatory metadata items generated by the cloud storage system,
- optional metadata generated by the cloud storage system; and
- optional metadata specified by the cloud user.

All cloud storage systems shall support containers, but the ability to create a containers is determiend by the presence or absence of the `cdmi_create_container` capability in the parent container.

Each CDMI container object is represented as a JSON object, containing one or more “fields”. For example, the “`metadata`” field contains metadata items.

EXAMPLE 1: CDMI Container Object

```
{
    "objectType" : "application/cdmi-container",
    "objectID" : "00007ED900104E1D14771DC67C27BF8B",
    "objectName" : "MyContainer/",
    "parentURI" : "/",
    "parentID" : "00007E7F0010128E42D87EE34F5A6560",
    "domainURI" : "/cdmi_domains/MyDomain/",
    "capabilitiesURI" : "/cdmi_capabilities/container/",
    "completionStatus" : "Complete",
    "metadata" : {
        "cdmi_ctime" : "2018-05-16T08:01:02.353Z"
    },
    "childrenrange" : "0-4",
    "children" : [
        "red",
        "green",
        "yellow",
        "orange/",
        "purple/"
    ]
}
```

The meaning, use, and permitted values of each field is described in each operation that creates, modifies or retreives CDMI container objects.

9.2 Container object details

9.2.1 Container object addressing

Container objects are addressed in CDMI in two ways:

- by name (e.g. `https://cloud.example.com/cdmi/2.0.0/container/`); and
- by ID (e.g. `https://cloud.example.com/cdmi/2.0.0/cdmi_objectid/00007ED900104E1D14771DC67C27BF8B/`).

Every container object has a single, globally-unique object ID that remains constant for the life of the object. Each container object may also have one or more URI addresses that allow the container object to be accessed.

When a container object is addressed via more than one unique URIs, all operations may be performed through any of these URIs. For example, a container object may be accessible via multiple virtual hosting paths, where `https://cloud.example.com/users/snia/cdmi/` is also accessible through `https://snia.example.com/cdmi/`. Conflicting writes via different paths shall be managed the same way that conflicting writes via one path are managed, via the principle of eventual consistency (see 9.3).

Following the URI conventions for hierarchical paths, container URIs shall consist of one or more container names that are separated by forward slashes ("`/`") and that end with a forward slash ("`/`").

If a request is performed against an existing container resource and the trailing slash at the end of the URI is omitted, the server shall respond with an HTTP status code of `301 Moved Permanently`. In addition, a `Location` header containing the URI with the trailing slash added shall be returned.

If a CDMI request is performed to create a new container resource and the trailing slash at the end of the URI is omitted, the server shall respond with an HTTP status code of `400 Bad Request`.

Non-CDMI requests to create a container resource shall include the trailing slash at the end of the URI; otherwise, the request shall be considered a request to create a data object.

Containers may also be nested.

EXAMPLE 2: The following URI represents a nested container:

```
https://cloud.example.com/container/subcontainer/
```

A nested container has a parent container object, shall be included in the children field of the parent container object, and shall inherit data system metadata and ACLs from its parent container.

This model allows direct mapping between CDMI-managed cloud storage and file systems (e.g., NFSv4 or WebDAV). If a CDMI container object is exported as a file system, then the file system may make the CDMI metadata accessible via file system-specific mechanisms. As files and directories are created by the file system, they become visible through the CDMI interface acting as a data path. The mapping between file system constructs and CDMI data objects, container objects, and metadata is outside the scope of this International Standard.

9.2.2 Container object fields

Individual fields within a container object may be accessed by specifying the field name after a question mark "`?`" appended to the end of the container object URI.

EXAMPLE 3: The following URI returns just the children field in the response body:

```
https://cloud.example.com/cdmi/2.0.0/container/?children
```

EXAMPLE 4: By specifying a range after the children field name, specific ranges of the children field may be accessed.

```
https://cloud.example.com/cdmi/2.0.0/container/?children=0-2
```

Children ranges are specified in a way that is similar to byte ranges as per Section 14.35.1 of RFC 2616 [23]. A client can determine the number of children present by requesting the childrenrange field without requesting a range of children.

A list of fields, separated by an ampersand "`&`" may be specified, allowing multiple fields to be accessed in a single request.

EXAMPLE 5: The following URI would return the children and metadata fields in the response body:

```
https://cloud.example.com/cdmi/2.0.0/container/?children&metadata
```

When a client provides fields that are not defined in this International Standard or deserializes an object containing fields that are not defined in this International Standard, these fields shall be persisted, but shall not be interpreted.

9.2.3 Container object metadata

The following optional container-specific data system metadata may be provided (see Table 47).

Table 47: Container metadata

Metadata Name	Type	Description	Requirement
`cdmi_assignedsize`	JSON string	The number of bytes that is reported via exported protocols (e.g., the device may be thin provisioned). This number may limit `cdmi_size`.	Optional

Container metadata may also include arbitrary user-supplied metadata, storage system metadata, and data system metadata as described in clause 16.

9.2.4 Container object access control

If read access to any of the requested fields is not permitted by the object ACL, only the permitted fields shall be returned. If no requested fields are permitted to be read, an HTTP status code of `403 Forbidden` shall be returned to the client.

If write access to any of the requested fields is not permitted by the object ACL, no updates shall be performed, and an HTTP status code of `403 Forbidden` shall be returned to the client.

9.2.5 Reserved container object names

This International Standard defines reserved container names that should not be used by clients when creating new containers. These container names are reserved for use by this International Standard, and if an attempt is made to create or delete them, an HTTP status code of `400 Bad Request` shall be returned to the client.

Reserved container names defined in this specification include:

- "`cdmi_objectid`"
- "`cdmi_domains`"
- "`cdmi_capabilities`"
- "`cdmi_snapshots` "
- "`cdmi_versions`"

As additional names may be added in future versions of this International Standard, server implementations shall prevent the creation of user-defined containers if the container name starts with "`cdmi_`".

9.2.6 Container object representations

The representations in this clause are shown using JSON notation. Both clients and servers shall support UTF-8 JSON representation. The request and response body JSON fields may be specified or returned in any order, with the exception that, if present, for container objects, the "`childrenrange`" and "`children`" fields shall appear last and in that order.

9.3 Create a container object using CDMI

9.3.1 Synopsis

To create a new container object, the following request shall be performed:

- `PUT <root URI>/<ContainerName>/<NewContainerName>/`

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<ContainerName>` is zero or more intermediate container objects that already exist, with one slash (i.e., "`/`") between each pair of container object names.
- `<NewContainerName>` is the name specified for the container object to be created.

After it is created, the container object shall also be accessible at `<root URI>/cdmi_objectid/<objectID>/`.

9.3.2 Delayed completion of create

In response to a create operation for a container object, the server may return an HTTP status code of `202 Accepted` to indicate that the object is in the process of being created. This response is useful for long-running operations (e.g., deserializing a source data object to create a large container object hierarchy). Such a response has the following implications.

- The server shall return a Location header with an absolute URI to the object to be created along with an HTTP status code of `202 Accepted`.
- With an HTTP status code of `202 Accepted`, the server implies that the following checks have passed:
  - user authorization for creating the container object;
  - user authorization for read access to any source object for move, copy, serialize, or deserialize; and
  - availability of space to create the container object or at least enough space to create a URI to report an error.
- A client might not be able to immediately access the created object, e.g., due to delays resulting from the implementation's use of eventual consistency.

The client performs GET operations to the URI to track the progress of the operation. In response, the server returns two fields in its response body to indicate progress.

- A mandatory `completionStatus` text field contains either "`Processing`", "`Complete`", or an error string starting with the value "`Error`".
- An optional `percentComplete` field contains the percentage that the accepted PUT has completed (0 to 100). GET does not return any children for the container object when `completionStatus` is not "`Complete`".

When the final result of the create operation is an error, the URI is created with the `completionStatus` field set to the error message. It is the client's responsibility to delete the URI after the error has been noted.

9.3.3 Capabilities

Capabilities that indicate which operations are supported are shown in Table 48.

Table 48: Capabilities - Create a CDMI container object using CDMI

Capability	Location	Description
`cdmi_create_container`	Parent container	Ability to create a new container object
`cdmi_create_reference`	Parent container	Ability to create a new reference
`cdmi_copy_container`	Parent container	Ability to create a container object that is a copy of another container object
`cdmi_move_container`	Parent container	Ability to move a container object from another location
`cdmi_deserialize_container`	Parent container	Ability to create a container object that is deserialized from the contents of the PUT or the contents of a data object

9.3.4 Request headers

The HTTP request headers for creating a CDMI container object using CDMI are shown in Table 49.

Table 49: Request headers - Create a container object using CDMI

Header	Type	Description	Requirement
`Accept`	Header string	“application/cdmi-container” or a consistent value described in 5.5.2	Optional
`Content-Type`	Header string	“application/cdmi-container”	Mandatory

9.3.5 Request message body

The request message body fields for creating a container object using CDMI are shown in Table 50.

Table 50: Request message body - Create a container object using CDMI

Field Name	Type	Description	Requirement
`metadata`	JSON object	Metadata for the container object • If this field is included, the contents of the JSON object provided in this field shall be used as container object metadata. • If this field is included when deserializing, serializing, copying, or moving a container object, the contents of the JSON object provided in this field shall be used as object metadata instead of the metadata from the source URI. • If this field is not included, no user-specified metadata shall be added to the object. • If this field is not included when deserializing, serializing, copying, or moving a container object, metadata from the source URI shall be used. • This field shall not be included when creating a reference to a container object.	Optional

Continued on next page

Table 50 – continued from previous page

Field Name	Type	Description	Requirement
domainURI	JSON string	URI of the owning domain • If different from the parent domain, the user shall have the "cross-domain" privilege (see cdmi_member_privileges in Table 80 . • If not specified, the existing domain shall be preserved.	Optional
exports	JSON object	A structure for each protocol enabled for this container object (see clause 13). This field shall not be included when referencing a container object.	Optional
deserialize	JSON string	URI of a CDMI data object with a value that contains a container object serialized as specified in clause 15. The serialized container object shall be deserialized to create the new container object, including all child objects. When deserializing a container object, any exported protocols from the original serialized container object are not applied to the newly created container object(s).	Optional[1]
copy	JSON string	URI of a source CDMI container object that shall be copied into the new destination container object. • If the destination container object URI and the copy source object URI both do not specify individual fields, the destination container object shall be a complete copy of the source container object, including all child objects under the source container object. • If the destination container object URI or the copy source object URI specifies individual fields, only the fields specified shall be used to create the destination container object. If specified fields are not present in the source, default field values shall be used. • If the destination container object URI and the copy source object URI both specify fields, an HTTP status code of 400 Bad Request shall be returned to the client. When copying a container object, exported protocols are not preserved across the copy. If there are insufficient permissions to read the container object at the source URI or create the container object at the destination URI, or if the read operation fails, the copy shall return an HTTP status code of 400 Bad Request, and the destination container object shall not be created.	Optional[1]
move	JSON string	URI of an existing local or remote CDMI container object (source URI) that shall be relocated, along with all child objects, to the URI specified in the PUT. The contents of the container object and all children, including the object ID, shall be preserved by a move, and the container object and all children of the source URI shall be removed after the objects at the destination have been successfully created. If there are insufficient permissions to read the objects at the source URI, write the objects at the destination URI, or delete the objects at the source URI, or if any of these operations fail, the move shall return an HTTP status code of 400 Bad Request, and the source and destination are left unchanged.	Optional[1]
reference	JSON string	URI of a CDMI container object that shall be redirected to by a reference. If other fields are supplied when creating a reference, the server shall respond with an HTTP status code of 400 Bad Request.	Optional[1]

Continued on next page

Table 50 – continued from previous page

Field Name	Type	Description	Requirement
deserializevalue	JSON string	A container object serialized as specified in clause 15 and encoded using base 64 encoding rules described in RFC 4648 [19], that shall be deserialized to create the new container object, including all child objects.	Optional[1]

9.3.6 Response headers

The HTTP response headers for creating a CDMI container object using CDMI are shown in Table 51.

Table 51: Response headers - Create a container object using CDMI

Header	Type	Description	Requirement
Content-Type	Header string	"application/cdmi-container"	Mandatory
Location	Header string	When an HTTP status code of 202 Accepted is returned, the server shall respond with the absolute URL of the object that is in the process of being created.	Conditional

9.3.7 Response message body

The response message body fields for creating a CDMI container object using CDMI are shown in Table 52.

Table 52: Response message body - Create a container object using CDMI

Field Name	Type	Description	Requirement
objectType	JSON string	"application/cdmi-container"	Mandatory
objectID	JSON string	Object ID of the object	Mandatory
objectName	JSON string	Name of the object	Mandatory
parentURI	JSON string	URI for the parent object Appending the objectName to the parentURI shall always produce a valid URI for the object.	Mandatory
parentID	JSON string	Object ID of the parent container object	Mandatory
domainURI	JSON string	URI of the owning domain	Mandatory
capabilitiesURI	JSON string	URI to the capabilities for the object	Mandatory
completionStatus	JSON string	A string indicating if the object is still in the process of being created or updated by another operation, and after that operation is complete, indicates if it was successfully created or updated or if an error occurred. The value shall be the string "Processing", the string "Complete", or an error string starting with the value "Error".	Mandatory

Continued on next page

[1] Only one of these fields shall be specified in any given operation. Except for value, these fields shall not be stored. If more than one of these fields is supplied, the server shall respond with an HTTP status code of 400 Bad Request.

Table 52 – continued from previous page

Field Name	Type	Description	Requirement
percentComplete	JSON string	A string indicating the percentage of completion if the object is still in the process of bewing created or updated by another operation. • When the value of completionStatus is "Processing", this field, if provided, shall indicate the percentage of completion as a numeric integer value from "0" through "100". • When the value of completionStatus is "Complete", this field, if provided, shall contain the value "100". • When the value of completionStatus is "Error", this field, if provided, may contain any integer value from "0" through "100".	Optional
metadata	JSON object	Metadata for the container object. This field includes any user and data system metadata specified in the request body metadata field, along with storage system metadata generated by the cloud storage system. See clause 16 for a further description of metadata.	Mandatory
exports	JSON object	A structure for each protocol that is enabled for this container object. See clause 13.	Optional[2]
snapshots	JSON array of JSON strings	URI(s) of the snapshot container objects. See clause 14.	Optional[2]
childrenrange	JSON string	The children of the container expressed as a range. If a range of children is requested, this field indicates the children returned as a range. This field should not be returned in the response message body that is associated with a copy, move, deserialize, or deserialize value operation.	Optional
children	JSON array of JSON strings	Names of the children objects in the container object. Child container objects end with "/". This field should not be returned in the response message body that is associated with a copy, move, deserialize, or deserialize value operation.	Optional

9.3.8 Response status

Table 53 describes the HTTP status codes that occur when creating a container object using CDMI.

Table 53: HTTP status codes - Create a CDMI container object using CDMI

HTTP status	Description
201 Created	The new container object was created.
202 Accepted	The container is in the process of being created. The CDMI client should monitor the completionStatus and percentComplete fields to determine the current status of the operation.
400 Bad Request	The request contains invalid parameters or field names.
401 Unauthorized	The authentication credentials are missing or invalid.
403 Forbidden	The client lacks the proper authorization to perform this request.
404 Not Found	The resource was not found at the specified URI.
409 Conflict	The operation conflicts with a non-CDMI access protocol lock or had caused a state transition error on the server.

[2] Returned only if present.

9.3.9 Examples

EXAMPLE 1: Create a new container with no metadata:

```
--> PUT /cdmi/2.0.0/MyContainer/ HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-container
--> Content-Type: application/cdmi-container
-->
--> {
--> }

<-- HTTP/1.1 201 Created
<-- Content-Type: application/cdmi-container
<--
<-- {
<--   "objectType" : "application/cdmi-container",
<--   "objectID" : "00007ED900104E1D14771DC67C27BF8B",
<--   "objectName" : "MyContainer/",
<--   "parentURI" : "/",
<--   "parentID" : "00007E7F0010128E42D87EE34F5A6560",
<--   "domainURI" : "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI" : "/cdmi_capabilities/container/",
<--   "completionStatus" : "Complete",
<--   "metadata" : {
<--     ...
<--   },
<--   "childrenrange": "",
<--   "children": []
<-- }
```

EXAMPLE 2: Create a container with metadata:

```
--> PUT /cdmi/2.0.0/MyContainer/ HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-container
--> Content-Type: application/cdmi-container
-->
--> {
-->   "metadata": {
-->     "Colour": "Yellow"
-->   }
--> }

<-- HTTP/1.1 201 Created
<-- Content-Type: application/cdmi-container
<--
<-- {
<--   "objectType" : "application/cdmi-container",
<--   "objectID" : "00007ED900104E1D14771DC67C27BF8B",
<--   "objectName" : "MyContainer/",
<--   "parentURI" : "/",
<--   "parentID" : "00007E7F0010128E42D87EE34F5A6560",
<--   "domainURI" : "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI" : "/cdmi_capabilities/container/",
<--   "completionStatus" : "Complete",
<--   "metadata" : {
<--     "Colour": "Yellow",
<--     ...
<--   },
<--   "childrenrange": "",
<--   "children": []
<-- }
```

EXAMPLE 3: Create a container that is a copy of a container:

```
--> PUT /cdmi/2.0.0/MyContainerCopy/ HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-container
```

(continues on next page)

(continued from previous page)

```
--> Content-Type: application/cdmi-container
-->
--> {
-->   "copy": "/MyContainer/"
--> }

<-- HTTP/1.1 201 Created
<-- Content-Type: application/cdmi-container
<--
<-- {
<--   "objectType" : "application/cdmi-container",
<--   "objectID" : "00007ED900104E1D14771DC67C27BF8B",
<--   "objectName" : "MyContainerCopy/",
<--   "parentURI" : "/",
<--   "parentID" : "00007E7F0010128E42D87EE34F5A6560",
<--   "domainURI" : "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI" : "/cdmi_capabilities/container/",
<--   "completionStatus" : "Complete",
<--   "metadata" : {
<--     "Colour": "Yellow",
<--     ...
<--   }
<-- }
```

1856 EXAMPLE 4: Rename a container:

```
--> PUT /cdmi/2.0.0/MyContainerRenamed/ HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-container
--> Content-Type: application/cdmi-container
-->
--> {
-->   "move": "/MyContainer/"
--> }

<-- HTTP/1.1 201 Created
<-- Content-Type: application/cdmi-container
<--
<-- {
<--   "objectType" : "application/cdmi-container",
<--   "objectID" : "00007ED900104E1D14771DC67C27BF8B",
<--   "objectName" : "MyContainerRenamed/",
<--   "parentURI" : "/",
<--   "parentID" : "00007E7F0010128E42D87EE34F5A6560",
<--   "domainURI" : "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI" : "/cdmi_capabilities/container/",
<--   "completionStatus" : "Complete",
<--   "metadata" : {
<--     "Colour": "Yellow",
<--     ...
<--   }
<-- }
```

9.4 Read a container object using CDMI

9.4.1 Synopsis

To read an existing container object, the following requests shall be performed:

- `GET <root URI>/<ContainerName>/<TheContainerName>/`
- `GET <root URI>/<ContainerName>/<TheContainerName>/?<fieldname>&<fieldname>&...`
- `GET <root URI>/<ContainerName>/<TheContainerName>/?children=<range>&...`
- `GET <root URI>/<ContainerName>/<TheContainerName>/?metadata=<prefix>&...`
- `GET <root URI>/cdmi_objectid/<ContainerObjectID>/`
- `GET <root URI>/cdmi_objectid/<ContainerObjectID>/?<fieldname>&<fieldname>&...`
- `GET <root URI>/cdmi_objectid/<ContainerObjectID>/?children=<range>&...`
- `GET <root URI>/cdmi_objectid/<ContainerObjectID>/?metadata=<prefix>&...`

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<ContainerName>` is zero or more intermediate container objects.
- `<TheContainerName>` is the name specified for the container object to be read from.
- `<fieldname>` is the name of a field.
- `<range>` is a numeric range within the list of children.
- `<prefix>` is a matching prefix that returns all metadata items that start with the prefix value.
- `<ContainerObjectID>` is the ID of the data object to be read from.

9.4.2 Capabilities

Capabilities that indicate which operations are supported are shown in Table 54.

Table 54: Capabilities - Read a CDMI Container Object using CDMI

Capability	Location	Description
`cdmi_read_metadata`	Container object	Ability to read the metadata of an existing container object
`cdmi_list_children`	Container object	Ability to list the children of an existing container object
`cdmi_list_children_range`	Container object	Ability to list a specific range of children of an existing container object
`cdmi_object_access_by_ID`	System wide capability	Ability to access the object by ID

9.4.3 Request headers

The HTTP request headers for reading a CDMI container object using CDMI are shown in Table 55.

Table 55: Request headers - Read a container object using CDMI

Header	Type	Description	Requirement
`Accept`	Header string	“application/cdmi-container” or a consistent value as described in 5.5.2	Optional

9.4.4 Request message body

A request body shall not be provided.

9.4.5 Response headers

The HTTP response headers for reading a CDMI container object using CDMI are shown in *Response headers - Read a container object using CDMI*.

Table 56: Response headers - Read a container object using CDMI

Header	Type	Description	Requirement
`Content-Type`	Header string	"application/cdmi-container"	Mandatory
`Location`	Header string	The server shall respond with an absolute URI to which the reference redirects if the object is a reference.	Conditional

9.4.6 Response message body

The response message body fields for reading a CDMI container object using CDMI are shown in Table 57

Table 57: Response message body - Read a container object using CDMI

Field Name	Type	Description	Requirement
`objectType`	JSON string	"`application/cdmi-container`"	Mandatory
`objectID`	JSON string	Object ID of the object	Mandatory
`objectName`	JSON string	Name of the object • For objects in a container, the objectName field shall be returned. • For objects not in a container (objects that are only accessible by ID), the "`objectName`" field does not exist and shall not be returned.	Conditional
`parentURI`	JSON string	URI for the parent object • For objects in a container, the parentURI field shall be returned. • For objects not in a container (objects that are only accessible by ID), the "`parentURI`" field does not exist and shall not be returned. Appending the "`objectName`" to the "`parentURI`" shall always produce a valid URI for the object.	Conditional
`parentID`	JSON string	Object ID of the parent container object • For objects in a container, the "`parentID`" field shall be returned. • For objects not in a container (objects that are only accessible by ID), the "`parentID`" field does not exist and shall not be returned.	Conditional
`domainURI`	JSON string	URI of the owning domain	Mandatory
`capabilitiesURI`	JSON string	URI to the capabilities for the object	Mandatory

Continued on next page

Table 57 – continued from previous page

Field Name	Type	Description	Requirement
completionStatus	JSON string	A string indicating if the object is still in the process of being created or updated by another operation, and after that operation is complete, indicates if it was successfully created or updated or if an error occurred. The value shall be the string "Processing", the string "Complete", or an error string starting with the value "Error".	Mandatory
percentComplete	JSON string	A string indicating the percentage of completion if the object is still in the process of bewing created or updated by another operation. • When the value of completionStatus is "Processing", this field, if provided, shall indicate the percentage of completion as a numeric integer value from 0 through 100. • When the value of completionStatus is "Complete", this field, if provided, shall contain the value "100". • When the value of completionStatus is "Error", this field, if provided, may contain any integer value from "0" through "100".	Optional
metadata	JSON object	Metadata for the container object. This field includes any user and data system metadata specified in the request body metadata field, along with storage system metadata generated by the cloud storage system. See clause 16 for a further description of metadata.	Mandatory
exports	JSON object	A structure for each protocol that is enabled for this container object (see clause 13)	Optional[1]
snapshots	JSON array of JSON strings	URIs of the snapshot container objects	Optional[1]
childrenrange	JSON string	The children of the container expressed as a range. If a range of children is requested, this field indicates the children returned as a range.	Mandatory
children	JSON array of JSON strings	Names of the children objects in the container object. When a client uses a child name in a request URI or a header URI, the client shall escape reserved characters according to RFC 3986 [2], e.g., a "%" character in a child name shall be replaced with "%25". • Children that are container objects shall have "/" appended to the child name. • Children that are references shall have "?" appended to the child name.	Mandatory

1887 If individual fields are specified in the GET request, only these fields are returned in the result body. Optional fields that
1888 are requested but do not exist are omitted from the result body.

1889 9.4.7 Response status

1890 Table 58 describes the HTTP status codes that occur when reading a container object using CDMI.

[1] Returned only if present.

Table 58: HTTP status codes - Read a container object using CDMI

HTTP status	Description
200 OK	The metadata for the container object is provided in the message body.
302 Found	The resource is a reference to another resource.
400 Bad Request	The request contains invalid parameters or field names.
401 Unauthorized	The authentication credentials are missing or invalid.
403 Forbidden	The client lacks the proper authorization to perform this request.
404 Not Found	The resource was not found at the specified URI.
406 Not Acceptable	The server is unable to provide the object in the content type specified in the Accept header.

9.4.8 Examples

EXAMPLE 1: GET to the container object URI to read all the fields of the container object:

```
--> GET /cdmi/2.0.0/MyContainer/ HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-container

<-- HTTP/1.1 200 OK
<-- Content-Type: application/cdmi-container
<--
<-- {
<--   "objectType" : "application/cdmi-container",
<--   "objectID" : "00007ED900104E1D14771DC67C27BF8B",
<--   "objectName" : "MyContainer/",
<--   "parentURI" : "/",
<--   "parentID" : "00007E7F0010128E42D87EE34F5A6560",
<--   "domainURI" : "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI" : "/cdmi_capabilities/container/",
<--   "completionStatus" : "Complete",
<--   "metadata" : {
<--     ...
<--   },
<--   "exports" : {
<--     "OCCI/iSCSI": {
<--       "identifier": "00007E7F00104BE66AB53A9572F9F51E",
<--       "permissions": [
<--         "https://example.com/compute/0/",
<--         "https://example.com/compute/1/"
<--       ]
<--     },
<--     "Network/NFSv4" : {
<--       "identifier" : "/users",
<--       "permissions" : "domain"
<--     },
<--     "childrenrange" : "0-4",
<--     "children" : [
<--       "red",
<--       "green",
<--       "yellow",
<--       "orange/",
<--       "purple/"
<--     ]
<--   }
<-- }
```

EXAMPLE 2: GET to the container object URI to read parentURI and children of the container object:

```
--> GET /cdmi/2.0.0/MyContainer/?parentURI&children HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-container
```

(continues on next page)

(continued from previous page)

```
<-- HTTP/1.1 200 OK
<-- Content-Type: application/cdmi-container
<--
<-- {
<--   "parentURI" : "/",
<--   "children" : [
<--     "red",
<--     "green",
<--     "yellow",
<--     "orange/",
<--     "purple/"
<--   ]
<-- }
```

1896 EXAMPLE 3: GET to the container object URI to read children 0..2 and childrenrange of the container object:

```
--> GET /cdmi/2.0.0/MyContainer/?childrenrange&children=0-2 HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-container

<-- HTTP/1.1 200 OK
<-- Content-Type: application/cdmi-container
<--
<-- {
<--   "childrenrange" : "0-2",
<--   "children" : [
<--     "red",
<--     "green",
<--     "yellow"
<--   ]
<-- }
```

1897 EXAMPLE 4: GET to the container object by ID to read children 0..2 and childrenrange of the container object:

```
--> GET /cdmi/2.0.0/cdmi_objectid/0000706D0010B84FAD185C425D8B537E/?childrenrange&
↪children=0-2 HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-container

<-- HTTP/1.1 200 OK
<-- Content-Type: application/cdmi-container
<--
<-- {
<--   "childrenrange": "0-2",
<--   "children": [
<--     "red",
<--     "green",
<--     "yellow"
<--   ]
<-- }
```

9.5 Update a container object using CDMI

9.5.1 Synopsis

To update part or all of an existing container object, the following requests shall be performed:

- `PATCH <root URI>/<ContainerName>/<TheContainerName>`
- `PATCH <root URI>/<ContainerName>/<TheContainerName>?metadata=<metadataname>&....`
- `PATCH <root URI>/cdmi_objectid/<ContainerObjectID>`
- `PATCH <root URI>/cdmi_objectid/<ContainerObjectID>?metadata=<metadataname>&....`

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<ContainerName>` is zero or more intermediate container objects.
- `<TheContainerName>` is the name of the container object to be updated.
- `<ContainerObjectID>` is the ID of the data object to be updated.

9.5.2 Delayed completion of snapshot

If the creation of a snapshot (see clause 14) is requested by including a snapshot field in the request message body, the server may return an HTTP status code of `202 Accepted`. Such a response has the following implications:

- With an HTTP status code of `202 Accepted`, the server implies that the following checks have passed:
  - user authorization for creating the snapshot,
  - user authorization for read access to the container object, and
  - availability of space to create the snapshot or at least enough space to create a URI to report an error.
- A client might not be able to immediately access the snapshot, e.g., due to delays resulting from the implementation's use of eventual consistency.

The client performs GET operations to the snapshot URI to track the progress of the operation. In particular, the server returns two fields in its response body to indicate progress:

- A `completionStatus` field contains either "`Processing`", "`Complete`", or an error string starting with the value "`Error`".
- An optional `percentComplete` field contains the percentage that the accepted PATCH has completed ("`0`" to "`100`"). GET does not return any value for the object when `completionStatus` is not "`Complete`".

When the final result of the snapshot operation is an error, the snapshot URI is created with the `completionStatus` field set to the error message. It is the client's responsibility to delete the URI after the error has been noted.

9.5.3 Capabilities

Capabilities that indicate which operations are supported are shown in Table 59.

Table 59: Capabilities - Update a CDMI container object using CDMI

Capability	Location	Description
`cdmi_modify_metadata`	Container object	Ability to modify the metadata of an existing container object
`cdmi_snapshot`	Container object	Ability to create a new snapshot of an existing container object
`cdmi_export_<protocol>`	Container object	Ability to add and modify exports for an existing container object
`cdmi_object_access_by_ID`	System wide capability	Ability to access the object by ID

9.5.4 Request headers

The HTTP request headers for updating a CDMI container object using CDMI are shown in Table 60.

Table 60: Request headers - Update a container object using CDMI

Header	Type	Description	Requirement
`Content-Type`	Header string	"`application/cdmi-container`"	Mandatory

9.5.5 Request message body

The request message body fields for updating a container object using CDMI are shown in Table 61.

Table 61: Request message body - Update a container object using CDMI

Field Name	Type	Description	Requirement
`metadata`	JSON object	Metadata for the container object. If present, the new metadata specified replaces the existing object metadata. If individual metadata items are specified in the URI, only those items are replaced; other items are preserved. See clause 16 for a further description of metadata.	Optional
`domainURI`	JSON string	URI of the owning domain • If different from the parent domain, the user shall have the "`cross-domain`" privilege (see `cdmi_member_privileges` in Table 80). • If not specified, the parent domain shall be used.	Optional

Continued on next page

Table 61 – continued from previous page

Field Name	Type	Description	Requirement
`snapshot`	JSON string	Name of the snapshot to be taken. This is not a URL, but rather, the final component of the absolute URL where the snapshot will exist when the snapshot operation successfully completes. • If a snapshot is added or changed, the PATCH operation only returns after the snapshot is added to the snapshot list. • After they are created, snapshots may be accessed as children container objects under the `cdmi_snapshots` child container object of the container object receiving a snapshot. • When creating a snapshot with the same name as an existing snapshot, the new snapshot will replace the existing snapshot.	Optional
`deserialize`	JSON string	URI of a CDMI data object with a value that contains a container object serialized as specified in clause 15. The serialized container object shall be deserialized to update the existing container object. The object ID of the serialized container object shall match the object ID of the destination container object. Otherwise, the server shall return an HTTP status code of `400 Bad Request`. • If the serialized container object does not contain children, the update is applied only to the container object, and any existing children are left as is. • If the serialized container object does contain children, then creates, updates, and deletes are recursively applied for each child, depending on the differences between the provided serialized state and the current state of the child.	Optional[1]
`copy`	JSON string	URI of a CDMI container object that shall be copied into the existing container object. Only the contents of the container object itself shall be copied, not any children of the container object. • If the destination container object URI and the copy source object URI both do not specify individual fields, the destination container object shall be replaced with the source container object, including all child objects under the source container object. • If the destination container object URI or the copy source object URI specifies individual fields, only the fields specified shall be used to update the destination container object. If specified fields are not present in the source, these fields shall be ignored. • If the destination container object URI and the copy source object URI both specify fields, an HTTP status code of `400 Bad Request` shall be returned to the client. When copying a container object, exported protocols are not preserved across the copy. If there are insufficient permissions to read the container object at the source URI or create the container object at the destination URI, or if the read operation fails, the copy shall return an HTTP status code of `400 Bad Request`, and the destination container object shall not be updated.	Optional[1]

Continued on next page

Table 61 – continued from previous page

Field Name	Type	Description	Requirement
`deserializevalue`	JSON sting	A container object serialized as specified in clause 15 and encoded using base 64 encoding rules described in RFC 4648 [19], that shall be deserialized to update the existing container object. The object ID of the serialized container object shall match the object ID of the destination container object. Otherwise, the server shall return an HTTP status code of `400 Bad Request`. • If the serialized container object does not contain children, the update is applied only to the container object, and any existing children are left as is. • If the serialized container object does contain children, then creates, updates, and deletes are recursively applied for each child, depending on the differences between the provided serialized state and the current state of the children.	Optional[1]
`exports`	JSON object	A structure for each protocol that is enabled for this container object (see clause 13). If an exported protocol is added or changed, the PATCH operation only returns after the export operation has completed.	Optional

[1] Only one of these fields shall be specified in any given operation. Except for value, these fields shall not be stored.

9.5.6 Response headers

The HTTP response header for updating a CDMI container object using CDMI is shown in Table 62.

Table 62: Response header - Update a container object using CDMI

Header	Type	Description	Requirement
`Location`	Header string	The server shall respond with an absolute URI to which the reference redirects if the object is a reference.	Conditional

9.5.7 Response message body

A response body can be provided as per RFC 2616 [23].

9.5.8 Response status

Table 63 describes the HTTP status codes that occur when updating a container object using CDMI.

Table 63: HTTP status codes - Update a container object using CDMI

HTTP status	Description
`204 No Content`	The data object content was returned in the response.
`202 Accepted`	The container or snapshot (subcontainer object) is in the process of being created. The CDMI client should montitor the `completionStatus` and `percentComplete` fields to determine the current status of the operation.
`302 Found`	The resource is a reference to another resource.
`400 Bad Request`	The request contains invalid parameters or field names.
`401 Unauthorized`	The authentication credentials are missing or invalid.
`403 Forbidden`	The client lacks the proper authorization to perform this request.
`404 Not Found`	The resource was not found at the specified URI.
`409 Conflict`	The operation conflicts with a non-CDMI access protocol lock or has caused a state transition error on the server.

9.5.9 Examples

EXAMPLE 1: PATCH to the container object URI to replace all metadata with new metadata:

```
--> PATCH /cdmi/2.0.0/MyContainer/ HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-container
-->
--> {
-->   "metadata" : {
-->     "colour" : "red",
-->     "number" : "7"
-->   }
--> }

<-- HTTP/1.1 204 No Content
```

EXAMPLE 2: PATCH to the container object URI to set a new exported protocol value:

```
--> PATCH /cdmi/2.0.0/MyContainer/?exports HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-container
-->
--> {
-->   "exports" : {
-->     "OCCI/iSCSI" : {
```

(continues on next page)

(continued from previous page)

```
-->       "identifier" : "00007ED900104E1D14771DC67C27BF8B",
-->       "permissions" : "00007E7F00104EB781F900791C70106C"
-->     },
-->     "Network/NFSv4" : {
-->        "identifier" : "/users",
-->        "permissions" : "domain"
-->     }
-->   }
--> }

<-- HTTP/1.1 204 No Content
```

9.6 Delete a container object using CDMI

9.6.1 Synopsis

To delete an existing container object, including all contained children and snapshots, the following requests shall be performed:

- `DELETE <root URI>/<ContainerName>/<TheContainerName>`
- `DELETE <root URI>/cdmi_objectid/<ContainerObjectID>`

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<ContainerName>` is zero or more intermediate container objects.
- `<TheContainerName>` is the name of the container object to be deleted.
- `<ContainerObjectID>` is the ID of the container object to be deleted.

9.6.2 Capabilities

Capabilities that indicate which operations are supported are shown in Table 64.

Table 64: Capabilities - Delete a CDMI container object using CDMI

Capability	Location	Description
`cdmi_delete_container`	Container object	Ability to delete an existing container object
`cdmi_object_access_by_ID`	System wide capability	Ability to access the object by ID

9.6.3 Request headers

Request headers can be provided as per RFC 2616 [23].

9.6.4 Request message body

A request body can be provided as per RFC 2616 [23].

9.6.5 Response headers

Response headers can be provided as per RFC 2616 [23].

9.6.6 Response message body

A response body can be provided as per RFC 2616 [23].

9.6.7 Response status

Table 65 describes the HTTP status codes that occur when deleting a container object using CDMI.

1967
1968 Table 65: HTTP status codes - Delete a container object using CDMI

HTTP status	Description
204 No Content	The container object was successfully deleted.
400 Bad Request	The request contains invalid parameters or field names.
401 Unauthorized	The authentication credentials are missing or invalid.
403 Forbidden	The client lacks the proper authorization to perform this request.
404 Not Found	The resource was not found at the specified URI.
409 Conflict	The operation conflicts with a non-CDMI access protocol lock or has caused a state transition error on the server.

1969 ### 9.6.8 Example

1970 EXAMPLE 1: DELETE to the container object URI:

```
--> DELETE /cdmi/2.0.0/MyContainer/ HTTP/1.1
--> Host: cloud.example.com

<-- HTTP/1.1 204 No Content
```

1971 EXAMPLE 2: DELETE by container object ID:

```
--> DELETE /cdmi/2.0.0/cdmi_objectid/00007ED900104E1D14771DC67C27BF8B/ HTTP/1.1
--> Host: cloud.example.com

<-- HTTP/1.1 204 No Content
```

9.7 Create (POST) a new data object using CDMI

9.7.1 Synopsis

To create a new data object in a specified container, the following request shall be performed:

- `POST <root URI>/<ContainerName>/`

To create a new data object where the data object does not belong to a container and is only accessible by ID (see 5.3.1), the following request shall be performed:

- `POST <root URI>/cdmi_objectid/`

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<ContainerName>` is zero or more intermediate container objects that already exist, with one slash (i.e., "`/`") between each pair of container object names.
- `<DataObjectName>` is the name specified for the data object to be created.

If created in a container, the data object shall be accessible as a child of the container with a server-assigned name, and shall also be accessible at `<root URI>/cdmi_objectid/<objectID>`.

If created in "`/cdmi_objectid/`", the data object shall only be accessible at `<root URI>/cdmi_objectid/<objectID>`.

9.7.2 Delayed completion of create

In response to a create operation for a data object, the server may return an HTTP status code of `202 Accepted` to indicate that the object is in the process of being created. This response is useful for long-running operations (e.g., copying a large data object from a source URI). Such a response has the following implications.

- The server shall return a `Location` header with an absolute URI to the object to be created along with an HTTP status code of `202 Accepted`.
- With an HTTP status code of `202 Accepted`, the server implies that the following checks have passed:
  - user authorization for creating the object;
  - user authorization for read access to any source object for move, copy, serialize, or deserialize; and
  - availability of space to create the object or at least enough space to create a URI to report an error.
- A client might not be able to immediately access the created object, e.g., due to delays resulting from the implementation's use of eventual consistency.

The client performs GET operations to the URI to track the progress of the operation. In response, the server returns two fields in its response body to indicate progress.

- A mandatory `completionStatus` text field contains either "`Processing`", "`Complete`", or an error string starting with the value "`Error`".
- An optional `percentComplete` field contains the percentage of the operation that has completed (0 to 100).

GET shall not return any value for the data object when `completionStatus` is not "`Complete`". If the final result of the create operation is an error, the URI is created with the `completionStatus` field set to the error message. It is the client's responsibility to delete the URI after the error has been noted.

9.7.3 Capabilities

Capabilities that indicate which operations are supported are shown in Table 66.

Table 66: Capabilities - Create a CDMI data object using CDMI

Capability	Location	Description
cdmi_post_dataobject cdmi_create_dataobject	Parent container	Ability to create a new data object
cdmi_create_reference	Parent container	Ability to create a new reference
cdmi_copy_dataobject	Parent container	Ability to create a data object that is a copy of another data object
cdmi_move_dataobject	Parent container	Ability to move a data object from another container
cdmi_deserialize_dataobject	Parent container	Ability to create a data object that is deserialized from the contents of the PUT or the contents of another data object
cdmi_serialize_dataobject cdmi_serialize_container cdmi_serialize_domain cdmi_serialize_queue	Parent container	Ability to create a data object that contains a serialized representation of an existing data object, container, domain or queue
cdmi_create_value_range	Parent container	Ability to create a data object using a specified byte range
cdmi_post_dataobject_by_ID	System wide capability	Ability to create a new data object in "/cdmi_objectid/"
cdmi_create_reference_by_ID	System wide capability	Ability to create a new reference in "/cdmi_objectid/"
cdmi_copy_dataobject_by_ID	System wide capability	Ability to create a data object in "/cdmi_objectid/" that is a copy of another data object
cdmi_object_move_to_ID	System wide capability	Ability to move a data object to "/cdmi_objectid/" from another container
cdmi_deserialize_dataobject_by_ID	System wide capability	Ability to create a data object in "/cdmi_objectid/" that is deserialized from the contents of the PUT or the contents of another data object
cdmi_serialize_dataobject_to_ID cdmi_serialize_container_to_ID cdmi_serialize_domain_to_ID cdmi_serialize_queue_to_ID	System wide capability	Ability to create a data object in "/cdmi_objectid/" that contains a serialized representation of an existing data object, container, domain or queue
cdmi_create_value_range_by_ID	System wide capability	Ability to create a data object in "/cdmi_objectid/" using a specified byte range
cdmi_multipart_mime	System wide capability	Ability to create a data object using multi-part MIME

9.7.4 Request headers

The HTTP request headers for creating a new CDMI data object using CDMI are shown in Table 67.

Table 67: Request headers - Create a new data object Using CDMI

Header	Type	Description	Requirement
Accept	Header string	"application/cdmi-object" or a consistent value as described in 5.5.2	Optional

Continued on next page

Table 67 – continued from previous page

Header	Type	Description	Requirement
`Content-Type`	Header string	"`application/cdmi-object`" or "`multipart/mixed`" • If "`multipart/mixed`" is specified, the body shall consist of at least two MIME parts, where the first part shall contain a body of content-type "`application/cdmi-object`", and the second and subsequent parts shall contain one or more byte ranges of the value. • If multiple byte ranges are included and the `Content-Range` header is omitted for a part, the data in the part shall be appended to the data in the preceding part, with the first part having a byte offset of zero.	Mandatory
`X-CDMI-Partial`	Header string	Indicates that the newly created object is part of a series of writes and has not yet been fully created. When set to "`true`", the `completionStatus` field shall be set to "`Processing`". `X-CDMI-Partial` works across CDMI and non-CDMI operations.	Optional

9.7.5 Request message body

The request message body fields for creating a new data object using CDMI are shown in Table 68.

Table 68: Request message body - Create a new data object Using CDMI

Field Name	Type	Description	Requirement
`mimetype`	JSON string	MIME type of the data contained within the value field of the data object • This field may be included when creating by value or when deserializing, serializing, copying, and moving a data object. • If this field is not included and multi-part MIME is not being used, the value of "`text/plain`" shall be assigned as the field value. • If this field is not included and multi-part MIME is being used, the value of the `Content-Type` header of the second MIME part shall be assigned as the field value. • This field field value shall be converted to lower case before being stored.	Optional
`metadata`	JSON object	Metadata for the data object • If this field is included, the contents of the JSON object provided in this field shall be used as data object metadata. • If this field is included when deserializing, serializing, copying, or moving a data object, the contents of the JSON object provided in this field shall be used as object metadata instead of the metadata from the source URI. • If this field is not included, no user-specified metadata shall be added to the object. • If this field is not included when deserializing, serializing, copying, or moving a data object, metadata from the source URI shall be used. • This field shall not be included when creating a reference to a data object.	Optional

Continued on next page

Table 68 – continued from previous page

Field Name	Type	Description	Requirement
domainURI	JSON string	URI of the owning domain • If different from the parent domain, the user shall have the "cross-domain" privilege (see cdmi_member_privileges in Table 80 . • If not specified, the domain of the parent container shall be used.	Optional
deserialize	JSON string	URI of a CDMI data object with a value that contains a data object serialized as specified in clause 15. The serialized data object shall be deserialized to create the new data object.	Optional[1]
serialize	JSON string	URI of a CDMI object that shall be serialized into the new data object	Optional[1]
copy	JSON string	URI of a source CDMI data object or queue object that shall be copied into the new destination data object. • If the destination data object URI and the copy source object URI both do not specify individual fields, the destination data object shall be a complete copy of the source data object. • If the destination data object URI or the copy source object URI specifies individual fields, only the fields specified shall be used to create the destination data object. If specified fields are not present in the source, default field values shall be used. • If the destination data object URI and the copy source object URI both specify fields, an HTTP status code of 400 Bad Request shall be returned to the client. • If the copy source object URI points to a queue object, as part of the copy operation, multiple queue values shall be concatenated into a single data object value. • If the copy source object URI points to one or more queue object values, as part of the copy operation, the specified queue values shall be concatenated into a single data object value. • If there are insufficient permissions to read the data object at the source URI or create the data object at the destination URI, or if the read operation fails, the copy shall return an HTTP status code of 400 Bad Request, and the destination object shall not be created.	Optional[1]
move	JSON string	URI of an existing local or remote CDMI data object (source URI) that shall be relocated to the URI specified in the PUT. The contents of the object, including the object ID, shall be preserved by a move, and the data object at the source URI shall be removed after the data object at the destination has been successfully created. If there are insufficient permissions to read the data object at the source URI, write the data object at the destination URI, or delete the data object at the source URI, or if any of these operations fail, the move shall return an HTTP status code of 400 Bad Request, and the source and destination are left unchanged.	Optional[1]
reference	JSON string	URI of a CDMI data object that shall be redirected to by a reference. If any other fields are supplied when creating a reference, the server shall respond with an HTTP status code of 400 Bad Request.	Optional[1]

Continued on next page

Table 68 – continued from previous page

Field Name	Type	Description	Requirement
deserializevalue	JSON string	A data object serialized as specified in clause 15 and encoded using base 64 encoding rules described in RFC 4648 [19], that shall be deserialized to create the new data object. • If multi-part MIME is being used and this field contains the value of the MIME boundary parameter, the contents of the second MIME part shall be assigned as the field value. • If the serialized data object in the second MIME part does not include a value field, the contents of the third MIME part shall be assigned as the field value of the value field.	Optional[1]
valuetransferencoding	JSON string	The value transfer encoding used for the data object value. Three value transfer encodings are defined: • "utf-8" indicates that the data object contains a valid UTF-8 string, and it shall be transported as a UTF-8 string in the value field. • "base64" indicates that the data object may contain arbitrary binary sequences, and it shall be transported as a base 64-encoded string in the value field. Setting the contents of the data object value field to any value other than a valid base 64 string shall result in an HTTP status code of 400 Bad Request being returned to the client. • "json" indicates that the data object contains a valid JSON object, and the value field shall be a JSON object containing valid JSON data. If the contents of the value field are set to any value other than a valid JSON object, an HTTP status code of 400 Bad Request shall be returned to the client. • This field shall only be included when creating a data object by value. • If this field is not included and multi-part MIME is not being used, the value of "utf-8" shall be assigned as the field value. • If this field is not included and multi-part MIME is being used, the value of "utf-8" shall be assigned as the field value if the Content-Type header of the second and all MIME parts includes the charset parameter as defined in RFC 2046 of "utf-8" (e.g., ";charset=utf-8"). Otherwise, the value of "base64" shall be assigned as the field value. This field applies only to the encoding of the value when represented in JSON; the Content-Transfer-Encoding header of the part specifies the encoding of the value within a multi-part MIME request, as defined in RFC 2045 [9].	Optional[1]

Continued on next page

Table 68 – continued from previous page

Field Name	Type	Description	Requirement
`value`	JSON string	The data object value • If this field is not included and multi-part MIME is not being used, an empty JSON String (i.e., “”) shall be assigned as the field value. • If this field is not included and multi-part MIME is being used, the contents of the second MIME part shall be assigned as the field value. • If the `valuetransferencoding` field indicates UTF-8 encoding, the value shall be a UTF-8 string escaped using the JSON escaping rules described in RFC 4627 [5]. • If the `valuetransferencoding` field indicates base 64 encoding, the value shall be first encoded using the base 64 encoding rules described in RFC 4648 [19]. • If the `valuetransferencoding` field indicates JSON encoding, the value shall contain a valid JSON object.	Optional[1]

[1] Only one of these fields shall be specified in any given operation. Except for value, these fields shall not be stored. If more than one of these fields is supplied, the server shall respond with an HTTP status code of `400 Bad Request`.

9.7.6 Response headers

The HTTP response headers for creating a new CDMI data object using CDMI are shown in Table 69.

Table 69: Response headers - Create a new data object using CDMI

Header	Type	Description	Requirement
`Content-Type`	Header string	"`application/cdmi-object`"	Mandatory
`Location`	Header string	The unique absolute URI for the new data object as assigned by the system. In the absence of file name information from the client, the system shall assign the URI in the form: `http://host:port/<root URI>/<ContainerName>/<ObjectID>` or `https://host:port/<root URI>/<ContainerName>/<ObjectID>`.	Mandatory

9.7.7 Response message body

The response message body fields for creating a new CDMI data object using CDMI are shown in Table 70.

Table 70: Response message body - Create a new data object using CDMI

Field Name	Type	Description	Requirement
`objectType`	JSON string	"`application/cdmi-object`"	Mandatory
`objectID`	JSON string	Object ID of the object	Mandatory
`objectName`	JSON string	Name of the object • For objects in a container, the objectName field shall be returned. • For objects not in a container (objects that are only accessible by ID), the objectName field does not exist and shall not be returned.	Conditional
`parentURI`	JSON string	URI for the parent object • For objects in a container, the parentURI field shall be returned. • For objects not in a container (objects that are only accessible by ID), the parentURI field does not exist and shall not be returned. Appending the objectName to the parentURI shall always produce a valid URI for the object.	Conditional
`parentID`	JSON string	Object ID of the parent container object • For objects in a container, the parentID field shall be returned. • For objects not in a container (objects that are only accessible by ID), the parentID field does not exist and shall not be returned.	Conditional
`domainURI`	JSON string	URI of the owning domain	Mandatory
`capabilitiesURI`	JSON string	URI to the capabilities for the object	Mandatory

Continued on next page

Table 70 – continued from previous page

Field Name	Type	Description	Requirement
completionStatus	JSON string	A string indicating if the object is still in the process of being created or updated by another operation, and after that operation is complete, indicates if it was successfully created or updated or if an error occurred. The value shall be the string "Processing", the string "Complete", or an error string starting with the value "Error".	Mandatory
percentComplete	JSON string	A string indicating the percentage of completion if the object is still in the process of bewing created or updated by another operation. • When the value of completionStatus is "Processing", this field, if provided, shall indicate the percentage of completion as a numeric integer value from "0" through "100". • When the value of completionStatus is "Complete", this field, if provided, shall contain the value "100". • When the value of completionStatus is "Error", this field, if provided, may contain any integer value from "0" through "100".	Optional
mimetype	JSON string	MIME type of the value of the data object	Mandatory
metadata	JSON object	Metadata for the data object. This field includes any user and data system metadata specified in the request body metadata field, along with storage system metadata generated by the cloud storage system. See clause 16 for a further description of metadata.	Mandatory

9.7.8 Response status

Table 71 describes the HTTP status codes that occur when creating a new data object using CDMI.

Table 71: HTTP status codes - Create a new data object using CDMI

HTTP status	Description
201 Created	The new data object was created.
202 Accepted	The data object is in the process of being created. The CDMI client should monitor the completionStatus and percentComplete fields to determine the current status of the operation.
400 Bad Request	The request contains invalid parameters or field names.
401 Unauthorized	The authentication credentials are missing or invalid.
403 Forbidden	The client lacks the proper authorization to perform this request.
404 Not Found	The resource was not found at the specified URI.
409 Conflict	The operation conflicts with a non-CDMI access protocol lock or has caused a state transition error on the server.

9.7.9 Examples

EXAMPLE 1: POST to the container object URI the data object contents:

```
--> POST /cdmi/2.0.0/MyContainer/ HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-object
--> Content-Type: application/cdmi-object
-->
--> {
```

(continues on next page)

(continued from previous page)

```
--> "mimetype" : "text/plain",
--> "metadata" : {
-->
--> },
--> "value" : "This is the Value of this Data Object"
--> }

<-- HTTP/1.1 201 Created
<-- Content-Type: application/cdmi-object
<-- Location: https://cloud.example.com/cdmi/2.0.0/MyContainer/
↪00007ED900104E1D14771DC67C27BF8B
<--
<-- {
<--   "objectType" : "application/cdmi-object",
<--   "objectID" : "00007ED900104E1D14771DC67C27BF8B",
<--   "objectName" : "00007ED900104E1D14771DC67C27BF8B",
<--   "parentURI" : "/MyContainer/",
<--   "parentID" : "00007ED900104E1D14771DC67C27BF8B",
<--   "domainURI" : "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI" : "/cdmi_capabilities/dataobject/",
<--   "completionStatus" : "Complete",
<--   "mimetype" : "text/plain",
<--   "metadata" : {
<--     ...
<--   }
<-- }
```

2026 EXAMPLE 2: POST to the object ID URI the data object contents:

```
--> POST /cdmi/2.0.0/cdmi_objectid/ HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-object
--> Content-Type: application/cdmi-object
-->
--> {
--> "mimetype": "text/plain",
--> "domainURI": "/cdmi_domains/MyDomain/",
--> "value": "This is the Value of this Data Object"
--> }

<-- HTTP/1.1 201 Created
<-- Location: https://cloud.example.com/cdmi/2.0.0/cdmi_objectid/
↪00007ED900104E1D14771DC67C27BF8B
<-- Content-Type: application/cdmi-object
<--
<-- {
<--   "objectType": "application/cdmi-object",
<--   "objectID": "00007ED900104E1D14771DC67C27BF8B",
<--   "domainURI": "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI": "/cdmi_capabilities/dataobject/",
<--   "completionStatus": "Complete",
<--   "mimetype": "text/plain",
<--   "metadata": {
<--     "cdmi_acl": [
<--       {
<--         "acetype": "ALLOW",
<--         "identifier": "OWNER@",
<--         "aceflags": "NO_FLAGS",
<--         "acemask": "ALL_PERMS"
<--       }
<--     ],
<--     ...
<--   }
<-- }
```

2027 EXAMPLE 3: POST to the object ID URI the data object fields and binary contents using multi-part MIME:

```
--> POST /cdmi/2.0.0/cdmi_objectid/ HTTP/1.1
```

(continues on next page)

(continued from previous page)

```
--> Host: cloud.example.com
--> Accept: application/cdmi-object
--> Content-Type: multipart/mixed; boundary=gc0p4Jq0M2Yt08j34c0p
-->
--> --gc0p4Jq0M2Yt08j34c0p
--> Content-Type: application/cdmi-object
-->
--> {
--> "domainURI": "/cdmi_domains/MyDomain/",
--> "metadata": {
--> "colour": "blue"
--> }
--> }
-->
--> --gc0p4Jq0M2Yt08j34c0p
--> Content-Type: application/octet-stream
--> Content-Transfer-Encoding: binary
-->
--> <37 bytes of binary data>
-->
--> --gc0p4Jq0M2Yt08j34c0p--

<-- HTTP/1.1 201 Created
<-- Location: https://cloud.example.com/cdmi/2.0.0/cdmi_objectid/
↪00007ED90010C2414303B5C6D4F83170
<-- Content-Type: application/cdmi-object
<--
<-- {
<--   "objectType": "application/cdmi-object",
<--   "objectID": "00007ED90010C2414303B5C6D4F83170",
<--   "domainURI": "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI": "/cdmi_capabilities/dataobject/",
<--   "completionStatus": "Complete",
<--   "mimetype": "application/octet-stream",
<--   "metadata": {
<--     "cdmi_size": "37",
<--     "colour": "blue",
<--     ...
<--   }
<-- }
```

9.8 Create (POST) a new queue object using CDMI

9.8.1 Synopsis

To create a new queue object (see clause 11) in a specified container where the name of the queue object is a server-assigned object identifier, the following request shall be performed:

- `POST <root URI>/<ContainerName>/`

To create a new queue object where the queue object does not belong to a container and is only accessible by ID (see 5.3.1), the following request shall be performed:

- `POST <root URI>/cdmi_objectid/`

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<ContainerName>` is zero or more intermediate container objects that already exist, with one slash (i.e., "`/`") between each pair of container object names.

If created in a container, the queue object shall be accessible as a child of the container with a server-assigned name, and shall also be accessible at `<root URI>/cdmi_objectid/<objectID>`.

If created in "`/cdmi_objectid/`", the queue object shall only be accessible at `<root URI>/cdmi_objectid/<objectID>`.

9.8.2 Delayed completion of create

In response to a create operation for a queue object, the server may return an HTTP status code of `202 Accepted` to indicate that the object is in the process of being created. This response is useful for long-running operations (e.g., copying a large number of queue values from a source URI). Such a response has the following implications.

- The server shall return a `Location` header with an absolute URI to the object to be created along with an HTTP status code of `202 Accepted`.
- With an HTTP status code of `202 Accepted`, the server implies that the following checks have passed:
  - user authorization for creating the object;
  - user authorization for read access to any source object for move, copy, serialize, or deserialize; and
  - availability of space to create the object or at least enough space to create a URI to report an error.
- A client might not be able to immediately access the created object, e.g., due to delays resulting from the implementation's use of eventual consistency.

The client performs GET operations to the URI to track the progress of the operation. In response, the server returns two fields in its response body to indicate progress.

- A mandatory `completionStatus` text field contains either "`Processing`", "`Complete`", or an error string starting with the value "`Error`".
- An optional `percentComplete` field contains the percentage of the operation that has completed (0 to 100).

GET shall not return any value for the queue object when `completionStatus` is not "`Complete`". If the final result of the create operation is an error, the URI is created with the `completionStatus` field set to the error message. It is the client's responsibility to delete the URI after the error has been noted.

9.8.3 Capabilities

Capabilities that indicate which operations are supported are shown in Table 72.

Table 72: Capabilities - Create a CDMI Queue object using CDMI

Capability	Location	Description
cdmi_post_queue cdmi_create_queue	Parent container	Ability to create a new queue object
cdmi_create_reference	Parent container	Ability to create a new reference
cdmi_copy_queue	Parent container	Ability to create a queue object that is a copy of another queue object
cdmi_move_queue	Parent container	Ability to move a queue object from another container
cdmi_deserialize_queue	Parent container	Ability to create a queue object that is deserialized from the contents of the PUT or the contents of another queue object
cdmi_post_queue_by_ID	System wide capability	Ability to create a new queue object in "/cdmi_objectid/"
cdmi_create_reference_by_ID	System wide capability	Ability to create a new reference in "/cdmi_objectid/"
cdmi_copy_queue_by_ID	System wide capability	Ability to create a queue object in "/cdmi_objectid/" that is a copy of another queue object
cdmi_object_move_to_ID	System wide capability	Ability to move a queue object to "/cdmi_objectid/" from another container
cdmi_deserialize_queue_by_ID	System wide capability	Ability to create a queue object in "/cdmi_objectid/" that is deserialized from the contents of the PUT or the contents of another data object
cdmi_serialize_dataobject_to_ID cdmi_serialize_container_to_ID cdmi_serialize_domain_to_ID cdmi_serialize_queue_to_ID	System wide capability	Ability to create a data object in "/cdmi_objectid/" that contains a serialized representation of an existing data object, container, domain or queue

9.8.4 Request headers

The HTTP request headers for creating a new CDMI queue object using CDMI are shown in Table 73.

Table 73: Request headers - Create a new queue object using CDMI

Header	Type	Description	Requirement
Accept	Header string	"application/cdmi-object" or a consistent value as described in 5.5.2	Optional
Content-Type	Header string	"application/cdmi-queue"	Mandatory
Content-Range	Header string	A valid ranges-specifier (see RFC 2616 [23] Section 14.35.1)	Optional

9.8.5 Request message body

The request message body fields for creating a new queue object using CDMI are shown in Table 74.

Table 74: Request message body - Create a new queue object using CDMI

Field Name	Type	Description	Requirement
`metadata`	JSON object	Metadata for the queue object • If this field is included, the contents of the JSON object provided in this field shall be used as queue object metadata. • If this field is included when deserializing, serializing, copying, or moving a queue object, the contents of the JSON object provided in this field shall be used as object metadata instead of the metadata from the source URI. • If this field is not included, no user-specified metadata shall be added to the object. • If this field is not included when deserializing, serializing, copying, or moving a queue object, metadata from the source URI shall be used. • This field shall not be included when creating a reference to a queue object.	Optional
`domainURI`	JSON string	URI of the owning domain • If different from the parent domain, the user shall have the "`cross-domain`" privilege (see `cdmi_member_privileges` in Table 80 . • If not specified, the domain of the parent container shall be used.	Optional
`deserialize`	JSON string	URI of a CDMI data object with a value that contains a queue object serialized as specified in clause 15. The serialized queue object shall be deserialized to create the new queue object.	Optional[1]
`copy`	JSON string	URI of a CDMI queue object that will be copied into the new queue object	Optional[1]
`move`	JSON string	URI of a CDMI queue object that will be copied into the new queue object. When the copy is successfully completed, the queue object at the source URI is removed.	Optional[1]
`reference`	JSON string	URI of a CDMI queue object that shall be redirected to by a reference. If other fields are supplied when creating a reference, the server shall respond with an HTTP status code of `400 Bad Request`.	Optional[1]
`deserializevalue`	JSON string	A queue object serialized as specified in clause 15 and encoded using base 64 encoding rules described in RFC 4648 [19], that shall be deserialized to create the new queue object.	Optional[1]

[1] Only one of these fields shall be specified in any given operation. Except for value, these fields shall not be stored. If more than one of these fields is supplied, the server shall respond with an HTTP status code of `400 Bad Request`.

9.8.6 Response headers

The response headers for creating a new CDMI queue object using CDMI are shown in Table 75.

Table 75: Response headers - Create a new queue object using CDMI

Header	Type	Description	Requirement
`Content-Type`	Header string	"`application/cdmi-queue`"	Mandatory
`Location`	Header string	The unique absolute URI for the new queue object as assigned by the system.	Mandatory

9.8.7 Response message body

The response message body fields for creating a new CDMI queue object using CDMI are shown in Table 76.

Table 76: Response message body - Create a new queue object using CDMI

Field Name	Type	Description	Requirement
`objectType`	JSON string	"`application/cdmi-queue`"	Mandatory
`objectID`	JSON string	Object ID of the object	Mandatory
`objectName`	JSON string	Name of the object • For objects in a container, the objectName field shall be returned. • For objects not in a container (objects that are only accessible by ID), the objectName field does not exist and shall not be returned.	Conditional
`parentURI`	JSON string	URI for the parent object • For objects in a container, the parentURI field shall be returned. • For objects not in a container (objects that are only accessible by ID), the parentURI field does not exist and shall not be returned. Appending the objectName to the parentURI shall always produce a valid URI for the object.	Conditional
`parentID`	JSON string	Object ID of the parent container object • For objects in a container, the parentID field shall be returned. • For objects not in a container (objects that are only accessible by ID), the parentID field does not exist and shall not be returned.	Conditional
`domainURI`	JSON string	URI of the owning domain	Mandatory
`capabilitiesURI`	JSON string	URI to the capabilities for the object	Mandatory
`completionStatus`	JSON string	A string indicating if the object is still in the process of being created or updated by another operation, and after that operation is complete, indicates if it was successfully created or updated or if an error occurred. The value shall be the string "`Processing`", the string "`Complete`", or an error string starting with the value "`Error`".	Mandatory

Continued on next page

Table 76 – continued from previous page

Field Name	Type	Description	Requirement
percentComplete	JSON string	A string indicating the percentage of completion if the object is still in the process of bewing created or updated by another operation. • When the value of completionStatus is "Processing", this field, if provided, shall indicate the percentage of completion as a numeric integer value from "0" through "100". • When the value of completionStatus is "Complete", this field, if provided, shall contain the value "100". • When the value of completionStatus is "Error", this field, if provided, may contain any integer value from "0" through "100".	Optional
metadata	JSON object	Metadata for the queue object. This field includes any user and data system metadata specified in the request body metadata field, along with storage system metadata generated by the cloud storage system. See clause 16 for a further description of metadata.	Mandatory
queueValues	JSON string	The range of designators for enqueued values. Every enqueued value shall be assigned a unique, monotonically-incrementing positive integer designator, starting from 0. If no values are enqueued, an empty string shall be returned. If values are enqueued, the lowest designator, followed by a hyphen ("-"), followed by the highest designator shall be returned.	Mandatory

9.8.8 Response status

Table 77 describes the HTTP status codes that occur when creating a new queue object using CDMI.

Table 77: HTTP status codes - Create a new queue object using CDMI

HTTP status	Description
201 Created	The new queue object was created.
202 Accepted	The queue object is in the process of being created. The CDMI client should monitor the completionStatus and percentComplete fields to determine the current status of the operation.
400 Bad Request	The request contains invalid parameters or field names.
401 Unauthorized	The authentication credentials are missing or invalid.
403 Forbidden	The client lacks the proper authorization to perform this request.
404 Not Found	The resource was not found at the specified URI.
409 Conflict	The operation conflicts with a non-CDMI access protocol lock or has caused a state transition error on the server.

9.8.9 Example

EXAMPLE 1: POST to the container object URI the queue object contents:

```
--> POST /cdmi/2.0.0/MyContainer/ HTTP/1.1
--> Host: cloud.example.com
--> ``Content-Type: application/cdmi-queue``
--> Accept: application/cdmi-queue
-->
--> {
--> }

<-- HTTP/1.1 201 Created
<-- Content-Type: application/cdmi-queue
<-- Location: https://cloud.example.com/cdmi/2.0.0/MyContainer/
↪00007ED900104E1D14771DC67C27BF8B
<--
<-- {
<--   "objectType" : "application/cdmi-queue",
<--   "objectID" : "00007ED900104E1D14771DC67C27BF8B",
<--   "objectName" : "00007ED900104E1D14771DC67C27BF8B",
<--   "parentURI" : "/MyContainer/",
<--   "parentID" : "00007ED900104E1D14771DC67C27BF8B",
<--   "domainURI" : "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI" : "/cdmi_capabilities/queue/",
<--   "completionStatus" : "Complete",
<--     "metadata" : {
<--     ...
<--   },
<--   "queueValues" : ""
<-- }
```

Part IV

CDMI Advanced

Clause 10

Domain object resource operations using CDMI

10.1 Overview

Domain objects represent the concept of administrative ownership of stored data within a CDMI™ storage system. Each object may be owned and managed by a different administrative entity, which is expressed as a domain.

If a cloud storage system supports domains, the `cdmi_domains` system-wide capability shall be present, and the `cdmi_domains` container shall be present in the CDMI root container.

A cloud storage system may include a hierarchy of domains that provide access to domain-related information within a CDMI context. This domain hierarchy is a series of CDMI objects that correspond to parent and child domains, with each domain corresponding to logical groupings of objects that are to be managed together. Domain measurement information about objects that are associated with each domain flow up to parent domains, facilitating billing and management operations that are typical for a cloud storage environment.

Fig. 7 shows the hierarchy of domains and shows how the domainURI links data objects, container objects and queue objects into the domain hierachy.

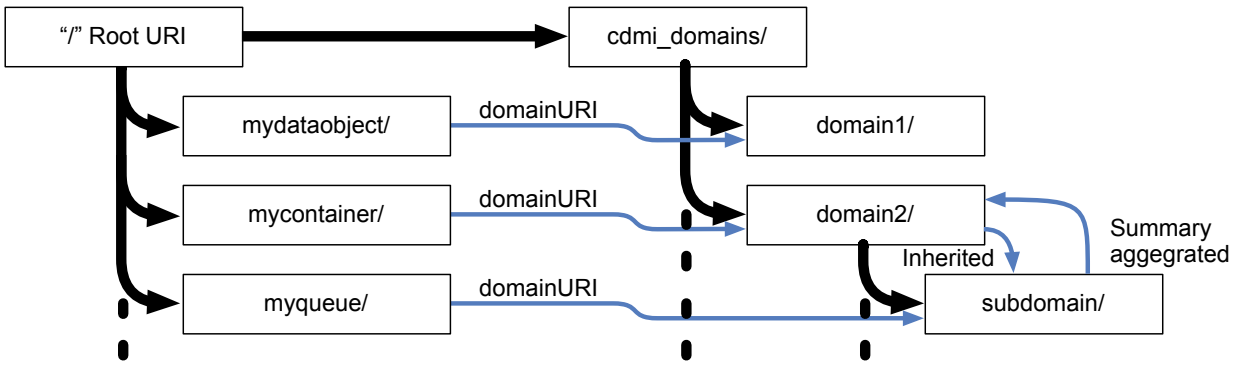


Fig. 7: Hierarchy of domains

Each CDMI domain object is represented as a JSON object, containing one or more "fields". For example, the "`metadata`" field contains metadata items.

EXAMPLE 1: CDMI domain object

```
{
    "objectType" : "application/cdmi-domain",
    "objectID" : "00007E7F00104BE66AB53A9572F9F51E",
    "objectName" : "MyDomain/",
    "parentURI" : "/cdmi_domains/",
    "parentID" : "00007E7F0010C058374D08B0AC7B3550",
```

(continues on next page)

(continued from previous page)

```
    "domainURI" : "/cdmi_domains/MyDomain/",
    "capabilitiesURI" : "/cdmi_capabilities/domain/",
    "metadata" : {
        "cdmi_domain_enabled": "true",
        "cdmi_authentication_methods": "anonymous, basic",
        ...
    },
    "childrenrange" : "0-1",
    "children" : [
        "cdmi_domain_summary/",
        "cdmi_domain_members/"
    ]
}
```

The meaning, use, and permitted values of each field is described in each operation that creates, modifies or retreives CDMI domain objects.

10.2 Domain object details

10.2.1 Domain object addressing

Domain objects are created as children of a special `cdmi_domains` container object, which is present in the root URI for the cloud storage system when domains are supported. The `cdmi_domains` container object is system-generated, read-only, cannot be deleted, and only permits the creation of children domain objects, as indicated by the presence of the `cdmi_create_domain` capability. The ability to create a sub-domain under an existing domain object is indicated by the presence of the `cdmi_create_domain` capability for a given domain object.

Domain objects are addressed in CDMI in two ways:

- by name (e.g., `https://cloud.example.com/cdmi/2.0.0/cdmi_domains/myDomain/`); and
- by ID (e.g., `https://cloud.example.com/cdmi/2.0.0/cdmi_objectid/00007ED90010329E642EBFBC8B57E9AD/`).

Every domain object has a single, globally-unique object ID that remains constant for the life of the object. Each domain object shall also have at least one URI address that allows the domain object to be accessed. Following the URI conventions for hierarchical paths, domain URIs shall start with "`<root URI>/cdmi_domains/`" and consist of one or more domain names that are separated by forward slashes ("`/`") and that end with a forward slash ("`/`").

If a request is performed against an existing domain resource and the trailing slash at the end of the URI is omitted, the server shall respond with an HTTP status code of `301 Moved Permanently`, and a ''Location'' header containing the URI with the trailing slash will be added.

If a CDMI request is performed to create a new domain resource and the trailing slash at the end of the URI is omitted, the server shall respond with an HTTP status code of `400 Bad Request`.

Domain objects may also be nested.

EXAMPLE 2: The following URI represents a nested domains:

```
https://cloud.example.com/cdmi/2.0.0/cdmi_domains/myDomain/subDomain/
```

A sub-domain has a parent domain object, shall be included in the children field of the parent domain object, and shall inherit Domain Membership from its parent domain (if not specified in the sub-domain).

10.2.2 Domain object fields

Individual fields within a domain object may be accessed by specifying the field name after a question mark "`?`" appended to the end of the domain object URI.

EXAMPLE 3: The following URI returns just the children field in the response message body:

```
https://cloud.example.com/cdmi/2.0.0/cdmi_domains/myDomain/?children
```

EXAMPLE 4: By specifying a range after the children field name, specific ranges of the children field may be accessed.

```
https://cloud.example.com/cdmi/2.0.0/cdmi_domains/myDomain/?
children=0-2
```

Children ranges are specified in a way that is similar to byte ranges as per Section 14.35.1 of RFC 2616 [23]. A client can determine the number of children present by requesting the childrenrange field without requesting a range of children.

A list of fields separated by an ampersand "`&`" may be specified, allowing multiple fields to be accessed in a single request.

EXAMPLE 5: The following URI would return the children and metadata fields in the response body:

```
https://cloud.example.com/cdmi/2.0.0/cdmi_domains/myDomain/?children;
metadata
```

When a client provides fields that are not defined in this International Standard or deserializes an object containing fields that are not defined in this International Standard, these fields shall be persisted, but shall not be interpreted.

10.2.3 Domain object metadata

The following domain-specific field shall be present for each domain (see Table 78).

Table 78: Required metadata for a domain object

Metadata name	Type	Description	Requirement
`cdmi_domain_enabled`	JSON string	Indicates if the domain is enabled and specified at the time of creation. Values shall be "`true`" or "`false`". • If this metadata item is not present at the time of domain creation, the value is set to "`false`". • If a domain is disabled, the cloud storage system shall not permit any operations to be performed against any URI managed by that domain. • When a domain is disabled, all operations that are performed against URIs that are managed by a disabled domain shall return an HTTP status code of 403 Forbidden.	Mandatory
`cdmi_domain_delete_reassign`	JSON string	If the domain is deleted, indicates to which domain the objects that belong to the domain shall be reassigned. • To delete a domain that contains objects, this metadata item shall be present. • If this metadata item is not present or does not contain the URI of a valid domain that is different from the URI of the domain being deleted, an attempt to delete a domain that has objects shall result in an HTTP status code of `400 Bad Request`.	Conditional
`cdmi_authentication_methods`	JSON array of JSON strings	Indicates a list of which authentication methods are enabled for the domain. Supported authentication method values are indicated by the cdmi_authentication_methods capability.	Optional

Domains may also contain domain-specific data system metadata items as defined in 16.3 and 16.4. Domain data system metadata shall be inherited to child domain objects.

10.2.4 Domain object access control

If read access to any of the requested fields is not permitted by the object ACL, only the permitted fields shall be returned. If no requested fields are permitted to be read, an HTTP status code of `403 Forbidden` shall be returned to the client.

If write access to any of the requested fields is not permitted by the object ACL, no updates shall be performed, and an HTTP status code of `403 Forbidden` shall be returned to the client.

10.2.5 Domain usage in access control

When a transaction is performed against a CDMI object, the associated domain object (i.e., the domain object indicated by the `domainURI`) specifies the authentication context. The user identity and credentials presented as part of the transaction are compared to the domain membership list to determine if the user is authorized within the domain and to resolve the user's principal. If resolved, the user's principal is evaluated against the object's ACL to determine if the transaction is permitted.

When evaluating members within a domain, delegations are evaluated first, in any order, followed by user records, in any order. If there is at least one matching record and none of the matching records indicate that the user is disabled, the user is considered to be a member of the domain.

When a sub-domain is initially created, the membership container contains one member record that is a delegation in which the delegation URI is set to the URI of the parent domain.

10.2.6 Domain object representations

The representations in this clause are shown using JSON notation. Both clients and servers shall support UTF-8 JSON representation. The request and response body JSON fields may be specified or returned in any order, with the exception that, if present, for domain objects, the childrenrange and children fields shall appear last and in that order.

10.3 Domain object summaries

Domain object summaries provide summary measurement information about domain usage and billing. If supported, a domain summary container named "cdmi_domain_summary" shall be present under each domain container. Like any container, the domain summary subcontainer may have an Access Control List (ACL) (see 17.1) that restricts access to this information.

Within each domain summary container are a series of domain summary data objects that are generated by the cloud storage system. The "yearly", "monthly", and "daily" containers of these data objects contain domain summary data objects corresponding to each year, month, and day, respectively. These containers are organized into the following structures:

```
https://example.com/cdmi/2.0.0/cdmi_domains/domain/

https://example.com/cdmi/2.0.0/cdmi_domains/domain/cdmi_domain_summary/

https://example.com/cdmi/2.0.0/cdmi_domains/domain/cdmi_domain_summary/
cumulative

https://example.com/cdmi/2.0.0/cdmi_domains/domain/cdmi_domain_summary/daily/

https://example.com/cdmi/2.0.0/cdmi_domains/domain/cdmi_domain_summary/daily/
2009-07-01

https://example.com/cdmi/2.0.0/cdmi_domains/domain/cdmi_domain_summary/daily/
2009-07-02

https://example.com/cdmi/2.0.0/cdmi_domains/domain/cdmi_domain_summary/daily/
2009-07-03

https://example.com/cdmi/2.0.0/cdmi_domains/domain/cdmi_domain_summary/
monthly/

https://example.com/cdmi/2.0.0/cdmi_domains/domain/cdmi_domain_summary/
monthly/2009-07

https://example.com/cdmi/2.0.0/cdmi_domains/domain/cdmi_domain_summary/
monthly/2009-08

https://example.com/cdmi/2.0.0/cdmi_domains/domain/cdmi_domain_summary/
monthly/2009-10

https://example.com/cdmi/2.0.0/cdmi_domains/domain/cdmi_domain_summary/
yearly/

https://example.com/cdmi/2.0.0/cdmi_domains/domain/cdmi_domain_summary/
yearly/2009

https://example.com/cdmi/2.0.0/cdmi_domains/domain/cdmi_domain_summary/
yearly/2010
```

The "cumulative" summary data object covers the entire time period, from the time the domain is created to the time it is accessed. Each data object at the daily, monthly, and yearly level contains domain summary information for the time period specified, bounded by domain creation time and access time.

If a time period extends earlier than the domain creation time, the summary information includes the time from when the domain was created until the end of the time period.

EXAMPLE 1: If a domain were created on July 4, 2009, at noon, the daily summary "2009-07-04" would contain information from noon until midnight, the monthly summary "2009-07" would contain information from noon on July 4 until midnight on July 31, and the yearly summary "2009" would contain information from noon on July 4 until midnight on December 31.

If a time period starts after the time when the domain was created and ends earlier than the time of access, the summary data object contains complete information for that time period.

EXAMPLE 2: If a domain were created on July 4, 2009, and on July 10, the "2009-07-06" daily summary data object was accessed, it would contain information for the complete day.

If a time period ends after the current access time, the domain summary data object contains partial information from the start of the time period (or the time the domain was created) until the time of access.

EXAMPLE 3: If a domain were created on July 4, 2009, and at noon on July 10, the "2009-07-10" daily summary data object was accessed, it would contain information from the beginning of the day until noon.

The information in Table 79 shall be present within the contents of each domain summary object, which are in JSON representation.

Table 79: Contents of domain summary objects

Metadata name	Type	Description	Requirement
cdmi_domainURI	JSON string	Domain name corresponding to the domain that is summarized	Mandatory
cdmi_summary_start	JSON string	An ISO-8601 time indicating the start of the time range that the summary information is presenting	Mandatory
cdmi_summary_end	JSON string	An ISO-8601 time indicating the end of the time range that the summary information is presenting	Mandatory
cdmi_summary_objecthours	JSON string	The sum of the time each object belonging to the domain existed during the summary time period	Optional
cdmi_summary_objectsmin	JSON string	The minimum number of objects belonging to the domain during the summary time period	Optional
cdmi_summary_objectsmax	JSON string	The maximum number of objects belonging to the domain during the summary time period	Optional
cdmi_summary_objectsaverage	JSON string	The average number of objects belonging to the domain during the summary time period	Optional
cdmi_summary_puts	JSON string	The number of objects written to the domain	Optional
cdmi_summary_gets	JSON string	The number of objects read from the domain	Optional
cdmi_summary_bytehours	JSON string	The sum of the time each byte belonging to the domain existed during the summary time period	Optional
cdmi_summary_bytesmin	JSON string	The minimum number of bytes belonging to the domain during the summary time period	Optional
cdmi_summary_bytesmax	JSON string	The maximum number of bytes belonging to the domain during the summary time period	Optional
cdmi_summary_bytesaverage	JSON string	The average number of bytes belonging to the domain during the summary time period	Optional
cdmi_summary_writes	JSON string	The number of bytes written to the domain	Optional
cdmi_summary_reads	JSON string	The number of bytes read from the domain	Optional
cdmi_summary_charge	JSON string	An ISO 4217 currency code (see [38]) that is followed or preceded by a numeric value and separated by a space, where the numeric value represents the closing charge in the indicated currency for the use of the service associated with the domain over the summary time period	Optional
cdmi_summary_kwhours	JSON string	The sum of energy consumed (in kilowatt hours) by the domain during the summary time period	Optional

Continued on next page

Table 79 – continued from previous page

Metadata name	Type	Description	Requirement
cdmi_summary_kwmin	JSON string	The minimum rate at which energy is consumed (in kilowatt hours per hour) by the domain during the summary time period	Optional
cdmi_summary_kwmax	JSON string	The maximum rate at which energy is consumed (in kilowatt hours per hour) by the domain during the summary time period	Optional
cdmi_summary_kwaverage	JSON string	The average rate at which energy is consumed (in kilowatt hours per hour) by the domain during the summary time period	Optional

EXAMPLE 4: An example of a daily domain summary object is as follows:

```
{
    "cdmi_domainURI" : "/cdmi_domains/MyDomain/",
    "cdmi_summary_start" : "2009-12-10T00:00:00",
    "cdmi_summary_end" : "2009-12-10T23:59:59",
    "cdmi_summary_objecthours" : "382239734",
    "cdmi_summary_puts" : "234234",
    "cdmi_summary_gets" : "489432",
    "cdmi_summary_bytehours" : "334895798347",
    "cdmi_summary_writes" : "7218368343",
    "cdmi_summary_reads" : "11283974933",
    "cdmi_summary_charge" : "4289.23 USD"
}
```

If the charge value is provided, the value is for the operational cost (excluding fixed fees) of service already performed and storage and bandwidth already consumed. Pricing of services is handled separately.

Domain summary information may be extended by vendors to include additional metadata or domain reports beyond the metadata items specified by this International Standard, as long as the field names for those metadata items do not begin with "cdmi_".

10.4 Domain object membership

In cloud storage environments, in the same way that domains are often created programmatically, domain user membership and credential mapping also shall be populated using such interfaces. By providing access to user membership, this capability enables self-enrollment, automatic provisioning, and other advanced self-service capabilities, either directly using CDMI or through software systems that interface with CDMI.

The domain membership capability provides information about, and allows the specification of, end users and groups of users that are allowed to access the domain via CDMI and other access protocols. The concept of domain membership is not intended to replace or supplant ACLs (see 17.1), but rather to provide a single, unified place to map identities and credentials to principals used by ACLs within the context of a domain (see model described in 10.2.5). It also provides a place for authentication mappings to external authentication providers, such as LDAP and Active Directory, to be specified.

If supported, a domain membership container named cdmi_domain_members shall be present under each domain. Like any container, the domain membership container has an Access Control List (see 17.1) that restricts access to this information.

Within each domain membership container are a series of user objects that are specified through CDMI to define each user known to the domain. These objects are formatted into the following structure:

```
https://example.com/cdmi/2.0.0/cdmi_domains/domain/
https://example.com/cdmi/2.0.0/cdmi_domains/domain/cdmi_domain_members/
https://example.com/cdmi/2.0.0/cdmi_domains/domain/cdmi_domain_members/
john_doe
https://example.com/cdmi/2.0.0/cdmi_domains/domain/cdmi_domain_members/
john_smith
```

The domain membership container may also contain subcontainers with data objects. Data objects in these subcontainers are treated the same as data objects in the domain membership container, and no meaning is inferred from the subcontainer name. This organization is used to create different access security relationships for groups of user objects and to allow delegation to a common set of members.

Table 80 lists the domain settings that shall be present within each domain member user object.

Table 80: Required settings for domain member user objects

Metadata name	Type	Description	Requirement
cdmi_member_enabled	JSON string	If true, this field indicates that requests associated with this domain member are allowed. If false, all requests performed by this domain member shall result in an HTTP status code of 403 Forbidden.	Mandatory
cdmi_member_type	JSON string	This field indicates the type of member record. Values include "user", "group", and "delegation".	Mandatory
cdmi_member_name	JSON string	This field contains the user or group name as presented by the client. This will normally be the standard full name of the principal.	Mandatory
cdmi_member_credentials	JSON string	This field contains credentials to be matched against the credentials as presented by the client. If this field is not present, one or more delegations shall be present and shall be used to resolve user credentials. As one cannot log in as a group but only as a member of a group, the "group" type member records shall not have credentials.	Optional

Continued on next page

Table 80 – continued from previous page

Metadata name	Type	Description	Requirement
cdmi_member_principal	JSON string	This field indicates to which principal name (used in ACLs) the user or group is mapped. If this field is not present, one or more delegations shall be present and shall be used to resolve the principal.	Optional
cdmi_member_privileges	JSON array of JSON strings	This field explicitly confers zero or more special privileges to a user or group. When delegated, privileges are conferred based on the information returned from the external system to which the delegation points. The following privileges are defined: • "administrator". Allows the principal to take ownership of any object/container. • "backup_operator". Bypass regular ACL checks to allow backup and restore of objects and containers, including all associated attributes, metadata, ACLs and ownership. • "cross_domain". Operations specifying a domain other than the domain of the parent object are permitted. Unless this privilege is conferred by the user record or a group (possibly nested) to which the user or group belongs, all attempts to change the domain of objects to a domain other than the parent domain shall fail.	Mandatory
cdmi_member_groups	JSON array of JSON strings	This field contains a JSON array of group names to which the user or group belongs.	Optional

2259 Table 81 lists the domain settings that shall be present within each domain member delegation object.

Table 81: Required settings for domain member delegation objects

Metadata name	Type	Description	Requirement
cdmi_member_enabled	JSON string	If true, this field indicates that requests associated with this domain member are allowed. If false, all requests performed by this domain member shall result in an HTTP status code of 403 Forbidden.	Mandatory
cdmi_member_type	JSON string	This field indicates the type of member record. Values include "user" and "delegation".	Mandatory
cdmi_delegation_URI	JSON string	This field contains the URI of an external identity resolution provider (such as LDAP or Active Directory) or the URI of a domain membership container object. External delegations are expressed in the form of ldap://<uri> or ad://<uri>.	Mandatory

2260 EXAMPLE 1: An example of a domain membership object for a user is as follows:

```
{
    "cdmi_member_enabled" : "true",
    "cdmi_member_type" : "user",
    "cdmi_member_name" : "John Doe",
    "cdmi_member_credentials" : "p+5/oX1cmExfOIrUxhX1lw==",
    "cdmi_member_groups" : [
        "users"
    ],
    "cdmi_member_principal" : "jdoe",
    "cdmi_privileges" : [
        "administrator",
        "cross_domain"
    ]
}
```

2261 EXAMPLE 2: An example of a domain membership object for a delegation is as follows:

```
{
    "cdmi_member_enabled" : "true",
    "cdmi_member_type" : "delegation",
    "cdmi_delegation_URI" : "/cdmi_domains/MyDomain/"
}
```

10.5 Create a domain object using CDMI

10.5.1 Synopsis

To create a new domain object, the following request shall be performed:

- `PUT <root URI>/cdmi_domains/<DomainName>/<NewDomainName>/`

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<DomainName>` is zero or more intermediate domains that already exist, with one slash (i.e., "`/`") between each pair of domain names.
- `<NewDomainName>` is the name specified for the domain to be created.

After it is created, the domain shall also be accessible at `<root URI>/cdmi_objectid/<objectID>/`.

10.5.2 Delayed completion of create

Delayed completion shall not be supported for creating domain objects.

10.5.3 Capabilities

Capabilities that indicate which operations are supported are shown in Table 82.

Table 82: Capabilities - Create a CDMI domain object using CDMI

Capability	Location	Description
`cdmi_create_domain`	Parent container	Ability to create a new domain object
`cdmi_copy_domain`	Parent container	Ability to create a domain object that is a copy of another domain object
`cdmi_deserialize_domain`	Parent container	Ability to create a domain object that is deserialized from the contents of the PUT or the contents of another data object

10.5.4 Request headers

The HTTP request headers for creating a CDMI domain object using CDMI are shown in Table 83

Table 83: Request headers - Create a domain object using CDMI

Header	Type	Description	Requirement
`Accept`	Header string	"`application/cdmi-domain`" or a consistent value as described in 5.5.2	Optional
`Content-Type`	Header string	"`application/cdmi-domain`"	Mandatory

10.5.5 Request message body

The request message body fields for creating a domain object using CDMI are shown in Table 84.

Table 84: Request message body - Create a domain object using CDMI

Field Name	Type	Description	Requirement
`metadata`	JSON object	Metadata for the domain object • If this field is included, the contents of the JSON object provided in this field shall be used as domain object metadata. • If this field is included when deserializing, serializing, copying, or moving a domain object, the contents of the JSON object provided in this field shall be used as object metadata instead of the metadata from the source URI. • If this field is not included, no user-specified metadata shall be added to the object. • If this field is not included when deserializing, serializing, copying, or moving a domain object, metadata from the source URI shall be used.	Optional
`copy`	JSON string	URI of a CDMI domain that shall be copied into the new domain, including all child domains and membership from the source domain	Optional[1]
`move`	JSON string	URI of an existing local CDMI domain object (source URI) that shall be relocated, along with all child domains, to the URI specified in the PUT. The contents of the domain and all sub-domains, including the object ID, shall be preserved by a move, and the domain and sub-domains of the source URI shall be removed after the objects at the destination have been successfully created. If there are insufficient permissions to read the objects at the source URI, write the objects at the destination URI, or delete the objects at the source URI, or if any of these operations fail, the move shall return an HTTP status code of `400 Bad Request`, and the source and destination are left unchanged.	Optional[1]
`deserialize`	JSON string	URI of a CDMI data object with a value that contains a domain object serialized as specified in clause 15. The serialized domain object shall be deserialized to create the new domain object, including all child objects.	Optional[1]
`deserializevalue`	JSON string	A domain object serialized as specified in clause 15 and encoded using base 64 encoding rules described in RFC 4648 [19], that shall be deserialized to create the new domain object, including all child objects.	Optional[1]

10.5.6 Response headers

The HTTP response headers for creating a domain object using CDMI are shown in Table 85

Table 85: Response headers - Create a domain object using CDMI

Header	Type	Description	Requirement
`Content-Type`	Header string	“`application/cdmi-domain`”	Mandatory

[1] Only one of these fields shall be specified in any given operation. Except for value, these fields shall not be stored. If more than one of these fields is supplied, the server shall respond with an HTTP status code of `400 Bad Request`.

10.5.7 Response message body

The response message body fields for creating a domain object using CDMI are shown in Table 86

Table 86: Response message body - Create a domain object using CDMI

Field Name	Type	Description	Requirement
objectType	JSON string	"application/cdmi-domain"	Mandatory
objectID	JSON string	Object ID of the domain	Mandatory
objectName	JSON string	Name of the object	Mandatory
parentURI	JSON string	URI for the parent object Appending the objectName to the parentURI shall always produce a valid URI for the object.	Mandatory
parentID	JSON string	Object ID of the parent container object	Mandatory
domainURI	JSON string	URI of the owning domain. A domain object is always owned by itself.	Mandatory
capabilitiesURI	JSON string	URI to the capabilities for the object	Mandatory
metadata	JSON object	Metadata for the domain object. This field includes any user and data system metadata specified in the request body metadata field, along with storage system metadata generated by the cloud storage system. See clause 16 for a further description of metadata.	Mandatory
childrenrange	JSON string	The sub-domains of the domain expressed as a range. If a range of sub-domains is requested, this field indicates the children returned as a range.	Mandatory
children	JSON array of JSON strings	Names of the children domains in the domain. Child containers end with "/".	Mandatory

10.5.8 Response status

Table 87 describes the HTTP status codes that occur when creating a domain object using CDMI.

Table 87: HTTP status codes - Create a domain object using CDMI

HTTP Status	Description
201 Created	The new domain object was created.
400 Bad Request	The request contains invalid parameters or field names.
401 Unauthorized	The authentication credentials are missing or invalid.
403 Forbidden	The client lacks the proper authorization to perform this request.
404 Not Found	The resource was not found at the specified URI.
409 Conflict	The operation conflicts with a non-CDMI access protocol lock or has caused a state transition error on the server.

10.5.9 Examples

EXAMPLE 1: PUT to the domain URI the domain name and metadata:

```
--> PUT /cdmi/2.0.0/cdmi_domains/MyDomain/ HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-domain
--> Content-Type: application/cdmi-domain
-->
--> "metadata":
--> {
-->   "cdmi_domain_enabled": "true"
--> }

<-- HTTP/1.1 201 Created
<-- Content-Type: application/cdmi-domain
<--
<-- {
<--   "objectType" : "application/cdmi-domain",
<--   "objectID" : "00007E7F00104BE66AB53A9572F9F51E",
<--   "objectName" : "MyDomain/",
<--   "parentURI" : "/cdmi_domains/",
<--   "parentID" : "00007E7F0010C058374D08B0AC7B3550",
<--   "domainURI" : "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI" : "/cdmi_capabilities/domain/",
<--   "metadata" : {
<--     "cdmi_domain_enabled": "true",
<--     "cdmi_authentication_methods": "anonymous, basic",
<--     ...
<--   },
<--   "childrenrange" : "0-1",
<--   "children" : [
<--     "cdmi_domain_summary/",
<--     "cdmi_domain_members/"
<--   ]
<-- }
```

10.6 Read a domain object using CDMI

10.6.1 Synopsis

To read an existing domain object, the following requests shall be performed:

- `GET <root URI>/cdmi_domains/<DomainName>/<TheDomainName>/`
- `GET <root URI>/cdmi_domains/<DomainName>/<TheDomainName>/?<fieldname>&<fieldname>&...`
- `GET <root URI>/cdmi_domains/<DomainName>/<TheDomainName>/?children=<range>&...`
- `GET <root URI>/cdmi_domains/<DomainName>/<TheDomainName>/?metadata=<prefix>&...`
- `GET <root URI>/cdmi_objectid/<DomainObjectID>/`
- `GET <root URI>/cdmi_objectid/<DomainObjectID>/?<fieldname>&<fieldname>&...`
- `GET <root URI>/cdmi_objectid/<DomainObjectID>/?children=<range>&...`
- `GET <root URI>/cdmi_objectid/<DomainObjectID>/?metadata=<prefix>&...`

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<DomainName>` is zero or more parent domains.
- `<TheDomainName>` is the name specified for the domain to be read from.
- `<fieldname>` is the name of a field.
- `<range>` is a numeric range within the list of children.
- `<prefix>` is a matching prefix that returns all metadata items that start with the prefix value.
- `<DomainObjectID>` is the ID of the domain object to be read from.

10.6.2 Capabilities

Capabilities that indicate which operations are supported are shown in Table 88.

Table 88: Capabilities - Read a CDMI domain object using CDMI

Capability	Location	Description
`cdmi_read_metadata`	Domain object	Ability to read the metadata of an existing domain object
`cdmi_list_children`	Domain object	Ability to list the children of an existing domain object
`cdmi_object_access_by_ID`	System wide capability	Ability to access the object by ID

10.6.3 Request headers

The HTTP request headers for reading a CDMI domain object using CDMI are shown in Table 89.

Table 89: Request headers - Read a domain object using CDMI

Header	Type	Description	Requirement
`Accept`	Header string	"`application/cdmi-domain`" or a consistent value as described in 5.5.2	Optional

10.6.4 Request message body

A request body shall not be provided.

10.6.5 Response headers

The HTTP response headers for reading a CDMI domain object using CDMI are shown in Table 90.

Table 90: Response headers - Read a domain object using CDMI

Header	Type	Description	Requirement
`Content-Type`	Header string	"`application/cdmi-domain`"	Mandatory
`Location`	Header string	The server shall respond with an absolute URI to which the reference redirects if the object is a reference.	Conditional

10.6.6 Response message body

The response message body fields for reading a CDMI domain object using CDMI are shown in Table 91

Table 91: Response message body - Read a domain object using CDMI

Field Name	Type	Description	Requirement
`objectType`	JSON string	"`application/cdmi-domain`"	Mandatory
`objectID`	JSON string	Object ID of the domain	Mandatory
`objectName`	JSON string	Name of the object	Mandatory
`parentURI`	JSON string	URI for the parent object Appending the "`objectName`" to the "`parentURI`" shall always produce a valid URI for the object.	Mandatory
`parentID`	JSON string	Object ID of the parent domain object • For domain objects directly under "`cdmi_domains`", the object ID of "`cdmi_domains`" container shall be returned. • For domain objects under another domain, the object ID of the parent domain shall be returned.	Mandatory
`domainURI`	JSON string	URI of the owning domain. A domain object shall always be owned by itself.	Mandatory
`capabilitiesURI`	JSON string	URI to the capabilities for the object	Mandatory
`metadata`	JSON object	Metadata for the domain object. This field includes any user and data system metadata specified in the request body metadata field, along with storage system metadata generated by the cloud storage system. See clause 16 for a further description of metadata.	Mandatory
`childrenrange`	JSON string	The sub-domains of the domain expressed as a range. If a range of sub-domains is requested, this field indicates the children returned as a range.	Mandatory
`children`	JSON array of JSON strings	The children of the domain. Sub-domains end with "`/`".	Mandatory

If individual fields are specified in the GET request, only these fields are returned in the result body. Optional fields that are requested but do not exist are omitted from the result body.

10.6.7 Response status

Table 92 describes the HTTP status codes that occur when reading a domain object using CDMI.

2322
2323 Table 92: HTTP status codes - Read a domain object using CDMI

HTTP Status	Description
200 OK	The domain object content was returned in the response.
302 Found	The resource is a reference to another resource.
400 Bad Request	The request contains invalid parameters or field names.
401 Unauthorized	The authentication credentials are missing or invalid.
403 Forbidden	The client lacks the proper authorization to perform this request.
404 Not Found	The resource was not found at the specified URI.
406 Not Acceptable	The server is unable to provide the object in the content type specified in the Accept header.

2324 ### 10.6.8 Examples

2325 EXAMPLE 1: GET to the domain URI to read all the fields of the domain:

```
--> GET /cdmi/2.0.0/cdmi_domains/MyDomain/ HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-domain

<-- HTTP/1.1 200 OK
<-- Content-Type: application/cdmi-domain
<--
<-- {
<--   "objectType": "application/cdmi-domain",
<--   "objectID": "00007E7F00104BE66AB53A9572F9F51E",
<--   "objectName": "MyDomain/",
<--   "parentURI": "/cdmi_domains/",
<--   "parentID": "00007E7F0010C058374D08B0AC7B3550",
<--   "domainURI": "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI": "/cdmi_capabilities/domain/",
<--   "metadata": {
<--     "cdmi_domain_enabled": "true",
<--     "cdmi_authentication_methods": "anonymous, basic",
<--     ...
<--   },
<--   "childrenrange": "0-1",
<--   "children": [
<--     "cdmi_domain_summary/",
<--     "cdmi_domain_members/"
<--   ]
<-- }
```

2326 EXAMPLE 2: GET to the domain URI to read the parentURI and children of the domain:

```
--> GET /cdmi/2.0.0/MyDomain/?parentURI&children HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-domain

<-- HTTP/1.1 200 OK
<-- Content-Type: application/cdmi-domain
<--
<-- {
<--   "parentURI" : "/cdmi_domains/",
<--   "children" : [
<--     "cdmi_domain_summary/",
<--     "cdmi_domain_members/"
<--   ]
<-- }
```

2327 EXAMPLE 3: GET to the domain URI to read the first two children of the domain:

```
--> GET /cdmi/2.0.0/MyDomain/?childrenrange&children=0-1 HTTP/1.1
--> Host: cloud.example.com
```

(continues on next page)

(continued from previous page)

```
--> Accept: application/cdmi-domain

<-- HTTP/1.1 200 OK
<-- Content-Type: application/cdmi-domain
<--
<-- {
<--   "childrenrange" : "0-1",
<--   "children" : [
<--     "cdmi_domain_summary/",
<--     "cdmi_domain_members/"
<--   ]
<-- }
```

10.7 Update a domain object using CDMI

10.7.1 Synopsis

To update part or all of an existing domain object, the following requests shall be performed:

- `PATCH <root URI>/cdmi_domains/<DomainName>/<TheDomainName>/`
- `PATCH <root URI>/cdmi_domains/<DomainName>/<TheDomainName>/?metadata=<metadataname>&...`
- `PATCH <root URI>/cdmi_objectid/<DomainObjectID>`
- `PATCH <root URI>/cdmi_objectid/<DomainObjectID>?metadata=<metadataname>&...`

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<DomainName>` is zero or more parent domains.
- `<TheDomainName>` is the name specified for the domain to be read from.
- `<DomainObjectID>` is the ID of the data object to be read from.

10.7.2 Delayed completion of update

Delayed completion shall not be supported for creating domain objects.

10.7.3 Capabilities

Capabilities that indicate which operations are supported are shown in Table 93.

Table 93: Capabilities - Update a CDMI domain object using CDMI

Capability	Location	Description
`cdmi_modify_metadata`	Domain object	Ability to modify the metadata of an existing domain object
`cdmi_object_access_by_ID`	System wide capability	Ability to access the object by ID

10.7.4 Request headers

The HTTP request headers for updating a CDMI domain object using CDMI are shown in Table 94.

Table 94: Request headers - Update a domain object using CDMI

Header	Type	Description	Requirement
`Content-Type`	Header string	“`application/cdmi-domain`”	Mandatory

10.7.5 Request message body

The request message body fields for updating a domain object using CDMI are shown in Table 95.

Table 95: Request message body - Update a domain object using CDMI

Field Name	Type	Description	Requirement
`metadata`	JSON object	Metadata for the domain object. If present, the new metadata specified replaces the existing object metadata. If individual metadata items are specified in the URI, only those items are replaced; other items are preserved. See clause 16 for a further description of metadata.	Optional
`copy`	JSON string	URI of a CDMI domain object that shall be copied into the existing domain object. Only the metadata and membership of the domain object itself shall be copied, not any sub-domains of the domain object. • If the destination domain object URI and the copy source domain object URI both do not specify individual fields, the destination domain object metadata and membership shall be replaced with the source domain object metadata and membership. • If the destination domain object URI or the copy source domain object URI specifies individual fields, only the fields specified shall be used to update the destination domain object. If specified fields are not present in the source, these fields shall be ignored. • If the destination domain object URI and the copy source domain object URI both specify fields, an HTTP status code of `400 Bad Request` shall be returned to the client. If there are insufficient permissions to read the domain object at the source URI or create the domain object at the destination URI, or if the read operation fails, the copy shall return an HTTP status code of `400 Bad Request`, and the destination domain object shall not be updated.	Optional[2]
`deserialize`	JSON string	URI of a CDMI data object with a value that contains a domain object serialized as specified in clause 15. The serialized domain object shall be deserialized to update the existing domain object. The object ID of the serialized domain object shall match the object ID of the destination domain object. Otherwise, the server shall return an HTTP status code of `400 Bad Request`. • If the serialized domain object does not contain sub-domains, the update is applied only to the domain object, and any existing sub-domains are left as is. • If the serialized domain object does contain sub-domains, then creates, updates, and deletes are recursively applied for each sub-domain, depending on the differences between the provided serialized state and the current state of the sub-domains.	Optional[2]

Continued on next page

Table 95 – continued from previous page

Field Name	Type	Description	Requirement
`deserializevalue`	JSON string	A domain object serialized as specified in clause 15 and encoded using base 64 encoding rules described in RFC 4648 [19], that shall be deserialized to update the existing domain object. The object ID of the serialized domain object shall match the object ID of the destination domain object. Otherwise, the server shall return an HTTP status code of `400 Bad Request`. • If the serialized domain object does not contain sub-domains, the update is applied only to the domain object, and any existing sub-domains are left as is. • If the serialized domain object does contain sub-domains, then creates, updates, and deletes are recursively applied for each sub-domain, depending on the differences between the provided serialized state and the current state of the sub-domains.	Optional[2]

10.7.6 Response header

The HTTP response header for updating a CDMI domain object using CDMI is shown in Table 96

Table 96: Response header - Update a domain object using CDMI

Header	Type	Description	Requirement
`Location`	Header string	The server shall respond with an absolute URI to which the reference redirects if the object is a reference.	Conditional

10.7.7 Response message body

A response body may be provided as per RFC 2616 [23].

10.7.8 Response status

Table 97 describes the HTTP status codes that occur when updating a domain object using CDMI.

Table 97: HTTP status codes - Update a domain object using CDMI

HTTP Status	Description
`204 No Content`	The data object content was returned in the response.
`302 Found`	The resource is a reference to another resource.
`400 Bad Request`	The request contains invalid parameters or field names.
`401 Unauthorized`	The authentication credentials are missing or invalid.
`403 Forbidden`	The client lacks the proper authorization to perform this request.
`404 Not Found`	The resource was not found at the specified URI.
`409 Conflict`	The operation conflicts with a non-CDMI access protocol lock or has caused a state transition error on the server.

10.7.9 Example

EXAMPLE 1: PATCH to the domain URI to set new field values:

[2] Only one of these fields shall be specified in any given operation. Except for value, these fields shall not be stored.

```
--> PATCH /cdmi/2.0.0/cdmi_domains/MyDomain/ HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-domain
-->
--> {
-->   "metadata" : {
-->     "test" : "value"
-->   }
--> }

<-- HTTP/1.1 204 No Content
```

10.8 Delete a domain object using CDMI

10.8.1 Synopsis

To delete an existing domain object, and transfer all objects associated with that domain to another domain (to preserve access), the following request shall be performed:

- `DELETE <root URI>/cdmi_domains/<DomainName>/<TheDomainName>/`
- `DELETE <root URI>/cdmi_objectid/<DomainObjectID>`

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<DomainName>` is zero or more parent domains.
- `<TheDomainName>` is the name specified for the domain to be deleted.
- `<DomainObjectID>` is the ID of the domain object to be deleted.

10.8.2 Capabilities

Capabilities that indicate which operations are supported are shown in Table 98.

Table 98: Capabilities - Delete a CDMI domain object using CDMI

Capability	Location	Description
`cdmi_delete_domain`	Domain object	Ability to delete an existing domain object
`cdmi_object_access_by_ID`	System wide capability	Ability to access the object by ID

10.8.3 Request headers

Request headers may be provided as per RFC 2616 [23].

10.8.4 Request message body

A request body may be provided as per RFC 2616 [23].

10.8.5 Response headers

Response headers may be provided as per RFC 2616 [23].

10.8.6 Response message body

A response body may be provided as per RFC 2616 [23].

10.8.7 Response status

Table 99 describes the HTTP status codes that occur when deleting a domain object using CDMI.

Table 99: HTTP status codes - Delete a domain object using CDMI

HTTP Status	Description
204 No Content	The domain object was successfully deleted.
400 Bad Request	The request contains invalid parameters or field names.
401 Unauthorized	The authentication credentials are missing or invalid.
403 Forbidden	The client lacks the proper authorization to perform this request.
404 Not Found	The resource was not found at the specified URI.
409 Conflict	The operation conflicts with a non-CDMI access protocol lock or has caused a state transition error on the server.

10.8.8 Example

EXAMPLE 1: DELETE to the domain object URI:

```
--> DELETE /cdmi/2.0.0/cdmi_domains/MyDomain/ HTTP/1.1
--> Host: cloud.example.com

<-- HTTP/1.1 204 No Content
```

Clause 11

Queue object resource operations using CDMI

11.1 Overview

Queue objects are similar to data objects, only with first-in, first-out access “queue”-style accesse semantics when storing and retrieving value data.

If a cloud storage system supports queues, the `cdmi_queues` system-wide capability shall be present. The ability to create a queue object is determiend by the presence or absence of the `cdmi_create_queue` and `cdmi_post_queue` capabilities in the parent container, and by the `cdmi_post_queue_by_ID` system-wide capability for creation by ID.

A queue object writer POSTs data into a queue object, and a queue object reader GETs value(s) from the queue object and subsequently deletes the value(s) to acknowledge receipt of the value(s) that it received. Queues provides a simple mechanism for one or more writers to send data to a single reader in a reliable way. If supported by the cloud storage system, cloud clients create the queue objects by using the mechanism described in 9.8 and this clause.

Each CDMI queue object is represented as a JSON object, containing one or more “fields”. For example, the “`metadata`” field contains metadata items.

EXAMPLE 1: CDMI queue object

```
{
    "objectType": "application/cdmi-queue",
    "objectID": "00007E7F00104BE66AB53A9572F9F51E",
    "objectName": "MyQueue",
    "parentURI": "/MyContainer/",
    "parentID" : "00007ED900104F67307652BAC9A37C93",
    "domainURI": "/cdmi_domains/MyDomain/",
    "capabilitiesURI": "/cdmi_capabilities/queue/",
    "completionStatus": "Complete",
    "metadata": {},
    "queueValues": "1-1",
    "mimetype": [
        "text/plain"
    ],
    "valuerange": [
        "0-19"
    ],
    "valuetransferencoding": [
        "utf-8"
    ],
    "value": [
        "First Enqueued Value"
    ]
}
```

The meaning, use, and permitted values of each field are described in each operation that creates, modifies or retreives CDMI queue objects.

11.2 Queue object details

11.2.1 Queue object addressing

Queue objects are addressed in CDMI in two ways:

- by name (e.g., `https://cloud.example.com/cdmi/2.0.0/queueobject`); and
- by ID (e.g., `https://cloud.example.com/cdmi/2.0.0/cdmi_objectid/00007ED900104F67307652BAC9A37C93/`).

Every queue object has a single, globally-unique object ID that remains constant for the life of the object. Each queue object may also have one or more URI addresses that allow the queue object to be accessed.

11.2.2 Queue object fields

Individual fields within a queue object can be accessed by specifying the field name after a question mark "`?`" appended to the end of the queue object URI.

EXAMPLE 2: The following URI returns just the number of values stored in the queue object in the response body:

```
https://cloud.example.com/cdmi/2.0.0/queueobject?queueValues
```

A list of unique fields, separated by an ampersand "`&`" can be specified, allowing multiple fields to be accessed in a single request.

EXAMPLE 3: The following URI returns the number of values stored and metadata fields in the response body:

```
https://cloud.example.com/cdmi/2.0.0/queueobject?queueValues&metadata
```

When a client provides fields that are not defined in this International Standard or deserializes an object containing fields that are not defined in this International Standard, these fields shall be persisted, but shall not be interpreted.

11.2.3 Queue object value

The encoding of the data stored in the queue object value field depends on the queue object valuetransferencoding field:

- If the value transfer encoding of the object is set to "`utf-8`", the data stored in the value of the queue object shall be a valid UTF-8 string, and it shall be transported as a UTF-8 string in the value field.
- If the value transfer encoding of the object is set to "`base64`", the data stored in the value of the queue object may contain arbitrary binary sequences, and it shall be transported as a base 64-encoded string in the value field.
- If the value transfer encoding of the object is set to "`json`", the data stored in the value of the queue object shall be a valid JSON object, and the value field shall contain a valid JSON object.

Specific ranges of the value of a queue object can be accessed by specifying a byte range after the value field name.

EXAMPLE 4: The following URI returns the first thousand bytes of the oldest value enqueued:

```
https://cloud.example.com/cdmi/2.0.0/queueobject?value=0-999
```

Because a byte range of a UTF-8 string is often not a valid UTF-8 string, the response to a range request shall always be transported in the value field as a base 64-encoded string.

Byte ranges are specified as single, inclusive byte ranges as per Section 14.35.1 of RFC 2616 [23].

If read access to any of the requested fields is not permitted by the object ACL, only the permitted fields shall be returned. If no requested fields are permitted to be read, an HTTP status code of `403 Forbidden` shall be returned to the client.

If write access to any of the requested fields is not permitted by the object ACL, no updates shall be performed, and an HTTP status code of `403 Forbidden` shall be returned to the client.

When a client provides or includes deserialization fields that are not defined in this International Standard, these fields shall be stored as part of the object.

The value of a queue object may also be specified and retrieved using multi-part MIME, where the CDMI JSON is transferred in the first MIME part and the raw queue values are transferred in the subsequent MIME parts. Each MIME

part, including any header fields, shall conform to RFC 2045 [9], RFC 2046 [10], and RFC 2616 [23], and the length of each part may optionally be specified by a ‘’Content-Length’’ header in addition to the MIME boundary delimiter.

Multiple non-overlapping ranges of the value of a queue object may also be accessed or updated in a multi-part MIME operation by transferring one MIME part for each range of the value. The byte ranges for these operations shall be specified as per Section 14.35.1 of RFC 2616 [23].

Multi-part MIME enables the efficient transfer of binary data alongside CDMI object metadata without incurring the overhead of the UTF-8 or Base64 encoding and validation required to represent binary data in JSON.

11.2.4 Queue object metadata

Queue object metadata may also include arbitrary user-supplied metadata, storage system metadata, and data system metadata, as specified in clause 16. Metadata shall be stored as a valid UTF-8 string. Binary data stored in user metadata shall be first encoded such that it can be contained in a UTF-8 string, with the use of base 64 encoding recommended.

Every queue object has a parent object from which the queue object inherits data system metadata that is not explicitly specified in the data object itself.

EXAMPLE 5: The “pages” queue object stored at the following URI would inherit data system metadata from its parent container, “OCR”:

```
https://cloud.example.com/cdmi/2.0.0/OCR/pages
```

11.2.5 Queue object access control

If read access to any of the requested fields is not permitted by the object ACL, only the permitted fields shall be returned. If no requested fields are permitted to be read, an HTTP status code of `403 Forbidden` shall be returned to the client.

If write access to any of the requested fields is not permitted by the object ACL, no updates shall be performed, and an HTTP status code of `403 Forbidden` shall be returned to the client.

11.2.6 Queue object consistency

Writing to a queue object is an atomic operation.

For non-value-related fields:

- If a client reads a queue object simultaneously with a write to that same queue object, the reading client shall get either the old version or the new version, but not a mixture of both.
- If a write is terminated due to errors, the contents of the queue object shall be as if the write never occurred (i.e., writes are atomic in the face of errors).

For value-related fields:

- If a client dequeues or deletes one or more queue values simultaneously with one or more queue values being enqueued to that same queue object, the order of operations shall be as if the dequeue/delete operation happens before the enqueue operation.
- If a dequeue, delete or enqueue is terminated due to errors, the contents of the queue object shall be as if the dequeue/delete/enqueue never occurred (i.e., writes are atomic in the face of errors).

Create and update timestamps that are returned in response to multiple client writes to a given object may indicate that a specific write is the newest (i.e., the write whose data is expected to be returned to subsequent reads until another write is processed). However, there is no guarantee that the write with the latest timestamp is the one whose data is returned on subsequent reads.

Implementations of this International Standard shall provide the atomicity features described in this subclause for queue objects that are accessed via CDMI. The atomicity properties of queue objects that are accessed by protocols other than CDMI are outside the scope of this International Standard.

11.2.7 Queue object representations

The representations in this clause are shown using JSON notation. Both clients and servers shall support UTF-8 JSON representation. The request and response body JSON fields may be specified or returned in any order, with the exception that, if present, for queue objects, the "`valuerange`" and "`value`" fields shall appear last and in that order.

11.3 Create a queue object using CDMI

11.3.1 Synopsis

To create a new queue object, the following request shall be performed:

- `PUT <root URI>/<ContainerName>/<QueueName>`

To create a new queue object by ID, see 9.8.

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<ContainerName>` is zero or more intermediate containers that already exist, with one slash (i.e., "`/`") between each pair of container names.
- `<QueueName>` is the name specified for the queue object to be created.

After it is created, the object shall also be accessible at `<root URI>/cdmi_objectid/<objectID>`.

The newly created queue shall have no values unless the queue is created as a result of copying or moving a source queue that has values or as a result of deserializing a serialized queue that has values.

11.3.2 Delayed completion of create

In response to a create operation for a queue object, the server may return an HTTP status code of `202 Accepted` to indicate that the object is in the process of being created. This response is useful for long-running operations (e.g., copying a large queue object from a source URI). Such a response has the following implications.

- The server shall return a ''Location'' header with an absolute URI to the object to be created along with an HTTP status code of `202 Accepted`.
- With an HTTP status code of `202 Accepted`, the server implies that the following checks have passed:
  - user authorization for creating the object;
  - user authorization for read access to any source object for move, copy, serialize, or deserialize; and
  - availability of space to create the object or at least enough space to create a URI to report an error.
- A client might not be able to immediately access the created object, e.g., due to delays resulting from the implementation's use of eventual consistency.

The client performs GET operations to the URI to track the progress of the operation. In response, the server returns two fields in its response body to indicate progress.

- A mandatory `completionStatus` text field contains either "`Processing`", "`Complete`", or an error string starting with the value "`Error`".
- An optional `percentComplete` field contains the percentage of the operation that has completed (0 to 100).

GET shall not return any value for the queue object when `completionStatus` is not "`Complete`". If the final result of the create operation is an error, the URI is created with the `completionStatus` field set to the error message. It is the client's responsibility to delete the URI after the error has been noted.

11.3.3 Capabilities

Capabilities that indicate which operations are supported are shown in Table 100.

Table 100: Capabilities - Create a CDMI queue object using CDMI

Capability	Location	Description
`cdmi_create_queue`	Parent container	Ability to create a new queue object
`cdmi_create_reference`	Parent container	Ability to create a new reference
`cdmi_copy_queue`	Parent container	Ability to create a queue object that is a copy of another queue object

Continued on next page

Table 100 – continued from previous page

Capability	Location	Description
cdmi_move_queue	Parent container	Ability to move a queue object from another queue object
cdmi_deserialize_queue	Parent container	Ability to create a queue object that is deserialized from the contents of the PUT or the contents of another data object

11.3.4 Request headers

The HTTP request headers for creating a CDMI queue object using CDMI are shown in Table 101

Table 101: Request headers - Create a queue object Using CDMI

Header	Type	Description	Requirement
Accept	Header string	"application/cdmi-queue"	Mandatory
Content-Type	Header string	"application/cdmi-queue"	Mandatory

11.3.5 Request message body

The request message body fields for creating a queue object using CDMI are shown in Table 102.

Table 102: Request message body - Create a queue object using CDMI

Field Name	Type	Description	Requirement
metadata	JSON object	Metadata for the queue object • If this field is included, the contents of the JSON object provided in this field shall be used as queue object metadata. • If this field is included when deserializing, serializing, copying, or moving a queue object, the contents of the JSON object provided in this field shall be used as object metadata instead of the metadata from the source URI. • If this field is not included, no user-specified metadata shall be added to the object. • If this field is not included when deserializing, serializing, copying, or moving a queue object, metadata from the source URI shall be used. • This field shall not be included when creating a reference to a queue object.	Optional
domainURI	JSON string	URI of the owning domain • If different from the parent domain, the user shall have the "cross_domain" privilege (see cdmi_member_privileges in Table 80). • If not specified, the domain of the parent container shall be used.	Optional
deserialize	JSON string	URI of a CDMI data object with a value that contains a queue object serialized as specified in clause 15. The serialized queue object shall be deserialized to create the new queue object.	Optional[1]

Continued on next page

Table 102 – continued from previous page

Field Name	Type	Description	Requirement
`copy`	JSON string	URI of a source CDMI queue object that shall be copied into the new destination queue object. • If the destination queue object URI and the copy source queue object URI both do not specify individual fields, the destination queue object shall be a complete copy of the source queue object, including all enqueued values. • If the destination queue object URI or the copy source queue object URI specifies individual fields, only the fields specified shall be used to create the destination queue object. If specified fields are not present in the source, default field values shall be used. • If the destination queue object URI and the copy source queue object URI both specify fields, an HTTP status code of `400 Bad Request` shall be returned to the client. If there are insufficient permissions to read the queue object at the source URI or create the queue object at the destination URI, or if the read operation fails, the copy shall return an HTTP status code of `400 Bad Request`, and the destination queue object shall not be created.	Optional[1]
`move`	JSON string	URI of an existing local or remote CDMI queue object (source URI) that shall be relocated to the URI specified in the PUT. The contents of the queue object, including the object ID, shall be preserved by a move, and the queue object at the source URI shall be removed after the queue object at the destination has been successfully created. If there are insufficient permissions to read the queue object at the source URI, write the queue object at the destination URI, or delete the queue object at the source URI, or if any of these operations fail, the move shall return an HTTP status code of `400 Bad Request`, and the source and destination are left unchanged.	Optional[1]
`reference`	JSON string	URI of a CDMI queue object that shall be redirected to by a reference. If other fields are supplied when creating a reference, the server shall respond with an HTTP status code of `400 Bad Request`.	Optional[1]
deserializevalue	JSON string	A queue object serialized as specified in clause 15 and encoded using base 64 encoding rules described in RFC 4648 [19], that shall be deserialized to create the new queue object.	Optional[1]

11.3.6 Response status

The HTTP response headers for creating a CDMI queue object using CDMI are shown in Table 103

[1] Only one of these fields shall be specified in any given operation. Except for value, these fields shall not be stored. If more than one of these fields is supplied, the server shall respond with an HTTP status code of `400 Bad Request`.

2541
2542 Table 103: Response headers - Create a queue object Using CDMI

Header	Type	Description	Requirement
`Content-Type`	Header string	"`application/cdmi-queue`"	Mandatory
`Location`	Header string	When an HTTP status code of `202 Accepted` is returned, the server shall respond with the absolute URL of the object that is in the process of being created.	Conditional

2543 11.3.7 Response message body

2544 The response message body fields for creating a CDMI queue object using CDMI are shown in Table 104

Table 104: Response message body - Create a queue object using CDMI

Field Name	Type	Description	Requirement
`objectType`	JSON string	"`application/cdmi-queue`"	Mandatory
`objectID`	JSON string	Object ID of the object	Mandatory
`objectName`	JSON string	Name of the object	Mandatory
`parentURI`	JSON string	URI for the parent object Appending the objectName to the parentURI shall always produce a valid URI for the object.	Mandatory
`parentID`	JSON string	Object ID of the parent container object	Mandatory
`domainURI`	JSON string	URI of the owning domain.	Mandatory
`capabilitiesURI`	JSON string	URI to the capabilities for the object	Mandatory
`completionStatus`	JSON string	A string indicating if the object is still in the process of being created or updated by another operation, and after that operation is complete, indicates if it was successfully created or updated or if an error occurred. The value shall be the string "`Processing`", the string "`Complete`", or an error string starting with the value "`Error`".	Mandatory
`percentComplete`	JSON string	A string indicating the percentage of completion if the object is still in the process of bewing created or updated by another operation. • When the value of `completionStatus` is "`Processing`", this field, if provided, shall indicate the percentage of completion as a numeric integer value from "`0`" through "`100`". • When the value of `completionStatus` is "`Complete`", this field, if provided, shall contain the value "`100`". • When the value of `completionStatus` is "`Error`", this field, if provided, may contain any integer value from "`0`" through "`100`".	Optional

Continued on next page

Table 104 – continued from previous page

Field Name	Type	Description	Requirement
metadata	JSON object	Metadata for the queue object. This field includes any user and data system metadata specified in the request body metadata field, along with storage system metadata generated by the cloud storage system. See clause 16 for a further description of metadata.	Mandatory
queueValues	JSON string	The range of designators for enqueued values. Every enqueued value shall be assigned a unique, monotonically-incrementing positive integer designator, starting from 0. If no values are enqueued, an empty string shall be returned. If values are enqueued, the lowest designator, followed by a hyphen ("-"), followed by the highest designator shall be returned.	Mandatory

11.3.8 Response status

The HTTP status codes that occur when creating a queue object using CDMI are described in Table 105.

Table 105: HTTP status codes - Create a queue object using CDMI

HTTP Status	Description
201 Created	The new queue object was created.
202 Accepted	The queue object is in the process of being created. The CDMI client should monitor the completionStatus and percentComplete fields to determine the current status of the operation.
400 Bad Request	The request contains invalid parameters or field names.
401 Unauthorized	The authentication credentials are missing or invalid.
403 Forbidden	The client lacks the proper authorization to perform this request.
404 Not Found	The resource was not found at the specified URI.
409 Conflict	The operation conflicts with a non-CDMI access protocol lock or has caused a state transition error on the server.

11.3.9 Examples

Example 1: PUT to the queue URI the queue object name and contents:

```
--> PUT /cdmi/2.0.0/MyContainer/MyQueue HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-queue
--> Content-Type: application/cdmi-queue
-->
--> {
-->   "metadata" : {
-->
-->   }
--> }

<-- HTTP/1.1 201 Created
<-- Content-Type: application/cdmi-queue
<--
<-- {
<--   "objectType" : "application/cdmi-queue",
<--   "objectID" : "00007E7F00104BE66AB53A9572F9F51E",
<--   "objectName" : "MyQueue",
<--   "parentURI " : "/MyContainer/",
<--   "parentID" : "00007ED900104F67307652BAC9A37C93",
<--   "domainURI" : "/cdmi_domains/MyDomain/",
```

(continues on next page)

(continued from previous page)

```
<--   "capabilitiesURI" : "/cdmi_capabilities/queue/",
<--   "completionStatus" : "Complete",
<--   "metadata" : {
<--     ...
<--   },
<--   "queueValues" : ""
<-- }
```

EXAMPLE 2: PUT to the queue object URI to create a new queue, copying from another queue:

```
--> PUT /cdmi/2.0.0/MyContainer/MyQueue HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-queue
-->
--> {
-->   "copy": "/MyContainer/SourceQueue?value=0-9"
--> }

<-- HTTP/1.1 201 Created
<-- Content-Type: application/cdmi-queue
<--
<-- {
<--   "objectType": "application/cdmi-queue",
<--   "objectID": "00007E7F00104BE66AB53A9572F9F51E",
<--   "objectName": "MyQueue",
<--   "parentURI ": "/MyContainer/",
<--   "parentID": "00007ED900104F67307652BAC9A37C93",
<--   "domainURI": "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI": "/cdmi_capabilities/queue/",
<--   "completionStatus": "Complete",
<--   "metadata": {
<--     ...
<--   },
<--   "queueValues": "0-9"
<-- }
```

11.4 Read a queue object using CDMI

11.4.1 Synopsis

To read all fields from an existing queue object, the following request shall be performed:

- GET <root URI>/<ContainerName>/<QueueName>
- GET <root URI>/<ContainerName>/<QueueName>?<fieldname>&<fieldname>&...
- GET <root URI>/<ContainerName>/<QueueName>?value=<range>&...
- GET <root URI>/<ContainerName>/<QueueName>?metadata=<prefix>&...
- GET <root URI>/<ContainerName>/<QueueName>?values=<count>
- GET <root URI>/cdmi_objectid/<QueueObjectID>
- GET <root URI>/cdmi_objectid/<QueueObjectID>?<fieldname>&<fieldname>&...
- GET <root URI>/cdmi_objectid/<QueueObjectID>?value=<range>&...
- GET <root URI>/cdmi_objectid/<QueueObjectID>?metadata=<prefix>&...
- GET <root URI>/cdmi_objectid/<QueueObjectID>?values=<count>

Where:

- <root URI> is the path to the CDMI cloud.
- <ContainerName> is zero or more intermediate containers.
- <QueueName> is the name of the queue object to be read from.
- <fieldname> is the name of a field.
- <range> is a byte range of the queue object value to be returned in the value field. If a byte range is requested, the range returned shall be from the oldest queue object value.
- <prefix> is a matching prefix that returns all metadata items that start with the prefix value.
- <count> is the number of values to be retrieved from the queue object. If more queue object entries are requested to be retrieved than exist in the queue object, the count is processed as if it is equal to the number of entries in the queue object.
- <QueueObjectID> is the ID of the queue object to be read from.

Reading a queue object shall, by default, return the complete value of the oldest item in the queue, unless the queue-Values range is empty.

11.4.2 Capabilities

Capabilities that indicate which operations are supported are shown in Table 106.

Table 106: Capabilities - Read a CDMI queue object using CDMI

Capability	Location	Description
cdmi_read_metadata	Queue object	Ability to read the metadata of an existing queue object
cdmi_read_value	Queue object	Ability to read the value of an existing queue object
cdmi_multipart_mime	Queue object	Ability to read a queue object using multi-part MIME
cdmi_object_access_by_ID	System wide capability	Ability to access the object by ID

11.4.3 Request headers

The HTTP request headers for reading a CDMI queue object using CDMI are shown in Table 107

Table 107: Request headers - Read a queue object using CDMI

Header	Type	Description	Requirement
Accept	Header string	"application/cdmi-queue", "multipart/mixed", or a consistent value as defined in 5.5.2 If "multipart/mixed", the body shall consist of one or more MIME parts, where the first part shall contain a body of content-type "application/cdmi-queue", and the second and subsequent parts shall each contain the corresponding queue value.	Optional

11.4.4 Request message body

A request body shall not be provided.

11.4.5 Response status

The HTTP response headers for reading a CDMI queue object using CDMI are shown in Table 108.

Table 108: Response headers - Read a queue object using CDMI

Header	Type	Description	Requirement
Content-Type	Header string	"application/cdmi-queue" or "multipart/mixed"	Mandatory
Location	Header string	The server shall respond with an absolute URI to which the reference redirects if the object is a reference.	Conditional

11.4.6 Response message body

The response message body fields for reading a CDMI queue object using CDMI are shown in Table 109

Table 109: Response message body - Read a queue object using CDMI

Field Name	Type	Description	Requirement
objectType	JSON string	"application/cdmi-queue"	Mandatory
objectID	JSON string	Object ID of the object	Mandatory
objectName	JSON string	Name of the object • For objects in a container, the objectName field shall be returned. • For objects not in a container (objects that are only accessible by ID), the "objectName" field does not exist and shall not be returned.	Conditional
parentURI	JSON string	URI for the parent object • For objects in a container, the parentURI field shall be returned. • For objects not in a container (objects that are only accessible by ID), the "parentURI" field does not exist and shall not be returned. Appending the "objectName" to the "parentURI" shall always produce a valid URI for the object.	Conditional

Continued on next page

Table 109 – continued from previous page

Field Name	Type	Description	Requirement
parentID	JSON string	Object ID of the parent container object • For objects in a container, the "parentID" field shall be returned. • For objects not in a container (objects that are only accessible by ID), the "parentID" field does not exist and shall not be returned.	Conditional
domainURI	JSON string	URI of the owning domain	Mandatory
capabilitiesURI	JSON string	URI to the capabilities for the object	Mandatory
completionStatus	JSON string	A string indicating if the object is still in the process of being created or updated by another operation, and after that operation is complete, indicates if it was successfully created or updated or if an error occurred. The value shall be the string "Processing", the string "Complete", or an error string starting with the value "Error".	Mandatory
percentComplete	JSON string	A string indicating the percentage of completion if the object is still in the process of bewing created or updated by another operation. • When the value of completionStatus is "Processing", this field, if provided, shall indicate the percentage of completion as a numeric integer value from 0 through 100. • When the value of completionStatus is "Complete", this field, if provided, shall contain the value "100". • When the value of completionStatus is "Error", this field, if provided, may contain any integer value from "0" through "100".	Optional
metadata	JSON object	Metadata for the queue object. This field includes any user and data system metadata specified in the request body metadata field, along with storage system metadata generated by the cloud storage system. See clause 16 for a further description of metadata.	Mandatory
queueValues	JSON string	The range of designators for enqueued values. Every enqueued value shall be assigned a unique, monotonically-incrementing positive integer designator, starting from 0. If no values are enqueued, an empty string shall be returned. If values are enqueued, the lowest designator, followed by a hyphen ("-"), followed by the highest designator shall be returned. • This field shall only be provided when completionStatus is "Complete" and when one or more values are enqueued.	Mandatory
mimetype	JSON array of JSON strings	MIME types for each queue object value * The MIME types of the values are returned, each corresponding to the value in the same position in the JSON array. * This field shall only be provided when completionStatus is "Complete" and when one or more values are enqueued.	Optional

Continued on next page

Table 109 – continued from previous page

Field Name	Type	Description	Requirement
valuerange	JSON array of JSON strings	The range of bytes of the queue object values to be returned in the value field • The value ranges of the values are returned, each corresponding to the value in the same position in the JSON array. • If a specific value range has been requested, the entry in the valuerange field shall correspond to the bytes requested. If the request extends beyond the end of the value, the valuerange field shall indicate the smaller byte range returned. • This field shall only be provided when completionStatus is "Complete" and when one or more values are enqueued.	Optional
valuetransferencoding	JSON array of JSON strings	The value transfer encoding used for each queue object value. Two value transfer encodings are defined: • "utf-8" indicates that the queue object value contains a valid UTF-8 string, and it shall be transported as a UTF-8 string in the value field. • "base64" indicates that the queue object value may contain arbitrary binary sequences, and it shall be transported as a base 64-encoded string in the value field. • "json" indicates that the queue object value contains a valid JSON object, and the value field shall contain a JSON object. The value transfer encodings are returned, each corresponding to the value in the same position in the JSON array. • This field shall only be provided when completionStatus is "Complete" and when one or more values are enqueued.	Optional
value	JSON array of JSON strings	The oldest enqueued queue object values • The values in the JSON array are returned in order from oldest to newest. • If the valuetransferencoding field indicates UTF-8 encoding, the corresponding value field shall contain a UTF-8 string using JSON escaping rules described in RFC 4627 [5]. • If the valuetransferencoding field indicates base 64 encoding, the corresponding value field shall contain a base 64-encoded string as described in RFC RFC 4648 [19]. • If the valuetransferencoding field indicates JSON encoding, the corresponding value field shall contain a JSON object. • The value field shall not be provided when using multi-part MIME. • The value field shall only be provided when the completionStatus field contains "Complete".	Conditional

2593 If individual fields are specified in the GET request, only these fields are returned in the result body. Optional fields that
2594 are requested but do not exist are omitted from the result body.

11.4.7 Response status

The HTTP status codes that occur when reading a queue object using CDMI are described in Table 110.

Table 110: HTTP status codes - Read a queue object using CDMI

HTTP Status	Description
200 OK	The queue object content was returned in the response.
302 Found	The resource is a reference to another resource.
400 Bad Request	The request contains invalid parameters or field names.
401 Unauthorized	The authentication credentials are missing or invalid.
403 Forbidden	The client lacks the proper authorization to perform this request.
404 Not Found	The resource was not found at the specified URI.
406 Not Acceptable	The server is unable to provide the object in the content type specified in the Accept header.

11.4.8 Examples

EXAMPLE 1: GET to the queue object URI to read all fields of the queue object:

```
--> GET /cdmi/2.0.0/MyContainer/MyQueue HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-queue

<-- HTTP/1.1 200 OK
<-- Content-Type: application/cdmi-queue
<--
<-- {
<--   "objectType": "application/cdmi-queue",
<--   "objectID": "00007E7F00104BE66AB53A9572F9F51E",
<--   "objectName": "MyQueue",
<--   "parentURI": "/MyContainer/",
<--   "parentID" : "00007ED900104F67307652BAC9A37C93",
<--   "domainURI": "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI": "/cdmi_capabilities/queue/",
<--   "completionStatus": "Complete",
<--   "metadata": {},
<--   "queueValues": "1-1",
<--   "mimetype": [
<--     "text/plain"
<--   ],
<--   "valuerange": [
<--     "0-19"
<--   ],
<--   "valuetransferencoding": [
<--     "utf-8"
<--   ],
<--   "value": [
<--     "First Enqueued Value"
<--   ]
<-- }
```

EXAMPLE 2: GET to the queue object URI to read the value and queue items of the queue object:

```
--> GET /cdmi/2.0.0/MyContainer/MyQueue?value&queueValues HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-queue

<-- HTTP/1.1 200 OK
<-- Content-Type: application/cdmi-queue
<--
<-- {
<--   "queueValues" : "1-1",
<--   "value" : [
```

(continues on next page)

(continued from previous page)

```
<--   "First Enqueued Value"
<--   ]
<-- }
```

2602 EXAMPLE 3: GET to the queue object URI to read the first five bytes of the value of the queue object:

```
--> GET /cdmi/2.0.0/MyContainer/MyQueue?value:0-4 HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-queue

<-- HTTP/1.1 200 OK
<-- Content-Type: application/cdmi-queue
<--
<-- {
<--   "value" : [
<--     "First"
<--   ]
<-- }
```

2603 EXAMPLE 4: GET to the queue object URI to read two values of the queue object:

```
--> GET /cdmi/2.0.0/MyContainer/MyQueue?mimetype&valuerange&values=2 HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-queue

<-- HTTP/1.1 200 OK
<-- Content-Type: application/cdmi-queue
<--
<-- {
<--   "mimetype" : [
<--     "text/plain",
<--     "text/plain"
<--   ],
<--   "valuerange" : [
<--     "0-19",
<--     "0-20"
<--   ],
<--   "value" : [
<--     "First Enqueued Value",
<--     "Second Enqueued Value"
<--   ]
<-- }
```

2604 EXAMPLE 5: GET to the queue object URI to read the queue object using multi-part MIME:

```
--> GET /cdmi/2.0.0/MyContainer/MyQueue HTTP/1.1
--> Host: cloud.example.com
--> Accept: multipart/mixed

<-- HTTP/1.1 200 OK
<-- Content-Type: multipart/mixed; boundary=gc0p4Jq0M2Yt08j34c0p
<--
<-- --gc0p4Jq0M2Yt08j34c0p
<-- Content-Type: application/cdmi-queue
<--
<-- {
<--   "objectType": "application/cdmi-queue",
<--   "objectID": "00007ED9001035E14BD1BA70C2EE98FC",
<--   "objectName": "MyQueue",
<--   "parentURI": "/MyContainer/",
<--   "parentID" : " 00007ED90010C2414303B5C6D4F83170",
<--   "domainURI": "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI": "/cdmi_capabilities/queue/",
<--   "completionStatus": "Complete",
<--   "metadata": {
<--     ...
<--   },
<--   "queueValues": "1-2",
```

(continues on next page)

(continued from previous page)

```
<--   "mimetype": [
<--     "application/octet-stream",
<--     "application/octet-stream"
<--   ],
<--   "valuerange": [
<--     "0-19",
<--     "0-36"
<--   ],
<--   "valuetransferencoding": [
<--     "base64",
<--   "base64"
<--   ]
<-- }
<--
<-- --gc0p4Jq0M2Yt08j34c0p
<-- Content-Type: application/octet-stream
<-- Content-Transfer-Encoding: binary
<--
<-- <20 bytes of binary data>
<--
<-- --gc0p4Jq0M2Yt08j34c0p
<-- Content-Type: application/octet-stream
<-- Content-Transfer-Encoding: binary
<--
<-- <37 bytes of binary data>
<--
<-- --gc0p4Jq0M2Yt08j34c0p--
```

11.5 Update a queue object using CDMI

11.5.1 Synopsis

To update some or all fields in an existing queue object (excluding the enqueueing of values), the following request shall be performed:

- `PATCH <root URI>/<ContainerName>/<QueueName>`
- `PATCH <root URI>/<ContainerName>/<QueueName>?metadata=<metadataname>&...`
- `PATCH <root URI>/cdmi_objectid/<QueueObjectID>`
- `PATCH <root URI>/cdmi_objectid/<QueueObjectID>?metadata=<metadataname>&...`

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<ContainerName>` is zero or more intermediate containers.
- `<QueueName>` is the name of the queue object to be updated.
- `<QueueObjectID>` is the ID of the queue object to be updated.

11.5.2 Capabilities

Capabilities that indicate which operations are supported are shown in Table 111.

Table 111: Capabilities - Update a queue object using CDMI

Capability	Location	Description
`cdmi_modify_metadata`	Queue object	Ability to modify the metadata of an existing queue object
`cdmi_object_access_by_ID`	System wide capability	Ability to access the object by ID

11.5.3 Request headers

The HTTP request headers for updating a CDMI queue object using CDMI are shown in Table 112

Table 112: Request headers - Update a queue object Using CDMI

Header	Type	Description	Requirement
`Content-Type`	Header string	“application/cdmi-queue”	Mandatory

11.5.4 Request message body

The request message body fields for updating a queue object using CDMI are shown in Table 113.

Table 113: Request message body - Update a queue object Using CDMI

Field Name	Type	Description	Requirement
`metadata`	JSON object	Metadata for the queue object. If present, the new metadata specified replaces the existing object metadata. If individual metadata items are specified in the URI, only those items are replaced; other items are preserved. See clause 16 for a further description of metadata.	Optional

Continued on next page

Table 113 – continued from previous page

Field Name	Type	Description	Requirement
`domainURI`	JSON string	URI of the owning domain • If different from the parent domain, the user shall have the "`cross-domain`" privilege (see `cdmi_member_privileges` in Table 80). • If not specified, the existing domain shall be preserved.	Optional
`deserialize`	JSON string	URI of a CDMI data object with a value that contains a queue object serialized as specified in clause 15. The serialized queue object shall be deserialized to update the existing queue object. • If the destination queue object URI and the source serialized queue object URI both do not specify individual fields, the destination queue object shall be replaced with the contents of the serialized source queue object, with the exception that the destination queue values shall be preserved. See 11.7 to deserailize enqueued items. • If the destination queue object URI or the source serialized queue object URI specifies individual fields, only the fields specified shall be used to update the destination queue object. If specified fields are not present in the source, these fields shall be ignored. If the value field is specified, it shall be ignored. • If the destination queue object URI and the source serialized queue object URI both specify fields, an HTTP status code of `400 Bad Request` shall be returned to the client. If there are insufficient permissions to read the serialized queue object at the source URI or update the queue object at the destination URI, or if the read operation fails, the update shall return an HTTP status code of `400 Bad Request`, and the destination queue object shall not be updated.	Optional[2]

Continued on next page

Table 113 – continued from previous page

Field Name	Type	Description	Requirement
`copy`	JSON string	URI of a source CDMI queue object that shall be copied into the existing destination queue object. • If the destination queue object URI and the copy source queue object URI both do not specify individual fields, the destination queue object shall be replaced with the source queue object, with the exception that the destination queue values shall be preserved. See 11.7 to copy enqueued items. • If the destination queue object URI or the copy source queue object URI specifies individual fields, only the fields specified shall be used to update the destination queue object. If specified fields are not present in the source, these fields shall be ignored. If the value field is specified, it shall be ignored. • If the destination queue object URI and the copy source queue object URI both specify fields, an HTTP status code of `400 Bad Request` shall be returned to the client. If there are insufficient permissions to read the queue object at the source URI or update the queue object at the destination URI, or if the read operation fails, the update shall return an HTTP status code of `400 Bad Request`, and the destination queue object shall not be updated.	Optional[2]
`deserializevalue`	JSON string	A queue object serialized as specified in clause 15 and encoded using base 64 encoding rules described in RFC 4648 [19], that shall be deserialized to update the existing queue object. The object ID of the serialized queue object shall match the object ID of the destination queue object. Otherwise, the server shall return an HTTP status code of `400 Bad Request`. • If the destination queue object URI does not specify individual fields, the destination queue object shall be replaced with the contents of the serialized source queue object, with the exception that the destination queue values shall be preserved. See 11.7 to deserailize enqueued items. • If the destination queue object URI specifies individual fields, only the fields specified shall be used to update the destination queue object. If specified fields are not present in the source, these fields shall be ignored. If the value field is specified, it shall be ignored. If there are insufficient permissions update the queue object at the destination URI, the update shall return an HTTP status code of `400 Bad Request`, and the destination queue object shall not be updated.	Optional[2]

11.5.5 Response header

The HTTP response header for updating a CDMI queue object using CDMI is shown in Table 114

[2] Only one of these fields shall be specified in any given operation. Except for value, these fields shall not be stored.

Table 114: Response header - Update a queue object Using CDMI

Header	Type	Description	Requirement
Location	Header string	The server shall respond with an absolute URI to which the reference redirects if the object is a reference.	Conditional

11.5.6 Response message body

A response body can be provided as per RFC 2616 [23].

11.5.7 Response status

Table 115 describes the HTTP status codes that occur when updating a queue object using CDMI.

Table 115: HTTP status codes - Update a queue object using CDMI

HTTP Status	Description
204 No Content	The data object content was returned in the response.
302 Found	The resource is a reference to another resource.
400 Bad Request	The request contains invalid parameters or field names.
401 Unauthorized	The authentication credentials are missing or invalid.
403 Forbidden	The client lacks the proper authorization to perform this request.
404 Not Found	The resource was not found at the specified URI.
409 Conflict	The operation conflicts with a non-CDMI access protocol lock or has caused a state transition error on the server.

11.5.8 Examples

EXAMPLE 1: PATCH to the queue object URI to set new metadata:

```
--> PATCH /cdmi/2.0.0/MyContainer/MyQueue HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-queue
-->
--> {
-->   "metadata" : {
-->
-->   }
--> }

<-- HTTP/1.1 204 No Content
```

EXAMPLE 2: PATCH to the queue object URI to move six queue values from another queue:

```
--> PATCH /cdmi/2.0.0/MyContainer/MyQueue HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-queue
-->
--> {
-->   "move": "/MyContainer/SourceQueue?value:10-15"
--> }

<-- HTTP/1.1 204 No Content
```

11.6 Delete a queue object using CDMI

11.6.1 Synopsis

To delete an existing queue object, along with all enqueued values, the following request shall be performed:

- `DELETE <root URI>/<ContainerName>/<QueueName>`
- `DELETE <root URI>/cdmi_objectid/<QueueObjectID>`

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<ContainerName>` is zero or more intermediate containers.
- `<QueueName>` is the name of the queue object to be deleted.
- `<QueueObjectID>` is the ID of the queue object to be deleted.

11.6.2 Capability

Capabilities that indicate which operations are supported are shown in Table 116.

Table 116: Capabilities - Delete a queue object using CDMI

Capability	Location	Description
`cdmi_delete_queue`	Queue object	Ability to delete an existing queue object
`cdmi_object_access_by_ID`	System wide capability	Ability to access the object by ID

11.6.3 Request header

Request headers can be provided as per RFC 2616 [23].

11.6.4 Request message body

A request body can be provided as per RFC 2616 [23].

11.6.5 Response headers

Response headers can be provided as per RFC 2616 [23].

11.6.6 Response message body

A response body can be provided as per RFC 2616 [23].

11.6.7 Response status

Table 117 describes the HTTP status codes that occur when deleting a queue object using CDMI.

Table 117: HTTP status codes - Delete a queue object Using CDMI

HTTP Status	Description
204 No Content	The queue object was successfully deleted.
400 Bad Request	The request contains invalid parameters or field names.
401 Unauthorized	The authentication credentials are missing or invalid.
403 Forbidden	The client lacks the proper authorization to perform this request.
404 Not Found	The resource was not found at the specified URI.
409 Conflict	The operation conflicts with a non-CDMI access protocol lock or has caused a state transition error on the server.

11.6.8 Example

EXAMPLE 1: DELETE to the queue object URI:

```
--> DELETE /cdmi/2.0.0/MyContainer/MyQueue HTTP/1.1
--> Host: cloud.example.com

<-- HTTP/1.1 204 No Content
```

11.7 Enqueue a new queue object value using CDMI

11.7.1 Synopsis

To enqueue one or more values into an existing queue object, the following request shall be performed:

- `POST <root URI>/<ContainerName>/<QueueName>`
- `POST <root URI>/cdmi_objectid/<QueueObjectID>`

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<ContainerName>` is zero or more intermediate containers that already exist, with one slash (i.e., "`/`") between each pair of container names.
- `<QueueName>` is the name of the queue object to be enqueued into.
- `<QueueObjectID>` is the ID of the queue object to be enqueued into.

11.7.2 Capabilities

Capabilities that indicate which operations are supported are shown in Table 118.

Table 118: Capabilities - Enqueue a new queue object value using CDMI

Capability	Location	Description
`cdmi_modify_value`	Queue object	Ability to enqueue a value into an existing queue object
`cdmi_multipart_mime`	System wide capability	Ability to modify a queue object using multi-part MIME
`cdmi_object_access_by_ID`	System wide capability	Ability to access the object by ID

11.7.3 Request headers

The HTTP request headers for enqueuing a new CDMI queue object value using CDMI are shown in Table 119

Table 119: Request headers - Enqueue a new queue object value using CDMI

Header	Type	Description	Requirement
`Content-Type`	Header string	"`application/cdmi-queue`" or "`multipart/mixed`" If "`multipart/mixed`", the first part shall contain a body of content-type "`application/cdmi-queue`", and the subsequent parts shall contain the queue values as described in 8.4.	Mandatory

11.7.4 Request message body

The request message body fields for enqueuing a new queue object value using CDMI are shown in Table 120.

Table 120: Request message body - Enqueue a new queue object value using CDMI

Field Name	Type	Description	Requirement
`mimetype`	JSON array of JSON strings	MIME type(s) of the data value(s) to be enqueued into the queue object. • If this field is not included and multi-part MIME is not being used, the value of "`text/plain`" shall be assigned as the field value. • If this field is not included and multi-part MIME is being used, the value of the "`Content-Type`" header of the corresponding MIME part shall be assigned as the field value. • The same number of array elements shall be present as is present in the value field, and the mimetype field shall be associated with the value in the corresponding position. • This mimetype field value shall be converted to lower case before being stored.	Optional
`copy`	JSON string	URI of a source CDMI data object or queue object from which the value shall be copied and enqueued. • If a copy source object URI to a data object is provided, the value, mimetype, and valuetransferencoding field values from the source data object are used to enqueue the new item into the destination queue object. • If a copy source object URI to a queue object is provided, the corresponding value, mimetype, and valuetransferencoding field values of the specified number of enqueued items in the source queue object are copied to the destination queue object.	Optional[3]
`move`	JSON string	URI of a source CDMI data object or queue object from which the value shall be moved and enqueued. • If a move source object URI to a data object is provided, the value, mimetype, and valuetransferencoding field values from the source data object are used to enqueue the new item into the destination queue object, and the source data object is atomically deleted. • If a move source object URI to a queue object is provided, the corresponding value, mimetype, and valuetransferencoding field values of the specified number of enqueued items in the source queue object are transferred to the destination queue object and atomically removed from the source queue object.	Optional[3]

Continued on next page

Table 120 – continued from previous page

Field Name	Type	Description	Requirement
valuetransferencoding	JSON array of JSON strings	The value transfer encoding used for the queue object value. Two value transfer encodings are defined: • "utf-8" indicates that the queue object value contains a valid UTF-8 string, and shall be transported as a UTF-8 string in the value field. • "base64" indicates that the queue object value may contain arbitrary binary sequences, and shall be transported as a base 64 encoded string in the value field. Setting the contents of the queue object value field to any value other than a valid base 64 string shall result in an HTTP status code of 400 Bad Request being returned to the client. • "json" indicates that the queue object value contains a valid JSON object, and the value field shall contain a JSON object. Setting the contents of the queue object value field to any value other than a valid JSOM object shall result in an HTTP status code of 400 Bad Request being returned to the client. • If this field is not included and multi-part MIME is not being used, the value of "utf-8" shall be assigned as the field value. • If this field is not included and multi-part MIME is being used, the value of "utf-8" shall be assigned as the field value if the "Content-Type" header of the corresponding MIME part includes the charset parameter as defined in RFC 2046 of "utf-8" (e.g., "; charset=utf-8"). Otherwise, the value of "base64" shall be assigned as the field value. This field applies only to the encoding of the value when represented in JSON; the "Content-Transfer-Encoding" header of the part specifies the encoding of the value within a multi-part MIME request, as defined in RFC 2045 [9].	Optional
value	JSON array of JSON strings	Data to be enqueued into the queue object. • If this field is not included and multi-part MIME is being used, the contents of the MIME parts shall be assigned as the field value. • If the corresponding valuetransferencoding field indicates UTF-8 encoding, the value shall be a UTF-8 string escaped using the JSON escaping rules described in RFC 4627 [5]. • If the corresponding valuetransferencoding field indicates base 64 encoding, the value shall be first encoded using the base 64 encoding rules as described in RFC 4648 [19]. • If the corresponding valuetransferencoding field indicates JSON encoding, the value shall contain a JSON object.	Optional[3]

[3] Only one of these fields shall be specified in any given operation. Except for value, these fields shall not be stored. If more than one of these fields is supplied, the server shall respond with an HTTP status code of 400 Bad Request.

11.7.5 Response headers

Response headers can be provided as per RFC 2616 [23].

11.7.6 Response message body

A response body can be provided as per RFC 2616 [23].

11.7.7 Response status

Table 121 describes the HTTP status codes that occur when enqueuing a new queue object using CDMI.

Table 121: HTTP status codes - Enqueue a new queue object value Using CDMI

HTTP Status	Description
204 No Content	The new queue object values were enqueued.
400 Bad Request	The request contains invalid parameters or field names.
401 Unauthorized	The authentication credentials are missing or invalid.
403 Forbidden	The client lacks the proper authorization to perform this request.
404 Not Found	The resource was not found at the specified URI.
409 Conflict	The operation conflicts with a non-CDMI access protocol lock or has caused a state transition error on the server.

11.7.8 Examples

EXAMPLE 1: POST to the queue object URI a new value:

```
--> POST /cdmi/2.0.0/MyContainer/MyQueue HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-queue
-->
--> {
-->   "mimetype" : [
-->     "text/plain"
-->   ],
-->   "value" : [
-->     "Value to Enqueue"
-->   ]
--> }

<-- HTTP/1.1 204 No Content
```

EXAMPLE 2: POST to the queue object URI to copy an existing value:

```
--> POST /cdmi/2.0.0/MyContainer/MyQueue HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-object
-->
--> {
-->   "copy" : "/MyContainer/MyDataObject.txt"
--> }

<-- HTTP/1.1 204 No Content
```

EXAMPLE 3: POST to the queue object URI to transfer 20 values from another queue object:

```
--> POST /cdmi/2.0.0/MyContainer/MyQueue HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-object
```

(continues on next page)

(continued from previous page)

```
-->
--> {
-->   "move" : "/MyContainer/FirstQueue?values=20"
--> }

<-- HTTP/1.1 204 No Content
```

EXAMPLE 4: POST to the queue object URI two new values:

```
--> POST /cdmi/2.0.0/MyContainer/MyQueue HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-object
-->
--> {
-->   "mimetype" : [
-->     "text/plain",
-->     "text/plain"
-->   ],
-->   "value" : [
-->     "First",
-->     "Second"
-->   ]
--> }

<-- HTTP/1.1 204 No Content
```

EXAMPLE 5: POST to the queue object URI two new values, one with base 64 transfer encoding and one with utf-8 transfer encoding:

```
--> POST /cdmi/2.0.0/MyContainer/MyQueue HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-object
-->
--> {
-->   "mimetype": [
-->     "text/plain",
-->     "text/plain",
-->     "application/json"
-->   ],
-->   "valuetransferencoding": [
-->     "utf-8",
-->     "base64",
-->     "json"
-->   ],
-->   "value": [
-->     "First",
-->     "U2Vjb25k",
-->     {
-->       "value" : "test"
-->     }
-->   ]
--> }

<-- HTTP/1.1 204 No Content
```

EXAMPLE 6: POST to the queue object URI the binary contents of two new values using multi-part MIME:

```
--> POST /cdmi/2.0.0/MyContainer/MyQueue HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: multipart/mixed; boundary=gc0p4Jq0M2Yt08j34c0p
-->
--> --gc0p4Jq0M2Yt08j34c0p
--> Content-Type: application/cdmi-queue
-->
--> {}
-->
--> --gc0p4Jq0M2Yt08j34c0p
--> Content-Type: application/octet-stream
```

(continues on next page)

(continued from previous page)

```
--> Content-Transfer-Encoding: binary
-->
--> <20 bytes of binary data>
-->
--> --gc0p4Jq0M2Yt08j34c0p
--> Content-Type: application/octet-stream
--> Content-Transfer-Encoding: binary
-->
--> <37 bytes of binary data>
-->
--> --gc0p4Jq0M2Yt08j34c0p--

<-- HTTP/1.1 204 No content
```

EXAMPLE 7: POST to the queue object URI the mime types and binary contents of two new values using multi-part MIME:

```
--> POST /cdmi/2.0.0/MyContainer/MyQueue HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: multipart/mixed; boundary=gc0p4Jq0M2Yt08j34c0p
-->
--> --gc0p4Jq0M2Yt08j34c0p
--> Content-Type: application/cdmi-queue
-->
--> {
-->   "mimetype" : [
-->     "application/pdf",
-->     "image/jpeg"
-->   ]
--> }
-->
--> --gc0p4Jq0M2Yt08j34c0p
--> Content-Type: application/octet-stream
--> Content-Transfer-Encoding: binary
-->
--> <20 bytes of binary data>
-->
--> --gc0p4Jq0M2Yt08j34c0p
--> Content-Type: application/octet-stream
--> Content-Transfer-Encoding: binary
-->
--> <37 bytes of binary data>
-->
--> --gc0p4Jq0M2Yt08j34c0p--

<-- HTTP/1.1 204 No content
```

11.8 Delete a queue object value using CDMI

11.8.1 Synopsis

To delete one or more of the oldest enqueued values in an existing queue, the following request shall be performed:

- `DELETE <root URI>/<ContainerName>/<QueueName>?value`
- `DELETE <root URI>/<ContainerName>/<QueueName>?values=<count>`
- `DELETE <root URI>/<ContainerName>/<QueueName>?values=<range>`
- `DELETE <root URI>/cdmi_objectid/<QueueObjectID>?value`
- `DELETE <root URI>/cdmi_objectid/<QueueObjectID>?values=<count>`
- `DELETE <root URI>/cdmi_objectid/<QueueObjectID>?values=<range>`

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<ContainerName>` is zero or more intermediate containers.
- `<QueueName>` is the name of the queue object to be deleted from.
- `<QueueObjectID>` is the ID of the queue object to be deleted from.
- `<count>` is the number of values, starting from the oldest, to be removed from the queue object. If more queue object entries are requested to be deleted than exist in the queue object, the count shall be considered equal to the number of entries in the queue object.
- `<range>` is the lowest to highest numbers as found in the queueValues field that are to be removed from the queue object. The first range value shall be smaller or equal to the lowest queue value. If the first range value is smaller than the lowest queue value, the lowest existing queue value shall be used. If the first range value is larger than the lowest queue value, an HTTP status code of `400 Bad Request` shall be returned to the client. If the second range value is higher than the highest existing queue value, the highest existing queue value shall be used, which allows for idempotent queue value deletion.

The “?value” suffix at the end of the queue resource URI shall be included to distinguish the deletion of the oldest value from the deletion of the queue object itself, as described in 11.6 (which deletes all enqueued values).

11.8.2 Capabilities

Capabilities that indicate which operations are supported are shown in Table 122.

Table 122: Capabilities - Delete a queue object value using CDMI

Capability	Location	Description
`cdmi_modify_value`	Queue object	Ability to delete a value from an existing queue object
`cdmi_object_access_by_ID`	System wide capability	Ability to access the object by ID

11.8.3 Request header

Request headers can be provided as per RFC 2616 [23].

11.8.4 Request message body

A request body can be provided as per RFC 2616 [23].

11.8.5 Response headers

Response headers can be provided as per RFC 2616 [23].

11.8.6 Response message body

A response body can be provided as per RFC 2616 [23].

11.8.7 Response status

Table 123 describes the HTTP status codes that occur when deleting a queue object value using CDMI.

Table 123: HTTP status codes - Delete a queue object value using CDMI

HTTP Status	Description
`204 No Content`	The queue object value was successfully deleted.
`400 Bad Request`	The request contains invalid parameters or field names.
`401 Unauthorized`	The authentication credentials are missing or invalid.
`403 Forbidden`	The client lacks the proper authorization to perform this request.
`404 Not Found`	The resource was not found at the specified URI.
`409 Conflict`	The operation conflicts with a non-CDMI access protocol lock or has caused a state transition error on the server.

11.8.8 Examples

EXAMPLE 1: DELETE to the queue object URI value to delete the oldest enqueued value:

```
--> DELETE /cdmi/2.0.0/MyContainer/MyQueue?value HTTP/1.1
--> Host: cloud.example.com

<-- HTTP/1.1 204 No Content
```

EXAMPLE 2: DELETE to the queue object URI value to remove the ten oldest values:

```
--> DELETE /cdmi/2.0.0/MyContainer/MyQueue?values=10 HTTP/1.1
--> Host: cloud.example.com

<-- HTTP/1.1 204 No Content
```

EXAMPLE 3: DELETE to the queue object URI value to remove queue values 10 through 19:

```
--> DELETE /cdmi/2.0.0/MyContainer/MyQueue?values=10-19 HTTP/1.1
--> Host: cloud.example.com

<-- HTTP/1.1 204 No Content
```

Clause 12

Capability object resource operations using CDMI

12.1 Overview

Capability objects indicate what specific functionality and operations are supported by a given CDMI server, and allow CDMI clients to discover what subset of this International Standard is implemented.

All CDMI servers shall support capabilities and the ability for CDMI clients to read capabilities.

Each CDMI capability object is represented as a JSON object, containing one or more "fields". For example, the "`capabilities`" field contains specific capability items.

EXAMPLE 1: CDMI capability object

```
{
  "objectType": "application/cdmi-capability",
  "objectID": "00007E7F00104BE66AB53A9572F9F51E",
  "objectName": "cdmi_capabilities/",
  "parentURI": "/",
  "parentID": "00007E7F0010128E42D87EE34F5A6560",
  "capabilities": {
      "cdmi_domains": "true",
      "cdmi_export_nfs": "true",
      "cdmi_export_iscsi": "true",
      "cdmi_queues": "true",
      "cdmi_notification": "true",
      "cdmi_query": "true",
      "cdmi_metadata_maxsize": "4096",
      "cdmi_metadata_maxitems": "1024"
  },
  "childrenrange": "0-3",
  "children": [
    "domain/",
    "container/",
    "dataobject/",
    "queue/"
  ]
}
```

The meaning, use, and permitted values of each field is described in 12.3.

12.2 Capability object details

12.2.1 Capability object addressing

Capability objects are addressed in CDMI in two ways:

- by name (e.g. `https://cloud.example.com/cdmi/2.0.0/cdmi_capabilities/`); and
- by ID (e.g. `https://cloud.example.com/cdmi/2.0.0/cdmi_objectid/00007E7F00104BE66AB53A9572F9F51E/`).

Every capability object has a single, globally-unique object ID that remains constant for the life of the object. Each capability object may also have one or more URI addresses that allow the capability object to be accessed.

When a capability object is addressed via more than one unique URIs, all operations may be performed through any of these URIs. For example, a capability object may be accessible via multiple virtual hosting paths, where `https://cloud.example.com/cdmi/2.0.0/users/snia/cdmi/cdmi_capabilities/` is also accessible through `https://snia.example.com/cdmi/2.0.0/cdmi_cabilities/`.

Following the URI conventions for hierarchical paths, capability URIs shall consist of one or more capability names that are separated by forward slashes ("`/`") and that end with a forward slash ("`/`").

If a request is performed against an existing capability resource and the trailing slash at the end of the URI is omitted, the server shall respond with an HTTP status code of `301 Moved Permanently`. In addition, a `Location` header containing the URI with the trailing slash added shall be returned.

Capabilities may also be nested.

EXAMPLE 2: The following URI represents a nested capability:

`https://cloud.example.com/cdmi/2.0.0/cdmi_capabilities/container/`

A nested capability has a parent capability object, and shall be included in the children field of the parent capability object.

12.2.2 Capability object fields

Every CDMI object (excluding capability objects) includes a server-generated "`capabilitiesURI`" field that contains the URI of the capabilities object that describes which operations are permitted for that CDMI object.

Fig. 8 (shown on the next page) shows the hierarchy of capabilities and shows how the capabilitiesURI links data objects, container objects, queue objects and domain objects into the capabilities tree.

System-wide capabilities are described by the root capabilities object, which is accessible at "`<root URI>/cdmi_capabilities/`".

Capabilities cannot be altered by clients, but may be changed by the CDMI server to reflect configuration changes or operational changes. For example, if a CDMI server is upgraded or reconfigured, additional capabilities may become present, or existing capabilties may no longer be present. In practice, capabilities rarely change, and a client can assume that they shall remain constant for the duration of a client-server HTTP/HTTPS session.

Cloud clients should use capabilities to discover what operations are supported. If an operation is attempted on a CDMI object that does not have a corresponding capability, an HTTP status code of `400 Bad Request` shall be returned to the client.

The capabilities defined as part of this International Standard are described starting in 12.2.7. Vendor-defined capabilities not specified in this International Standard shall not start with "`cdmi_`".

12.2.3 Capability object metadata

Capability objects do not have metadata.

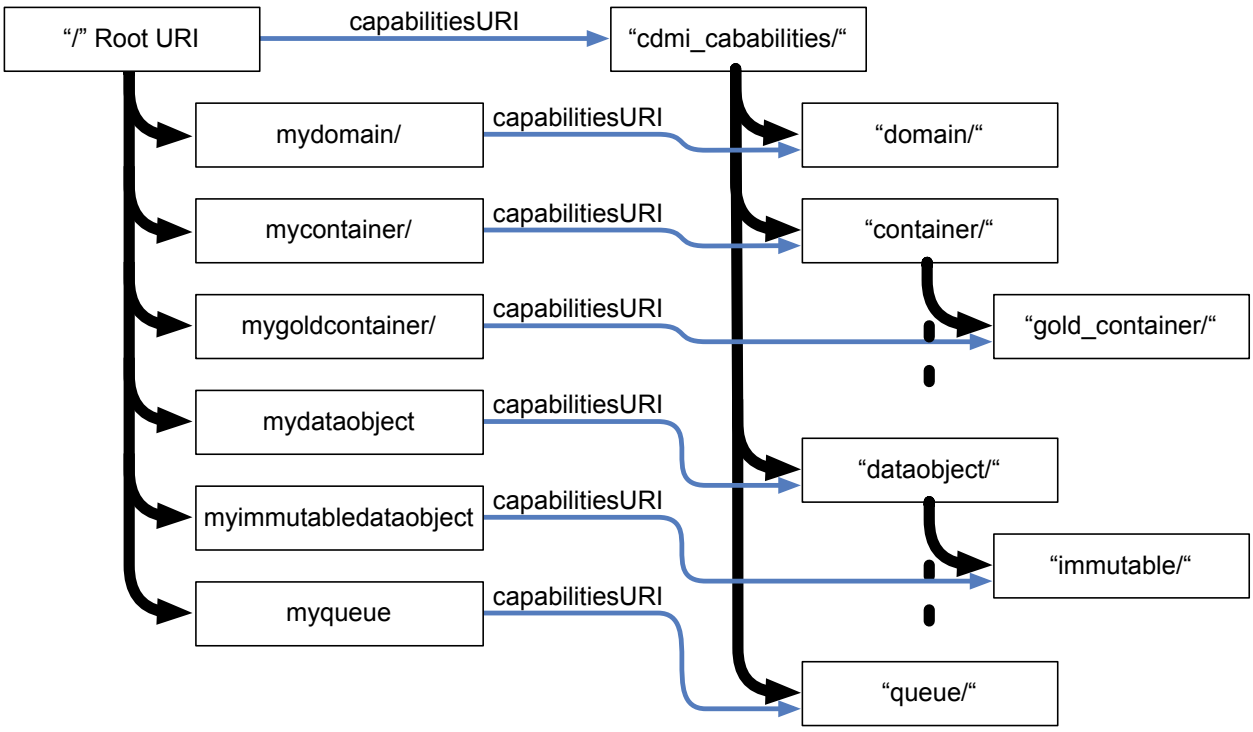


Fig. 8: Hierarchy of capabilities

12.2.4 Capability object access control

Capability objects are not subject to CDMI ACLs. Any authenticated CDMI client shall be capable of reading all Capability objects[1] .

Capabilities may differ from the operations permitted by an Access Control List (ACL) (see 17.1) associated with a given object. For example, a read-only cloud may not permit write access to a container or object, despite the presence of an ACL allowing write access.

12.2.5 Queue object consistency

Capabilitiy objects are read-only.

12.2.6 Capability object representations

The representations in this clause are shown using JSON notation. Both clients and servers shall support UTF-8 JSON representation. The request and response body JSON fields may be specified or returned in any order, with the exception that, if present, for capability objects, the "`childrenrange` and "`children fields` shall appear last and in that order.

[1] A CDMI Server may filter the visibility of capability objects and/or capability items for security purposes, for example, to prevent the client discovery of the names and characteristics of classification levels above the client's maximum classification level. Such filtering is out of scope of this International Standard.

12.2.7 Cloud storage system-wide capabilities

Table 124 defines the system-wide capabilities in a cloud storage system. These capabilities, which are found in the capabilities object, are referred to by the root URI (root capabilities).

Table 124: System-wide capabilities

Capability name	Type	Definition
cdmi_domains	JSON string	If present and "true", the CDMI server supports domains. If not present, the domainURI field shall not be present in response bodies and the "cdmi_domains" URI shall not be present.
cdmi_export_smb	JSON string	If present and "true", the CDMI server supports SMB exports.
cdmi_dataobjects	JSON string	If present and "true", the CDMI server supports data objects.
cdmi_export_iscsi	JSON string	If present and "true", the CDMI server supports iSCSI exports.
cdmi_export_nfs	JSON string	If present and "true", the CDMI server supports NFS protocol exports.
cdmi_export_occi_iscsi	JSON string	If present and "true", the CDMI server supports OCCI/iSCSI exports.
cdmi_export_webdav	JSON string	If present and "true", the CDMI server supports WebDAV exports.
cdmi_metadata_maxitems	JSON string	If present, this capability indicates the maximum number of user-defined metadata items supported per object. If not present, there is no limit placed on the number of user-defined metadata items.
cdmi_metadata_maxsize	JSON string	If present, this capability indicates the maximum size, in bytes, of each user-defined metadata item supported per object. If not present, there is no limit placed on the size of user-defined metadata items.
cdmi_metadata_maxtotalsize	JSON string	If present, this capability indicates the maximum size, in bytes, of user-defined metadata supported by the CDMI server. If not present, there is no limit placed on the size of user-defined metadata.
cdmi_notification	JSON string	If present and "true", the CDMI server supports notification queues.
cdmi_logging	JSON string	If present and "true", the CDMI server supports logging queues.
cdmi_query	JSON string	If present and "true", the CDMI server supports query queues.
cdmi_query_regex	JSON string	If present and "true", the CDMI server supports query with regular expressions.
cdmi_query_contains	JSON string	If present and "true", the CDMI server supports query with "contains" expressions.
cdmi_query_tags	JSON string	If present and "true", the CDMI server supports query with tag-matching expressions.
cdmi_query_value	JSON string	If present and "true", the CDMI server supports query of value fields.
cdmi_queues	JSON string	If present and "true", the CDMI server supports queue objects.
cdmi_security_access_control	JSON string	If present and "true", the CDMI server supports ACLs. See 12.2.9 for additional information.

Continued on next page

Table 124 – continued from previous page

Capability name	Type	Definition
cdmi_security_data_integrity	JSON string	If present and "true", the CDMI server supports data integrity/authenticity. See 12.2.9 for additional information.
cdmi_security_encryption	JSON string	If present and "true", the CDMI server supports data at-rest encryption. See 12.2.9 for additional information.
cdmi_security_immutability	JSON string	If present and "true", the CDMI server supports data immutability/retentions. See 12.2.9 for additional information.
cdmi_security_sanitization	JSON string	If present and "true", the CDMI server supports data/media sanitization. See 12.2.9 for additional information.
cdmi_serialization_json	JSON string	If present and "true", the CDMI server supports JSON as a serialization format.
cdmi_snapshots	JSON string	If present and "true", the CDMI server supports snapshots.
cdmi_references	JSON string	If present and "true", the CDMI server supports references.
cdmi_object_move_from_local	JSON string	If present and "true", the CDMI server supports moving CDMI objects from URIs within the same storage system.
cdmi_object_move_from_remote	JSON string	If present and "true", the CDMI server supports moving CDMI objects from URIs within other CDMI storage systems.
cdmi_object_move_from_ID	JSON string	If present and "true", the CDMI server supports moving CDMI objects without a path from a /cdmi_objectid/ URI within the same storage system. This effectively adds a path, allowing the object to be accessed by ID and by path.
cdmi_object_move_to_ID	JSON string	If present and "true", the CDMI server supports moving CDMI objects with a path to a /cdmi_objectid/ URI within the same storage system. This effectively removes the path, leaving the object only accessible by ID.
cdmi_object_copy_from_local	JSON string	If present and "true", the CDMI server supports copying CDMI objects from URIs within the same storage system.
cdmi_object_copy_from_remote	JSON string	If present and "true", the CDMI server supports copying CDMI objects from URIs within other CDMI storage systems.
cdmi_object_access_by_ID	JSON string	If present and "true", the CDMI server supports accessing, updating, and deleting objects through /cdmi_objectid/.
cdmi_post_dataobject_by_ID	JSON string	If present and "true", the CDMI server supports adding a new data object by ID via POST to "/cdmi_objectid/".
cdmi_post_queue_by_ID	JSON string	If present and "true", the CDMI server supports adding a new queue object by ID via POST to "/cdmi_objectid/".
cdmi_deserialize_dataobject_by_ID	JSON string	If present and "true", the CDMI server supports deserializating serialized data objects when creating a new data object by ID via POST to "/cdmi_objectid/".
cdmi_deserialize_queue_by_ID	JSON string	If present and "true", the CDMI server supports deserializating serialized queue objects when creating a new queue object by ID via POST to "/cdmi_objectid/".

Continued on next page

Table 124 – continued from previous page

Capability name	Type	Definition
cdmi_serialize_dataobject_to_ID	JSON string	If present and "true", the CDMI server supports serializing data objects when creating a new data object by ID via POST to "/cdmi_objectid/".
cdmi_serialize_domain_to_ID	JSON string	If present and "true", the CDMI server supports serializing domain objects when creating a new data object by ID via POST to "/cdmi_objectid/".
cdmi_serialize_container_to_ID	JSON string	If present and "true", the CDMI server supports serializing container objects when creating a new data object by ID via POST to "/cdmi_objectid/".
cdmi_serialize_queue_to_ID	JSON string	If present and "true", the CDMI server supports serializing queue objects when creating a new data object by ID via POST to "/cdmi_objectid/".
cdmi_copy_dataobject_by_ID	JSON string	If present and "true", the CDMI server supports copying an existing data object when creating a new data object by ID via POST to "/cdmi_objectid/".
cdmi_copy_queue_by_ID	JSON string	If present and "true", the CDMI server supports copying an existing queue object when creating a new queue object by ID via POST to "/cdmi_objectid/".
cdmi_create_reference_by_ID	JSON string	If present and "true", the CDMI server supports creating a new reference via POST to "/cdmi_objectid/".
cdmi_copy_dataobject_from_queue	JSON string	If present and "true", the CDMI server supports the ability to copy to a data object from a queue object.
cdmi_multipart_mime	JSON string	If present and "true", the CDMI server supports storing and retrieving the value of data and queue objects using multi-part MIME.
cdmi_create_value_range_by_ID	JSON string	If present and "true", the CDMI server supports a new data object's value to be created with byte ranges through "/cdmi_objectid/".
cdmi_dac	JSON string	If present and "true", the CDMI server supports delegated access control.
cdmi_dac_methods	JSON array of JSON strings	If present, this capability contains a list of URI schemes supported for DAC URIs, as specified in the IANA URI Schemes registry. The following schemes shall be supported: • "https" – mandatory for all DAC implementations The following schemes may be supported: • "http" – optional for DAC implementations • "mailto" – optional for DAC implementations
cdmi_enc_cms	JSON string	If present and "true", the CDMI server supports operations against the contents of CMS encrypted objects.
cdmi_enc_jwe	JSON string	If present and "true", the CDMI server supports operations against the contents of JWE encrypted objects.
cdmi_enc_inplace	JSON string	If present and "true", the CDMI server supports operations to encrypt and decrypt objects in place, including updates.
cdmi_enc_access	JSON string	If present and "true", the CDMI server supports operations to decrypt objects on access.
cdmi_cms_encryption	JSON array of JSON strings	If present, this capability lists which CMS ContentEncryptionAlgorithmIdentifier encryption algorithms are supported for operations against the contents of CMS encrypted objects.

Continued on next page

Table 124 – continued from previous page

Capability name	Type	Definition
cdmi_cms_digest	JSON array of JSON strings	If present, this capability lists which CMS MessageAuthenticationCodeAlgorithm digest algorithms are supported for operations against the contents of CMS encrypted objects.
cdmi_cms_signature	JSON array of JSON strings	If present, this capability lists which CMS SignatureAlgorithmIdentifier signature algorithms are supported for operations against the contents of CMS encrypted objects.
cdmi_jwe_enc	JSON array of JSON strings	If present, this capability lists which JOSE "enc" encryption algorithms are supported for operations against the contents of JWE encrypted objects, as defined in RFC 7518 [15].
cdmi_jwe_alg	JSON array of JSON strings	If present, this capability lists which JOSE "alg" encryption algorithms are supported for operations against the contents ofJWE encrypted objects, as defined in RFC 7518 [15].
cdmi_jws_alg	JSON array of JSON strings	If present, this capability lists which JOSE "alg" encryption algorithms are supported for operations against the contents of JWS signatures, as defined in RFC 7518 [15].
cdmi_valuetransferencoding_json	JSON string	If present and "true", the CDMI server supports JSON value transfer encodings.

12.2.8 Storage system metadata capabilities

Table 125 defines the capabilities for storage system metadata in a cloud storage system. These capabilities are found in the capabilities objects for domain objects, data objects, container objects, and queue objects. See 16.2 for a description of these storage system metadata items.

Table 125: Capabilities for storage system metadata

Capability name	Type	Definition
cdmi_acl	JSON string	If present and "true", the CDMI server supports ACLs. When a CDMI implementation supports ACLs for the purpose of access control, the system-wide capability of cdmi_security_access_control specified in 12.2.7 of 12.2.7 shall also be set to "true". If not present, there is no support for ACL-based access control.
cdmi_size	JSON string	If present and "true", the CDMI server shall generate a cdmi_size storage system metadata for each stored object.
cdmi_ctime	JSON string	If present and "true", the CDMI server shall generate a cdmi_ctime storage system metadata for each stored object.
cdmi_atime	JSON string	If present and "true", the CDMI server shall generate a cdmi_atime storage system metadata for each stored object.
cdmi_mtime	JSON string	If present and "true", the CDMI server shall generate a cdmi_mtime storage system metadata for each stored object.
cdmi_acount	JSON string	If present and "true", the CDMI server shall generate a cdmi_acount storage system metadata for each stored object.
cdmi_mcount	JSON string	If present and "true", the CDMI server shall generate a cdmi_mcount storage system metadata for each stored object.
cdmi_dac_uri	JSON string	If present and "true", the CDMI server supports delegated access control metadata.
cdmi_dac_certificate	JSON string	If present and "true", the CDMI server supports delegated access control metadata.
cdmi_enc_signature	JSON string	If present and "true", the CDMI server shall generate a cdmi_signature storage system metadata for each stored object when a corresponding sign_id data system metadata item is present.
cdmi_version_object	JSON string	If present and "true", the CDMI server shall generate a cdmi_version_object storage system metadata for each version-enabled data object and data object version.
cdmi_version_current	JSON string	If present and "true", the CDMI server shall generate a cdmi_version_current storage system metadata for each version-enabled data object and data object version.
cdmi_version_oldest	JSON array of JSON strings	If present and "true", the CDMI server shall generate a cdmi_version_oldest storage system metadata for each version-enabled data object and data object version.
cdmi_version_parent	JSON string	If present and "true", the CDMI server shall generate a cdmi_version_parent storage system metadata for each data object version that has a previous version.

Continued on next page

Table 125 – continued from previous page

Capability name	Type	Definition
cdmi_version_children	JSON array of JSON strings	If present and "true", the CDMI server shall generate a cdmi_version_children storage system metadata for each data object version.

12.2.9 Data system metadata capabilities

Table 126 defines the capabilities that indicate which data system metadata items are interpreted for objects stored in a cloud storage system. These capabilities are found in the capabilities objects for domains, data objects, containers, and queues. See 16.3 for a description of the meaning of the corresponding data system metadata items.

Table 126: Capabilities for data system metadata

Capability name	Type	Definition
cdmi_assignedsize	JSON string	If present and "true", the CDMI server supports the cdmi_assignedsize data system metadata as defined in 16.3.
cdmi_data_redundancy	JSON string	If present, the CDMI server supports the cdmi_data_redundancy data system metadata as defined in 16.3. The value of the capability shall be set to a positive numeric string representing the maximum value that the server supports.
cdmi_data_dispersion	JSON string	If present and "true", the CDMI server supports the cdmi_data_dispersion data system metadata as defined in 16.3.
cdmi_data_retention	JSON string	If present and "true", the CDMI server supports both the cdmi_retention_id and cdmi_retention_period data system metadata as defined in 16.3.
cdmi_data_autodelete	JSON string	If present and "true", the CDMI server supports the cdmi_data_autodelete data system metadata as defined in 16.3.
cdmi_data_holds	JSON string	If present and "true", the CDMI server supports the cdmi_hold_id data system metadata as defined in 16.3. When a cloud storage system supports holds for the purpose of making data immutable, the system-wide capability of cdmi_security_immutability specified in Table 124 of 12.2.7 shall be present and set to "true".
cdmi_encryption	JSON array of JSON strings	If present, the CDMI server supports the cdmi_encryption data system metadata as defined in 16.3. When present, this capability shall contain one or more JSON strings, each string corresponding to an algorithm/mode/length value as described in the cdmi_encryption data system metadata in 16.3. When a cloud storage system supports at-rest encryption, the system-wide capability of cdmi_security_encryption specified in Table 124 of 12.2.7 shall be present and set to "true".
cdmi_geographic_placement	JSON string	If present and "true", the CDMI server supports the cdmi_geographic_placement data system metadata as defined in 16.3.
cdmi_immediate_redundancy	JSON string	If present, the CDMI server supports the cdmi_immediate_redundancy data system metadata as defined in 16.3. When present, this capability shall contain a string set to a positive numeric string representing the maximum value that the server supports.

Continued on next page

Table 126 – continued from previous page

Capability name	Type	Definition
cdmi_infrastructure_redundancy	JSON string	If present, the CDMI server supports the cdmi_infrastructure_redundancy data system metadata as defined in 16.3. When present, this capability shall contain a string set to a positive numeric string representing the maximum value that the server supports.
cdmi_latency	JSON string	If present and "true", the CDMI server supports the cdmi_latency data system metadata as defined in 16.3.
cdmi_RPO	JSON string	If present and "true", the CDMI server supports the cdmi_RPO data system metadata as defined in 16.3.
cdmi_RTO	JSON string	If present and "true", the CDMI server supports the cdmi_RTO data system metadata as defined in 16.3
cdmi_sanitization_method	JSON array of JSON strings	If present, the CDMI server supports the cdmi_sanitization_method data system metadata as defined in 16.3. When present, this capability shall contain one or more JSON strings, each string corresponding to a sanitization method as described in the cdmi_sanitization_method data system metadata in 16.3. When a cloud storage system supports sanitization, the system-wide capability of cdmi_security_sanitization specified in Table 124 of 12.2.7 shall be present and set to "true".
cdmi_throughput	JSON string	If present and "true", the CDMI server supports the cdmi_throughput data system metadata as defined in 16.3.
cdmi_value_hash	JSON array of JSON strings	If present, the CDMI server supports the cdmi_value_hash data system metadata as defined in 16.3. When present, this capability shall contain one or more JSON strings, each string corresponding to an algorithm/length value as described in the cdmi_value_hash data system metadata in 16.3. When a cloud storage system supports value hashing, the system-wide capability of cdmi_security_data_integrity specified in Table 124 of 12.2.7 shall be present and set to "true".
cdmi_enc_key_id	JSON string	When the cloud storage system supports the cdmi_enc_key_id data system metadata as defined in clause 16.3, the cdmi_enc_key_id capability shall be present and set to the string value "true". When this capability is absent, or present and set to the string value "false", cdmi_enc_key_id data system metadata shall not be used.
cdmi_enc_value_sign_id	JSON string	When the cloud storage system supports the cdmi_enc_value_sign_id data system metadata as defined in clause 16.3, the cdmi_enc_value_sign_id capability shall be present and set to the string value "true". When this capability is absent, or present and set to the string value "false", cdmi_enc_value_sign_id data system metadata shall not be used.

Continued on next page

Table 126 – continued from previous page

Capability name	Type	Definition
cdmi_enc_value_verify_id	JSON string	When the cloud storage system supports the cdmi_enc_value_verify_id data system metadata as defined in clause 16.3, the cdmi_enc_value_verify_id capability shall be present and set to the string value "true". When this capability is absent, or present and set to the string value "false", cdmi_enc_value_verify_id data system metadata shall not be used.
cdmi_enc_object_sign_id	JSON string	When the cloud storage system supports the cdmi_enc_object_sign_id data system metadata as defined in clause 16.3, the cdmi_enc_object_sign_id capability shall be present and set to the string value "true". When this capability is absent, or present and set to the string value "false", cdmi_enc_object_sign_id data system metadata shall not be used.
cdmi_enc_object_verify_id	JSON string	When the cloud storage system supports the cdmi_enc_object_verify_id data system metadata as defined in clause 16.3, the cdmi_enc_object_verify_id capability shall be present and set to the string value "true". When this capability is absent, or present and set to the string value "false", cdmi_enc_object_verify_id data system metadata shall not be used.
cdmi_versioning	JSON array of JSON strings	If present, this capability indicates that the cloud storage system shall support versioning of data objects and contains a list of which versioning behaviors are supported. The following values are defined: • "value" indicates that the system shall support the versioning of the object value. • "user" indicates that the system shall support the versioning of the object value and user metadata. • "all" indicates that the system shall support the versioning of all updates made to a data object. When present, the system shall support the following storage system metadata: cdmi_version_object, cdmi_version_current, cdmi_version_oldest, cdmi_version_parent, and cdmi_version_children as indicated by the corresponding storage system metadata capabilities.
cdmi_versions_count	JSON string	If present, this capability specifies the maximum number of historical versions that may be specified. If absent, restrictions on the number of historical versions specified shall be ignored.
cdmi_version_age	JSON string	If present, this capability specifies the maximum age of historical versions that may be specified. If absent, restrictions on the age of historical versions specified shall be ignored.
cdmi_versions_size	JSON string	If present, this capability specifies the maximum total size of historical versions that may be specified. If absent, restrictions on the size of historical versions specified shall be ignored.

12.2.10 Data object capabilities

Table 127 defines the capabilities for data objects in a cloud storage system.

Table 127: Capabilities for data objects

Capability name	Type	Definition
`cdmi_read_value`	JSON string	If present and "`true`", this capability indicates that the CDMI server shall support the ability to read the object's value.
`cdmi_read_value_range`	JSON string	If present and "`true`", this capability indicates that the CDMI server shall support the ability to read the object's value with byte ranges.
`cdmi_read_metadata`	JSON string	If present and "`true`", this capability indicates that the CDMI server shall support the ability to read the object's metadata.
`cdmi_modify_value`	JSON string	If present and "`true`", this capability indicates that the CDMI server shall support the ability to modify the object's value.
`cdmi_modify_value_range`	JSON string	If present and "`true`", this capability indicates that the CDMI server shall support the ability to modify the object's value with byte ranges.
`cdmi_modify_metadata`	JSON string	If present and "`true`", this capability indicates that the CDMI server shall support the ability to modify the object's metadata.
`cdmi_modify_deserialize_dataobject`	JSON string	If present and "`true`", this capability indicates that the CDMI server shall support the ability of the data object to deserialize a serialized data object into the data object as an update.
`cdmi_delete_dataobject`	JSON string	If present and "`true`", this capability indicates that the CDMI server shall support the ability to delete the object.

12.2.11 Container object capabilities

Table 128 defines the capabilities for containers in a cloud storage system.

Table 128: Capabilities for container objects

Capability name	Type	Definition
cdmi_list_children	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to list the container's children.
cdmi_list_children_range	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to list the container's children with ranges.
cdmi_read_metadata	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to read the container's metadata.
cdmi_modify_metadata	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to modify the container's metadata.
cdmi_modify_deserialize_container	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability of the container object to deserialize a serialized container object into the container object as an update.
cdmi_snapshot	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability of the container object to create a new snapshot.
cdmi_serialize_dataobject	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to serialize a data object.
cdmi_serialize_container	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to serialize the container and all children's contents.
cdmi_serialize_queue	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to serialize a queue object.
cdmi_serialize_domain	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to serialize the domain and all child domains.
cdmi_deserialize_container	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability of the container to deserialize the serialized containers and associated serialized children into the container.
cdmi_deserialize_queue	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability of the container to deserialize the serialized queue objects into the container.
cdmi_deserialize_dataobject	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability of the container to deserialize the serialized data objects into the container.
cdmi_create_dataobject	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability of the container to add a new data object.
cdmi_post_dataobject	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability of the container to add a new data object via POST.
cdmi_post_queue	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability of the container to add a new queue object via POST.

Continued on next page

Table 128 – continued from previous page

Capability name	Type	Definition
cdmi_create_container	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to create a new container object via PUT.
cdmi_create_queue	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to create new queue objects..
cdmi_create_reference	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to create a new child reference via PUT.
cdmi_export_container_smb	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to export a container as a file system via SMB.
cdmi_export_container_nfs	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to export a container as a file system via NFS.
cdmi_export_container_iscsi	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to export a container as a file system via iSCSI.
cdmi_export_container_occi	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to export a container as a file system via OCCI.
cdmi_export_container_webdav	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to export a container as a file system via WebDAV.
cdmi_delete_container	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to delete a container.
cdmi_move_container	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to move a container object into a container.
cdmi_copy_container	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to copy a container object into a container.
cdmi_move_dataobject	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to move a data object into a container.
cdmi_copy_dataobject	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to copy a data object into a container.
cdmi_create_value_range	JSON string	If present and "true", this capability indicates that the container allows a new data object's value to be created with byte ranges.

12.2.12 Domain object capabilities

Table 129 defines the capabilities for domains in a cloud storage system. (All capabilities refer to what may be done via CDMI content-type operations.

Table 129: Capabilities for domain objects

Capability name	Type	Definition
cdmi_create_domain	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to add a new subdomain.
cdmi_delete_domain	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to delete a domain.
cdmi_move_domain	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to move a domain.
cdmi_domain_summary	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to support domain summaries.
cdmi_domain_members	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to support domain user management.
cdmi_list_children	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to list the domain's children.
cdmi_read_metadata	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to read the domain's metadata.
cdmi_modify_metadata	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to modify the domain's metadata.
cdmi_modify_deserialize_domain	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to deserialize a serialized domain object into the domain object as an update.
cdmi_copy_domain	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to copy the domain (via PUT) to another URI.
cdmi_deserialize_domain	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to deserialize serialized domains and associated serialized children into the domain.

Continued on next page

Table 129 – continued from previous page

Capability name	Type	Definition
cdmi_authentication_methods	JSON array of JSON strings	If present, the CDMI server supports authentication methods that are supported by a domain. When present, this capability shall contain one or more of the following JSON strings: • “anonymous” - Absence of authentication supported • “basic” - HTTP basic authentication supported (RFC 2617 [8]) • “digest” - HTTP digest authentication supported (RFC 2617 [8]) • “krb5” - Kerberos authentication supported, using the Kerberos domain specified in the CDMI domain (RFC 4559 [14]) • “x509” - certificate-based authentication via TLS (RFC 5246 [25], RFC 8446 [24]) • “s3” - S3 API signed header authentication supported • “openstack” - OpenStack Identity API header authentication supported Interoperability with these authentication methods are not defined by this International Standard. Servers may include other authentication methods not included in the above list. In these cases, it is up to the CDMI client and CDMI server to ensure interoperability.

12.2.13 Queue object capabilities

Table 130 defines the capabilities for queue objects in a cloud storage system.

Table 130: Capabilities for queue objects

Capability name	Type	Definition
cdmi_read_value	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to read a queue's value.
cdmi_read_metadata	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to read the queue's metadata.
cdmi_modify_value	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to modify the queue's value.
cdmi_modify_metadata	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to modify the queue's metadata.
cdmi_modify_deserialize_queue	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to deserialize a serialized queue into the queue as an update.
cdmi_delete_queue	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to delete a queue.
cdmi_move_queue	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to move a queue to another URI.
cdmi_copy_queue	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to copy a queue to another URI.
cdmi_reference_queue	JSON string	If present and "true", this capability indicates that the CDMI server shall support the ability to reference a queue from another queue.

12.3 Read a capabilities object using CDMI

12.3.1 Synopsis

To read an existing capability object, the following requests shall be performed:

- `GET <root URI>/cdmi_capabilities/<Capability>/<TheCapability>/`
- `GET <root URI>/cdmi_capabilities/<Capability>/<TheCapability>/?<fieldname>&<fieldname>&...`
- `GET <root URI>/cdmi_capabilities/<Capability>/<TheCapability>/?children=<range>&...`
- `GET <root URI>/cdmi_objectid/<CapabilityObjectID>/`
- `GET <root URI>/cdmi_objectid/<CapabilityObjectID>/?<fieldname>&<fieldname>&...`
- `GET <root URI>/cdmi_objectid/<CapabilityObjectID>/?children=<range>&...`

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<Capability>` is zero or more parent capabilities.
- `<TheCapability>` is the name specified for the capability to be read from.
- `<fieldname>` is the name of a field.
- `<range>` is a numeric range within the list of children.
- `<prefix>` is a matching prefix that returns all metadata items that start with the prefix value.
- `<CapabilityObjectID>` is the ID of the capability object to be read from.

12.3.2 Capabilities

Capabilities that indicate which operations are supported are shown in Table 131.

Table 131: Capabilities - Read a capabilities object using CDMI

Capability	Location	Description
`cdmi_object_access_by_ID`	System wide capability	Ability to access the object by ID

12.3.3 Request headers

The HTTP request headers for reading a CDMI capabilities object using CDMI are shown in Table 132.

Table 132: Request headers - Read a capabilities object using CDMI

Header	Type	Description	Requirement
`Accept`	Header string	"`application/cdmi-capability`" or a consistent value as described in 5.5.2	Optional

12.3.4 Request message body

A request body shall not be provided.

12.3.5 Response headers

The HTTP response headers for reading a CDMI capabilities object using CDMI are shown in Table 133.

Table 133: Response headers - Read a capabilities object Using CDMI

Header	Type	Description	Requirement
Content-Type	Header string	"application/cdmi-capability"	Mandatory

12.3.6 Response message body

The response message body fields for reading a CDMI capabilities object using CDMI are shown in Table 134.

Table 134: Response message body - Read a capabilities object using CDMI

Field name	Type	Description	Requirement
objectType	JSON string	"application/cdmi-capability"	Mandatory
objectID	JSON string	Object ID of the object	Mandatory
objectName	JSON string	Name of the object	Mandatory
parentURI	JSON string	URI for the parent object Appending the "objectName" to the "parentURI" shall always produce a valid URI for the object.	Mandatory
parentID	JSON string	Object ID of the parent capability object.	Mandatory
capabilities	JSON object	The capabilities supported by the corresponding object. Capabilities in the "/cdmi_capabilities/" object are system-wide capabilities. Capabilities found in children objects under "/cdmi_capabilities/" correspond to the capabilities of a specific subset of objects.	Mandatory
childrenrange	JSON string	The child capabilities of the capability expressed as a range. If a range of child capabilities is requested, this field indicates the children returned as a range.	Mandatory
children	JSON array of JSON strings	Names of the children capabilities objects. For the root container capabilities, this includes "domain/", "container/", "dataobject/", and "queue/". Within each of these capabilities objects, further more specialized capabilities profiles may be specified by the CDMI server.	Mandatory

If individual fields are specified in the GET request, only these fields are returned in the result body. Optional fields that are requested but do not exist are omitted from the result body.

12.3.7 Response status

Table 135 describes the HTTP status codes that occur when reading a capabilities object using CDMI.

Table 135: HTTP status codes - Read a capabilities object using CDMI

HTTP status	Description
200 OK	The capabilities object content was returned in the response.
400 Bad Request	The request contains invalid parameters or field names.
401 Unauthorized	The authentication credentials are missing or invalid.
403 Forbidden	The client lacks the proper authorization to perform this request.
404 Not Found	The resource was not found at the specified URI.
406 Not Acceptable	The server is unable to provide the object in the content type specified in the Accept header.

12.3.8 Examples

EXAMPLE 1: GET to the root container capabilities URI to read all fields of the container:

```
--> GET /cdmi/2.0.0/cdmi_capabilities/ HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-capability

<-- HTTP/1.1 200 OK
<-- Content-Type: application/cdmi-capability
<--
<-- {
<--   "objectType": "application/cdmi-capability",
<--   "objectID": "00007E7F00104BE66AB53A9572F9F51E",
<--   "objectName": "cdmi_capabilities/",
<--   "parentURI": "/",
<--   "parentID": "00007E7F0010128E42D87EE34F5A6560",
<--   "capabilities": {
<--     "cdmi_domains": "true",
<--     "cdmi_export_nfs": "true",
<--     "cdmi_export_iscsi": "true",
<--     "cdmi_queues": "true",
<--     "cdmi_notification": "true",
<--     "cdmi_query": "true",
<--     "cdmi_metadata_maxsize": "4096",
<--     "cdmi_metadata_maxitems": "1024"
<--   },
<--   "childrenrange": "0-3",
<--   "children": [
<--   "domain/",
<--   "container/",
<--   "dataobject/",
<--   "queue/"
<--   ]
<-- }
```

EXAMPLE 2: GET to the root container capabilities URI to read the capabilities and children of the container:

```
--> GET /cdmi/2.0.0/cdmi_capabilities/?capabilities&children HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-capability

<-- HTTP/1.1 200 OK
<-- Content-Type: application/cdmi-capability
<--
<-- {
<--   "capabilities": {
<--     "cdmi_domains": "true",
<--     "cdmi_export_nfs": "true",
<--     "cdmi_export_iscsi": "true",
<--     "cdmi_queues": "true",
<--     "cdmi_notification": "true",
<--     "cdmi_query": "true",
<--     "cdmi_metadata_maxsize": "4096",
<--     "cdmi_metadata_maxitems": "1024"
<--     },
<--     "children": [
<--       "domain/",
<--       "container/",
<--       "dataobject/",
<--       "queue/"
<--     ]
<-- }
```

EXAMPLE 3: GET to the root container capabilities URI to read the first two children contained within a domain:

```
--> GET /cdmi/2.0.0/cdmi_capabilities/?childrenrange&children=0-1 HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-capability
```

(continues on next page)

(continued from previous page)

```
<-- HTTP/1.1 200 OK
<-- Content-Type: application/cdmi-capability
<--
<-- {
<--   "childrenrange" : "0-1",
<--   "children" : [
<--     "domain/",
<--     "container/"
<--   ]
<-- }
```

Clause 13

Exported protocols

13.1 Overview

Container objects can be exported via multiple storage protocols. This is specified by adding an `exports` field to the container object. The `exports` field contains zero or more named exports, each of which has elements corresponding to the export protocol type, such as:

- The type of export protocol;
- The user-facing identity of the exported container, where required by the export protocol (e.g. iSCSI target, NFS directory);
- The domain of the protocol name server for the clients being served, where required by the export protocol;
- The list of who may mount that container via that protocol, as standardized by that protocol or optionally by leveraging the name mapping protocol (see 13.2.3) and specifying CDMI-resolvable user or groupnames;
- Required protocol-specific export parameters;
- Optional protocol-specific export parameters; and
- Export control parameters.

The ability to export containers via a specific protocol is determined by the presence or absence of a `cdmi_export_<protocol>` system wide capabilities, which are listed in 12.2.7. The ability to export a specific container via a specific protocol is indicated by the `cdmi_export_<protocol>` capability.

Exports are represented as a JSON object having zero or more named protocol-specific exports.

The meaning, use, and permitted values for the fields associated with each export type are described later in this clause.

13.2 Container object export details

13.2.1 Container object export addressing

Container object exports are addressed in CDMI in two ways:

- by name (e.g. `https://cloud.example.com/cdmi/2.0.0/container/?exports`); and
- by ID (e.g. `https://cloud.example.com/cdmi/2.0.0/cdmi_objectid/00007ED900104E1D14771DC67C27BF8B/?exports`).

See 9.1 for more details on container object addressing.

13.2.2 Container object export fields

The export of a container, via data path protocols other than CDMI, is accomplished by creating or updating a container and supplying one or more export protocol structures, one for each such protocol. In this International standard, all such protocols are referred to as foreign protocols.

This International standard defines JSON export structures for several well known foreign protocols. All depend on the following user and groupname mapping feature in the case that multi-protocol access to the container is desired. However, name mapping is not required if an external domain is used, or if CDMI is used only to provision containers to be used exclusively by foreign protocols.

Implementations that support authenticated and authorized access to CDMI objects via both CDMI and foreign protocols need a way to support the setting of security on a per-object basis. The numerous methods of doing this include:

- Defining or adopting a security scheme and mapping all requests into that scheme. CDMI implementations that adopt this scheme shall use a name mapping technique to accomplish it, as (a) this mapping is easier for administrators to manage than straight id-to-id mapping, and (b) it is desired that interoperable CDMI implementations behave similarly in this respect. This means that the name of the principal in an incoming request is mapped to the name of a principal in the security domain, and that principal's id is acquired and used in the authorization procedure.
- Allowing each protocol to set its own security, which implies that an object might be accessible to a given user via one protocol but not another.
- Using the security scheme of the last protocol that was used to set permissions on the object. This method also requires mapping the principal in the incoming request to a principal in the security domain of the object. As in the first case, the server shall use a name mapping procedure to obtain the id that is used to authorize the user against the desired object's ACL.

CDMI does not mandate which method shall be used. It does, however, specify how users and groups shall be mapped between protocols.

13.2.3 Mapping names from CDMI to another protocol

Clients wishing to restrict exports via foreign protocols to mounting only by certain users and groups may be required to provide user and groupname mapping information to the server. This mapping information is also required if access to the container is desired by multiple protocols, e.g., both CDMI and NFS. The mapping is done as follows.

1. When a CDMI container is exported, the server should use the appropriate mechanism, e.g., Powershell `WmiClass.Create( )` on the Windows platform or `/etc/exports` on Unix, to limit permitted mounts of the share from other servers, as specified in the "`root_hosts`", "`rw_hosts`" and "`ro_hosts`" lines of the "`exports`" property. The syntax of each hosts line follows the syntax of `/etc/exports` in the Linux operating system, as encoded in a JSON string. If the CDMI server is unable to limit mounts as specified by each hosts line, an error shall result, but the success or failure of the operation depends on the implementation.
2. When possible, authentication credential resolution should be consistant across both CDMI and all exported protocols.
3. Authentication credential resolution shall be performed in the following order: #. CDMI Domain membership mapping (See 10.4), #. Delegated domain mapping (See 10.4), #. Export name mapping.
4. Implementations may ignore or override export name mapping as requried to enforce implementation-specific security policies.

5. The usermap list for that protocol shall be searched, in order, for an entry matching the username obtained from the authentication credential resolution process (see 13.2.7 for details on the search).
6. The CDMI principal name obtained from the first matching usermap entry during this search is then used to authorize the user request via the security mechanism of the protocol whose security governs access to the object.

Groupname mapping for each foreign protocol shall be specified in a `groupname` field of the foreign protocol export specification. Its syntax is identical to the syntax for the `username` field.

13.2.4 Administrative users

By default, the following users shall be considered "`root`", or administrative users, and equivalent to each other:

- root (Unix/NFS/LDAP),
- Administrator (Windows/AD/SMB), and
- the domain owner (CDMI).

Servers shall automatically map these users to the root user of the target protocol unless otherwise instructed by the usermaps.

As an automatic mapping does not meet strict security standards, servers shall override these built-in entries with any usermap entries that apply to one or more root users.

In the following example, root gets mapped to nobody, and everyone else is mapped to a user of the same name in the NFS domain and the CDMI domain.

EXAMPLE 1: NFS export user mapping

```
--> PUT /cdmi/2.0.0/MyContainer HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/vnd.org.snia.cdmi.container+json
--> Content-Type: application/vnd.org.snia.cdmi.container+json
-->
--> {
-->   "exports": {
-->     "nfs": {
-->       "usermap": [
-->         [
-->           "nobody",
-->           "<-",
-->           "root"
-->         ],
-->         [
-->           "*",
-->           "<-->",
-->           "*"
-->         ]
-->       ]
-->     }
-->   }
--> }
```

13.2.5 Mapping domains from CDMI to another protocol

The internet domain name corresponding to each exported CDMI container shall be described in the "`domain`" element of the protocol export specification as a JSON-formatted string . If the "`domain`" element is not present in the protocol export specification, it shall be assumed the domain is the same as the server hosting the CDMI implementation.

13.2.6 Permissions mapping

Security authorizations and entitlements may not directly correspond across users, groups, file system protocols, operating systems, enterprises or different cloud provider environments. CDMI's primary area of concern is representing a rich set of network files system authorizations and entitlements in a CDMI Access Control List (ACL).

As there are a number of possible ways to coordinate the permissions/ACLs and CDMI ACLs, this International specification does not mandate a particular method. However, all mappings of user and groupnames between domains shall use the name mapping mechanism specified in 13.2.7.

13.2.7 User and groupname mapping syntax and evaluation rules

A BNF-style grammar for name mapping is as follows:

```
name_mapping_list = protocol protocol mapping_list
protocol = "cdmi" | "nfs" | "smb" | "ldap"
mapping_list = name mapping_operator name
name = pattern | utf8_name | quoted_utf8_name
quoted_utf8_name = " utf8_name "
utf8_name = <any legal utf8 character sequence not including the characters ",',\,/,:,*,?>
pattern =  <utf8_name> * | *
mapping_operator = "<--" | "<-->" | "-->"
```

To restate this in English, a mapping entry consists of two names separated by a directional indicator. As most environments use the same usernames and groupnames across administrative domains, the most common mapping is "`* <--> *`", which maps any name to the same name in the foreign protocol domain, and vice versa. It is highly recommended that this be both the default map and the last entry on all more complex maps.

CDMI specifies pattern matching on names in the name map, but only prefix matching is required. The symbol "`*`" at the end of a character string shall match zero or more occurrences of any non-whitespace character.

Evaluation of the name mapping list shall proceed in order; once a match is made, evaluation shall cease and the result of the match shall be returned.

If no matches are found on the match list, the result is system dependent. However, it is recommended that servers either deny access altogether or map the user in question to the equivalent of "`anonymous`" on the destination protocol. It is also recommended that an entry be devoted to the special user "`EVERYONE@`".

13.3 NFS exported protocol

An NFS export specifes the information required by an NFS server to provide an NFS export. Normally, this information is contained in the `/etc/exports` file on a server or the equivalent.

Elements for an NFS export are described in Table 136.

Table 136: Elements of the NFS protocol export structure

Element	Type	Description	Requirement
`type`	JSON String	The export type is set to "`NFS`"	Mandatory
`protocol`	JSON String	The protocol being requested. Values shall be "`NFSv3`", "`NFSv4`", "`NFSv4.1`", or any subsequent NFS version enshrined in an IETF RFC. Version 2 of NFS is not supported by CDMI.	Mandatory
`path`	JSON String	The pathname to which the export should be surfaced. This value shall be a UTF8 string of the form `[<server>:]/<path>`, where the <server> component is optional, (e.g., "`myserver:/lessons/number1`"). If specified, the <server> component of the path must be obtained from an administrator of the service running the CDMI implementation.	Mandatory
`usermap`	JSON Array of JSON Arrays	Authentication credential mapping of user names, as specified in 13.2.3.	Mandatory
`groupmap`	JSON Array of JSON Arrays	Authentication credential mapping of group names, as specified in 13.2.3.	Mandatory
`encryption`	JSON String	This value shall be "`rpcsec_gss`" or future TLS-based transport security.	Optional
`domain_servers`	JSON Array of JSON Strings	A list of server names or IP addresses that function as name servers for the domain given in "`domain`". If given, this list shall override the names obtainable by the CDMI server via other programmatic means.	Optional
`mount_name`	JSON String	The name the client should use to surface the export. This name replaces the last name in the path string, (e.g., mounting "`myserver:/lessons/number1`" with a mountname of "`1`" over the directory `/somepath/lessons/num1` should result in a `/somepath/lessons/1` directory on the client).	Optional
`root_hosts`	JSON Array of JSON Strings	A list of names of hosts that may access the container in superuser mode. The default shall be an empty list.	Optional
`rw_hosts`	JSON Array of JSON Strings	A list of names of hosts that may access the container in `rw` mode. The default shall be an empty list.	Optional
`ro_hosts`	JSON Array of JSON Strings	A list of names of hosts that may access the container in `ro` mode only. The default shall be an empty list.	Optional

Continued on next page

Table 136 – continued from previous page

Element	Type	Description	Requirement
recurse	JSON String	This value shall be either "true" or "false". The default shall be "true". When true, recurse indicates that mounts within the CDMI directory structure (presumably put there by other NFS operations) shall be followed and the mounted directory exposed as though it were part of the CDMI container actually being exported. This parameter is equivalent to the Linux "crossmnt" parameter.	Optional
parameters	JSON String	A string containing NFS server-specific parameters to be passed to the NFS server. The format of this string is implementation specific. The default shall be an empty string.	Optional

Servers shall support wildcard matching on the "*" and "?" characters in the hosts lists, so that "*.cs.uscs.edu" matches all servers in the cs.ucsc.edu department.

Servers may also support IP address ranges in the various lists of hosts. These IP addresses shall be augmented by the same wildcard matching as is used for ordinary host names (e.g., "192.168.1.*" exports to all the machines on local class C network).

Servers shall return an HTTP status code of 400 Bad Request when an export setting does not conform to an allowable setting on the server.

EXAMPLE 2: NFS exports

```
{
    "exports" : {
        "1" : {
            "type" : "nfs",
            "protocol" : "NFSv4",
            "path" : "/myexport",
            "domain_servers" : "lab.example.com",
            "root_hosts" : [ "admin.lab.example.com" ],
            "ro_hosts" : [ "*.lab.example.com" ],
            "usermap" : [
                { "jimsmith", "<-->", "jims" },
                { "*", "<-->", "*" }
            ],
            "groupmap" : [
                { "admins", "<-", "wheel" },
                { "everyone", "<-", "*" }
            ]
        }
}
```

13.4 SMB exported protocol

An SMB export specifes the information required by an SMB server to provide an SMB export.

Elements for an SMB export are described in Table 137

Table 137: Elements of the SMB protocol export structure

Element	Type	Description	Requirement
`type`	JSON String	The export type is set to "`SMB`"	Mandatory
`sharename`	JSON String	The name that SMB shall use to discover the share.	Mandatory
`usermap`	JSON Array of JSON Arrays	Authentication credential mapping of user names, as specified in 13.2.3.	Mandatory
`groupmap`	JSON Array of JSON Arrays	Authentication credential mapping of group names, as specified in 13.2.3.	Mandatory
`root_hosts`	JSON Array of JSON Strings	A list of names of hosts that may access the container in superuser mode. The default shall be an empty list.	Optional
`rw_hosts`	JSON Array of JSON Strings	A list of names of hosts that may access the container in `rw` mode. The default shall be an empty list.	Optional
`ro_hosts`	JSON Array of JSON Strings	A list of names of hosts that may access the container in `ro` mode only. The default shall be an empty list.	Optional
`domain_servers`	JSON Array of JSON Strings	A list of server names or IP addresses that function as name servers for the domain given in "`domain`". If given, this list shall override the names obtainable by the CDMI server via other programmatic means.	Optional
`comment`	JSON String	This value shall be JSON String containing a user-friendly share name for the client.	Optional
`parameters`	JSON String	A string containing SMB server-specific parameters to be passed to the SMB server. The format of this string is implementation specific. The default shall be an empty string.	Optional

Servers shall return an HTTP status code of `400 Bad Request` when an export setting does not conform to an allowable setting on the server.

EXAMPLE 3: SMB exports

```
{
    "exports" : {
        "1" : {
            "type" : "smb",
            "rw_hosts" : [ "*" ],
            "domain_servers" : "lab.mycollege.edu",
            "usermap" : [
                { "jimsmith", "<-->", "james.smith" },
                { "*", "<-->", "*" }
            ],
            "groupmap" : [
                { "admins", "<-", "Administrators" },
                { "everyone", "<-", "*" }
            ]
        }
```

(continues on next page)

(continued from previous page)

```
        }
}
```

13.5 iSCSI exported protocol

An iSCSI export specifes the information required by an iSCSI server (see RFC 7143 [4]) to provide an iSCSI export. Each container is exported as a single SCSI Logical Unit as a Logical Unit Number (LUN). One or more iSCSI initiators import the LUN through an iSCSI target node and port using one or more iSCSI network portals (IP addresses).

Elements for an iSCSI export are described in Table 138

Table 138: Elements of the iSCSI protocol export structure

Element	Type	Description	Requirement
`type`	JSON String	The export type is set to "`iSCSI`"	Mandatory
`permissions`	JSON Array of JSON Strings	One or more target identifiers for initiators that are permitted to access the iSCSI export. Target identifiers may be in `iqn`, `naa`, or `eui` format and shall have the target portal group tag appended in hexadecimal. If absent, any initiator may access the export.	Optional
`parameters`	JSON String	A string containing iSCSI server-specific parameters to be passed to the iSCSI server. The format of this string is implementation specific. The default shall be an empty string.	Optional
`target_identifier`	JSON String	iSCSI target information (IP addresses or fully qualified domain names, target identifier, and LUN)	Read-Only
`logical_unit_number`	JSON String	iSCSI Logical Unit Number	Read-Only
`logical_unit_name`	JSON String	iSCSI Logical Unit Name	Read-Only
`portals`	JSON Array of JSON Strings	One or more IP addresses or fully qualified domains names through which the iSCSI export may be accessed. This field is server populated.	Read-Only

Servers shall return an HTTP status code of `400 Bad Request` when an export setting does not conform to an allowable setting on the server.

EXAMPLE 4: iSCSI export creation

```
"exports" :
{
    "1" : {
        type: "iSCSI",
        "permissions": [
            "iqn.2010-01.com.acme:host1",
            "iqn.2010-01.com.acme:host2"
        ]
    }
}
```

EXAMPLE 5: Reading iSCSI export information after creation

```
"exports" :
{
    "1" : {
        type: "iSCSI",
        "portals": [
            "192.168.1.101",
            "192.168.1.102"
        ],
        "target_identifier": "iqn.2010-01.com.cloudprovider:acmeroot.container1,t,0x0001",
        "logical_unit_number": "3",
        "logical_unit_name": "0x60012340000000000000000000000001",
        "permissions": [
            "iqn.2010-01.com.acme:host1",
            "iqn.2010-01.com.acme:host2"
        ]
```

(continues on next page)

(continued from previous page)

```
        }
}
```

13.6 WebDAV exported protocol

A WebDAV export specifes the information required by an WebDAV server (see RFC 4918 [6]) to provide an WebDAV export.

Elements for an WebDAV export are described in Table 139

Table 139: Elements of the WebDAV protocol export structure

Element	Type	Description	Requirement
`type`	JSON String	The export type is set to "`WebDAV`"	Mandatory
`usermap`	JSON Array of JSON Arrays	Authentication credential mapping of user names, as specified in 13.2.3.	Mandatory
`groupmap`	JSON Array of JSON Arrays	Authentication credential mapping of group names, as specified in 13.2.3.	Mandatory
`parameters`	JSON String	A string containing WebDAV server-specific parameters to be passed to the WebDAV server. The format of this string is implementation specific. The default shall be an empty string.	Optional

Servers shall return an HTTP status code of `400 Bad Request` when an export setting does not conform to an allowable setting on the server.

WebDAV supports locking, but it is up to implementations to support any locking of access through CDMI as a result, and the interaction between the two protocols is purposely not described in this International Standard.

EXAMPLE 6: WebDAV export

```
"exports" :
{
    "1" : {
        type: "WebDAV",
        "usermap" : [
            { "*", "<-->", "*" }
        ],
        "groupmap" : [
            { "*", "<-->", "*" }
        ]
    }
}
```

13.7 OCCI exported protocol

Container objects can be exported via multiple protocols. This is especially useful when CDMI is being used as a storage interface in a cloud computing environement, as illustrated in Fig. 9 below.

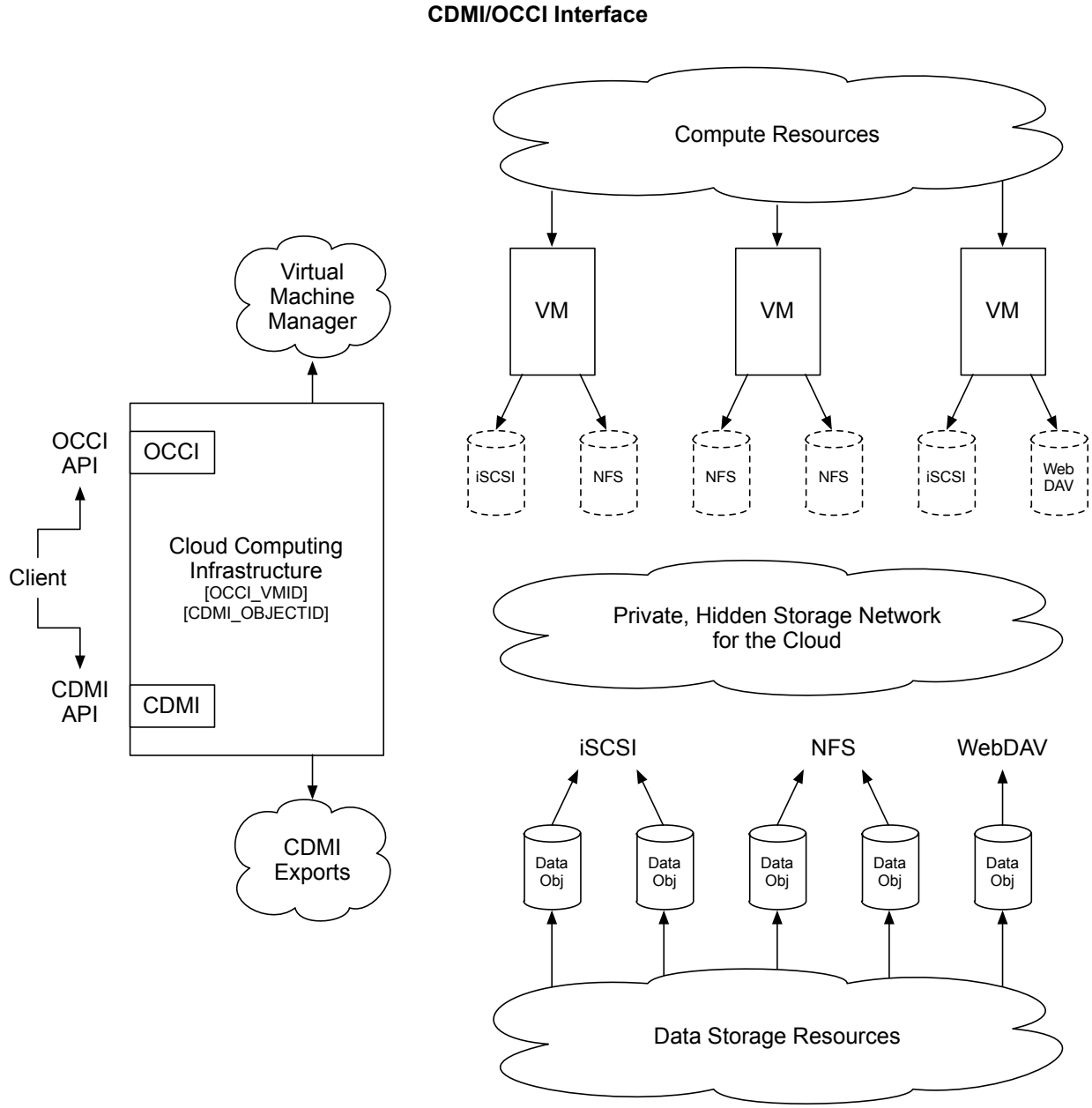


Fig. 9: CDMI and OCCI in an integrated cloud computing environment

In this example, CDMI containers may also be used as virtual disks by virtual machines in the cloud computing environement. The cloud computing infrastructure management is shown as implementing both an Open Cloud Computer Interface (OCCI) and CDMI interfaces. With the internal knowledge of the network and the virtual machine manager's mapping of drives, this infrastructure may associate the CDMI containers to the guests using the appropriate exported protocol.

To support exported protocols and improve their interoperability with CDMI, CDMI provides a type of exported protocol that contains information obtained via the OCCI interface. In addition, OCCI provides a type of storage that corresponds

to a CDMI container that is exported with a specific type of protocol used by OCCI. A client of both interfaces performs operations that align the architectures, including the following:

- The client creates a CDMI container through the CDMI interface and exports it as an OCCI export protocol type. The CDMI container object ID is returned as a result.
- The client creates a virtual machine through the OCCI interface and attaches a storage volume of type CDMI using the object ID and protocol type. The OCCI virtual machine ID is returned as a result.
- The client updates the export protocol structure of the CDMI container object with the OCCI virtual machine ID to allow the virtual machine access to the container.
- The client starts the virtual machine through the OCCI interface.

CDMI defines an export protocol structure for the Open Cloud Computing Interface (13.7) as follows:

- The type is "`OCCI/<protocol standard>`" (e.g., "`OCCI/NFSv4`").
- The identifier is the CDMI container ID.
- A JSON array of URIs to OCCI compute resources shall have access (permissions) to the exported container.

EXAMPLE 5: OCCI export

```
"OCCI/iSCSI":
    {
    "identifier": "00007E7F00104BE66AB53A9572F9F51E",
    "permissions":
        [
            "https://example.com/compute/0/",
            "https://example.com/compute/1/"
        ]
    }
```

For more detail on using the OCCI export protocol structure attributes, see 13.1. Because the actual networking and access control is under the control of a hidden, common infrastructure that implements both OCCI and CDMI, the normal permission structure shall not be provided.

Clause 14

CDMI snapshots

14.1 Overview

A snapshot is a point-in-time copy (image) of a container and all of its contents, including subcontainers and all data objects and queue objects. The client names a snapshot of a container at the time the snapshot is requested. A snapshot operation creates a new container to contain the point-in-time image. The first processing of a snapshot operation also adds a `cdmi_snapshots` child container to the source container. Each new snapshot container is added as a child of the `cdmi_snapshots` container. The snapshot does not include the `cdmi_snapshots` child container or its contents (see Fig. 10).

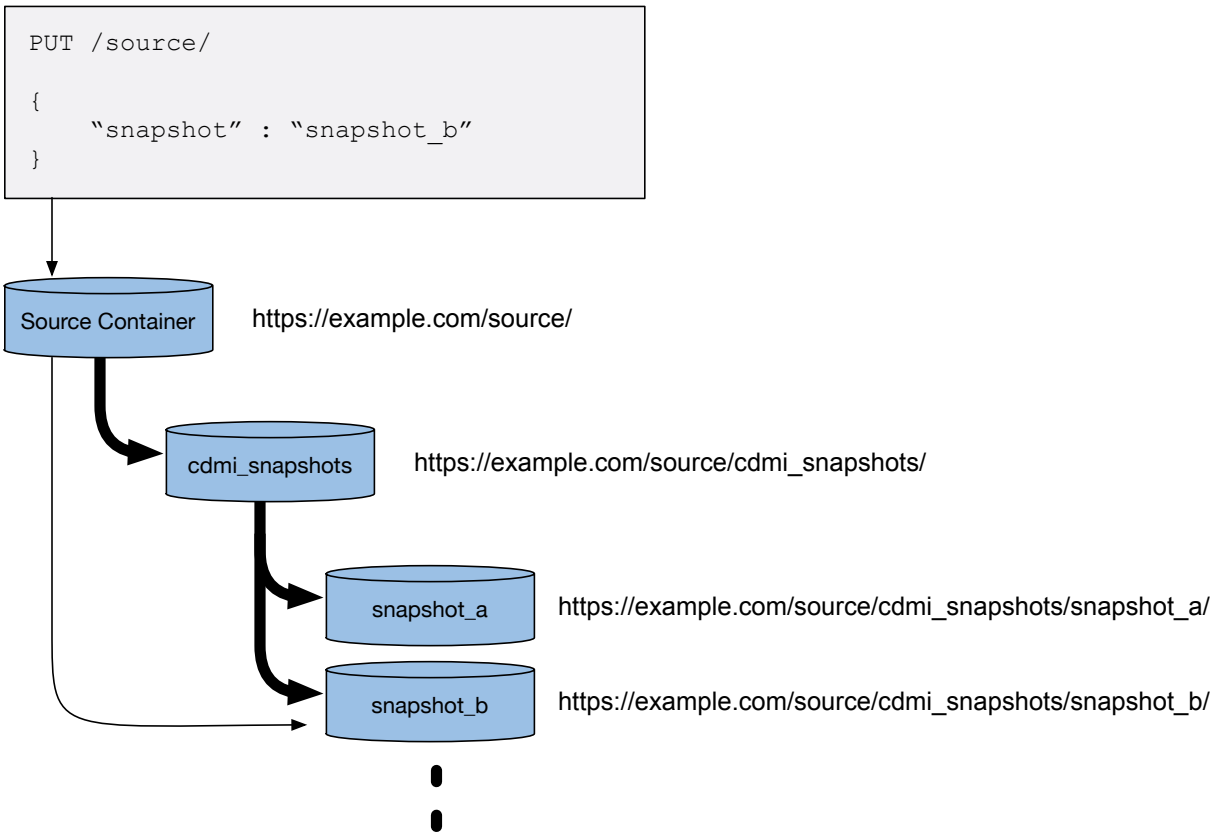


Fig. 10: Snapshot container structure

14.2 Creating a snapshot

14.2.1 Operation context

A snapshot operation is requested using the container update operation (see 9.5), in which the snapshot field specifies the requested name of the snapshot.

A snapshot may be accessed in the same way that any other CDMI™ object is accessed. An important use of a snapshot is to allow the contents of the source container to be restored to their values at a previous point in time using a CDMI copy operation.

14.2.2 Example

EXAMPLE 1: PATCH to an existing container to create a snapshot:

```
--> PATCH /cdmi/2.0.0/MyContainer/ HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-container
-->
--> {
-->   "snapshot" : "MySnapshot"
--> }

<-- HTTP/1.1 201 Created
```

14.3 Deleting a snapshot

14.3.1 Operation context

A snapshot can be deleted by performing a CDMI container delete operation on the corresponding child container in the `cdmi_snapshots` container, or by performing a CDMI container delete operation on the snapshot Object ID.

14.3.2 Example

EXAMPLE 1: DELETE to an existing snapshot:

```
--> DELETE /cdmi/2.0.0/MyContainer/cdmi_snapshots/MySnapshot HTTP/1.1
--> Host: cloud.example.com

<-- HTTP/1.1 204 No Content
```

Clause 15

Serialization/deserialization

15.1 Overview

Bulk data movement is often needed between, into, or out of clouds. When moving bulk data, cloud serialization operations provide a means to normalize data to a canonical, self-describing format, which includes:

- data migration between clouds,
- data migration during upgrades (or replacements) of cloud implementations, and
- robust backup.

The canonical format of serialized data describes how the data is to be represented in a byte stream. As long as this byte stream is not changed during the transfer from source to destination, the data may be reconstituted on the destination system.

15.2 Canonical format

15.2.1 General requirements

Support for CDMI serialization using JSON as the canonical format requires the presence of the `cdmi_serialization_json` capability.

The canonical format shall represent specified data objects and container objects as they exist within the storage system. Each object shall be represented by the metadata for the object, identifiers, and the data stream contents of the data object. Because data and storage system metadata is inherited from enclosing container objects, all parent metadata shall be represented in the top-level of the canonical format. To preserve the actual metadata values that apply to the data object that is being serialized, the non-overridden metadata is included from both the immediate parent container object of the specified object and from the parent of each higher-level container object.

The canonical format shall have the following characteristics:

- recursive JSON for the data object, consistent with the rest of CDMI;
- user and data system metadata for each data object/container object;
- data stream contents for each data object and queue object;
- binary data represented using escaped JSON strings; and
- typing of data values consistent with CDMI JSON representations.

15.2.2 Example JSON canonical serialized format

EXAMPLE 1: In this example, a data object and a queue object in a container object have been selected for serialization:

```
{
    "objectType": "application/cdmi-container",
    "objectID": "00007E7F00102E230ED82694DAA975D2",
    "objectName": "MyContainer/",
    "parentURI": "/",
    "parentID": "00007E7F0010128E42D87EE34F5A6560",
    "domainURI": "/cdmi_domains/MyDomain/",
    "capabilitiesURI": "/cdmi_capabilities/container/",
    "completionStatus": "Complete",
    "metadata": {
                ...
        },
    "exports": {
        "OCCI/iSCSI": {
            "identifier": "00007E7F00104BE66AB53A9572F9F51E",
            "permissions": [
                "https://example.com/compute/0/",
                "https://example.com/compute/1/"
            ]
        },
        "Network/NFSv4": {
            "identifier": "/users",
            "permissions": "domain"
        }
    },
    "childrenrange": "0-1",
    "children": [
        {
            "objectType": "application/cdmi-object",
            "objectID": "00007ED900104F67307652BAC9A37C93",
            "objectName": "MyDataObject.txt",
            "parentURI": "/MyContainer/",
            "parentID": "00007E7F00102E230ED82694DAA975D2",
            "domainURI": "/cdmi_domains/MyDomain/",
            "capabilitiesURI": "/cdmi_capabilities/dataobject/",
            "completionStatus": "Complete",
            "mimetype": "text/plain",
            "metadata": {
```

(continues on next page)

(continued from previous page)

```
                                ...
                        },
                "valuerange": "0-36",
                "valuetransferencoding": "utf-8",
                "value": "This is the Value of this Data Object"
            },
            {
                "objectType": "application/cdmi-queue",
                "objectID": "00007E7F00104BE66AB53A9572F9F51E",
                "objectName": "MyQueue",
                "parentURI": "/MyContainer/",
                "parentID": "00007E7F00102E230ED82694DAA975D2",
                "domainURI": "/cdmi_domains/MyDomain/",
                "capabilitiesURI": "/cdmi_capabilities/queue/",
                "completionStatus": "Complete",
                "metadata": {
                                ...
                        },
                "queueValues": "0-1",
                "mimetype": [
                    "text/plain",
                    "text/plain"
                ],
                "valuetransferencoding": [
                    "utf-8",
                    "utf-8"
                ],
                "valuerange": [
                    "0-2",
                    "0-3"
                ],
                "value": [
                    "red",
                    "blue"
                ]
            }
        ]
}
```

To allow efficient deserialization in stream mode when serializing container objects to JSON, data object `value` fields and container `children` arrays should be the last items in the canonical serialized JSON format.

15.3 Exporting serialized data

A canonical encoding of the data is obtained by creating a new data object and specifying that the source for the creation is to serialize a given CDMI™ data object, container object, or queue object. On a successful serialization, the result shall be a data object that is created with the serialized data as its value. If a container object has an exported block protocol, the serialized data may contain the block-by-block contents of that container object along with its metadata.

The resulting data object that is produced is the canonical representation of the selected data object, container object and children, or queue object.

- If the source specified is a data object, the canonical format shall contain all data object fields, including the `value`, `valuetransferencoding`, and `metadata` fields.
- If the source being specified is a queue object, the canonical format shall contain all queue object fields, including the `value` and `valuetransferencoding` fields of enqueued items, along with the metadata of the queue object itself.
- If the source being specified is a container object, the canonical format shall contain all container object fields, recursively, including all children of the container object. If a user attempts to serialize a container object that includes children that the user, who is performing the serialization operation, does not have permission to read, these objects shall not be included in the resulting serialized object.

When performing a serialization operation, objects shall only be included if the principal initiating the serialization has sufficient permissions to read those objects.

15.4 Importing serialized data

Canonical data may be deserialized back into the cloud by creating a new data object, container object, or queue object and by specifying that the source for the creation is to deserialize a given CDMI data object or by specifying the serialized data in base64 encoding in the deserializevalue field.

The destination may or may not exist previously. If not, a create operation is performed. If a container object already exists, an update operation with serialized children shall update the container object and all children. If the serialized container object does not contain children, only the container object is updated. Data objects are recreated as specified in the canonical format, including all metadata and the data object ID.

Table 140: Serialization import behaviour

User has `cross_domain`	User specifies `domainURI`	Description
No	No	The `domainURI` of the parent object shall match the `domainURI` in each seralized object being deserialized. If the `domainURI` in any serialize object does not match the `domainURI` of the parent object, the entire deserialize operation shall fail, and an HTTP status code of `400 Bad Request` shall be returned.
No	Yes	The specified `domainURI` shall be used, overriding the original `domainURI` in each serialized object being deserialized. If a `domainURI` other than the `domainURI` of the parent is specified, the entire deserialize operation shall fail, and an HTTP status code of `400 Bad Request` shall be returned.
Yes	No	The original `domainURI` in each serialized object being deserialized shall be used. If any of the original `domainURI` in each serialized object being deserialized is not valid in the context of the storage system on which the deserialization operation is being performed, the entire deserialize operation shall fail, and an HTTP status code of `400 Bad Request` shall be returned.
Yes	Yes	The specified `domainURI` shall be used, overriding the original `domainURI` in each serialized object being deserialized. If a `domainURI` that is specified is not valid in the context of the storage system on which the deserialization operation is being performed, the entire deserialize operation shall fail, and an HTTP status code of `400 Bad Request` shall be returned.

Deserialization operations shall restore all metadata from the specified source. If the original provider of the serialized data-supported vendor extensions is through custom metadata keys and values, then these customized requirements shall be restored when deserialized. However, the custom metadata keys and values may be treated as user metadata (preserved, but not interpreted) by the destination provider. Preservation allows custom data requirements to move between clouds without losing this information.

Clause 16

Metadata

16.1 Overview

CDMI metadata allows for additional information to be associated with stored objects. JSON objects, strings and arrays are used to transfer metadata in CDMI operations, which allows for metadata to be hierachical. CDMI servers may place a restriction on the number of metadata items, maximum size per metadata item, and total size of metadata items, as specified in the `cdmi_metadata_maxitems`, `cdmi_metadata_maxsize`, and `cdmi_metadata_maxtotalsize` capabilities. CDMI servers shall not place a restriction on the depth of the metadata hierarchy and number of array items, outside of the above restrictions.

When objects are created, object metadata is created according to the following process:

1. Metadata items specified in the create operation are added, overriding pre-existing metadata items
2. Storage System metadata items are added to the object, overriding pre-existing metadata items subject to the restrictions described in Section 16.2

When objects are updated, object metadata is updated according to the following process:

1. Existing metadata items are deleted, changed and/or added, as specified in the update operation
2. Storage System metadata items are updated for the object, overriding pre-existing metadata items subject to the restrictions described in Section 16.2

When objects are read, object metadata is returned according to the following process:

1. Data System Metadata items is inherited from the parent container
2. Metadata items stored with the object are returned, overriding any inherited Data System Metadata items

16.2 Support for storage system metadata

After an object has been created or updated, the storage system metadata, as described in Table 141, shall be generated or updated by the cloud storage system, and shall immediately be made available to a CDMI client in the metadata that is returned as a result of the create operation and any subsequent retrievals.

Which storage system metadata is supported by the CDMI server defined in 12.2.8. Storage system metadata that is not supported by the CDMI server shall be preserved.

Table 141: Storage system metadata

Metadata name	Type	Description	Requirement
`cdmi_size`	JSON string	The number of bytes consumed by the object. This storage system metadata item is computed by the storage system, and any attempts to set or modify it will be ignored.	Optional
`cdmi_ctime`	JSON string	The time when the object was created, in ISO-8601 point-in-time format, as described in 5.6. For a newly created object, this value shall be set to the creation time. This metadata value may only be updated by a client if it has the "`backup_operator`" privilege. If a client does not have the backup operator privilege, updates of this metadata item shall be ignored.	Optional
`cdmi_atime`	JSON string	The time when the object was last accessed in ISO-8601 point-in-time format, as described in 5.6. The access or modification of a child is not considered an access of a parent container (access/modify times do not propagate up the tree). For a newly created object, this value shall be set to the creation time. This metadata value may only be updated by a client if it has the "`backup_operator`" privilege. If a client does not have the backup operator privilege, updates of this metadata item shall be ignored.	Optional
`cdmi_mtime`	JSON string	The time when the object was last modified, in ISO-8601 point-in-time format, as described in 5.6. The modification of a child is not considered a modification of a container object (modification times do not propagate up the tree). For a newly created object, this value shall be set to the creation time. This metadata value may only be updated by a client if it has the "`backup_operator`" privilege. If a client does not have the backup operator privilege, updates of this metadata item shall be ignored.	Optional
`cdmi_acount`	JSON string	The number of times that the object has been accessed since it was originally created. Accesses include all reads, writes, and lists. For a newly created object, this value shall be set to the value "`0`". This metadata value may only be updated by a client if it has the "`backup_operator`" privilege. If a client does not have the backup operator privilege, updates of this metadata item shall be ignored.	Optional

Continued on next page

Table 141 – continued from previous page

Metadata name	Type	Description	Requirement
cdmi_mcount	JSON string	The number of times that the object has been modified since it was originally created. Modifications include all value and metadata changes. Modifications to metadata resulting from reads (such as updates to atime) do not count as a modification. For a newly created object, this value shall be set to the value "0". This metadata value may only be updated by a client if it has the "backup_operator" privilege. If a client does not have the backup operator privilege, updates of this metadata item shall be ignored.	Optional
cdmi_hash	JSON string	The hash of the value of the object, encoded using Base16 encoding rules described in RFC 4648 [19]. This metadata field shall be present when the "cdmi_value_hash" data system metadata for the object or a parent object indicates that the value of the object should be hashed. This storage system metadata item is computed by the storage system, and any attempts to set or modify it will be ignored.	Optional
cdmi_owner	JSON string	The name of the principal that has owner privileges for the object. If not specified when the object is created, this principal associated with the user creating the object shall be used. This metadata value can be updated by users with appropriate permissions.	Optional
cdmi_acl	JSON array of JSON objects	Standard ACL metadata as described in 17.1. If not specified when the object is created, the ACL metadata shall be generated in by the system. This metadata value can be updated by users with appropriate permissions.	Optional
cdmi_dac_uri	JSON string	Contains the URI used to submit a DAC request for the data object. URI schemes supported is defined in the cdmi_dac_methods capability. Both cdmi_dac_certificate and cdmi_dac_uri shall be included for delegated access control to be enabled for a given object.	Optional
cdmi_dac_certificate	JSON object	A JSON object, containing a JWE JWK which shall include a public key that is used to submit a DAC request for the data object, and should contains a X.509 certificate or certificate chain used to verify the identity of the DAC provider. Both cdmi_dac_certificate and cdmi_dac_uri shall be included for delegated access control to be enabled for a given object.	Optional
cdmi_enc_signature	JSON object	Contains JWS compact serialization of a signature for the entire object (value and metadata). See clause 23.7 for more details.	Optional

16.3 Support for data system metadata

When specified, data system metadata, as described in Table 142, provides guidelines to the cloud storage system on how to provide storage data services for data managed through the CDMI interface.

Data system metadata is inherited from parent objects to any children objects. If a child object explicitly contains data system metadata, the metadata value of the child object data system metadata shall override any corresponding inherited metadata value of the parent object data system metadata.

Which data system metadata is supported by the CDMI server defined in 12.2.9. Data system metadata that is not supported by a CDMI server shall be preserved.

Table 142: Data system metadata

Metadata name	Type	Description	Requirement
`cdmi_data_redundancy`	JSON string	If this data system metadata item is present and set to a positive numeric string, it indicates that the client is requesting a desired number of complete copies. Additional copies may be made to satisfy demand for the value. When this data system metadata item is absent, or is present and is not set to a positive numeric string, this data system metadata item shall not be used.	Optional
`cdmi_immediate_redundancy`	JSON string	If this data system metadata item is present and set to "`true`", it indicates that the client is requesting that at least the number of copies indicated in `cdmi_data_redundancy` contain the newly written value before the operation completes. This metadata is used to make sure that multiple copies of the data are written to permanent storage to prevent possible data loss. When this data system metadata item is absent, or is present and is not set to "`true`", this data system metadata item shall not be used. If the requested number of copies cannot be created within the HTTP timeout period, the transaction shall complete, but the `cdmi_immediate_redundancy_provided` data system metadata shall be set to "`false`".	Optional
`cdmi_assignedsize`	JSON string	If this data system metadata item is present and set to a positive numeric string, it indicates that the client is specifying the size in bytes that is desired to be reported for a container object exported via other protocols (see 9.2.3). The system is not required to reserve this space and may thin-provision the requested space. Thus, the requested value may be greater than the actual storage space consumed. When this data system metadata item is absent, or is present and is not set to a positive numeric string, this data system metadata item shall not be used. This data system metadata item is only applied against container objects and is not inherited by child objects.	Optional

Continued on next page

Table 142 – continued from previous page

Metadata name	Type	Description	Requirement
cdmi_infrastructure_redundancy	JSON string	If this data system metadata item is present and set to a positive numeric string, it indicates that the client is requesting a desired number of independent storage infrastructures supporting the multiple copies of data. This metadata is used to convey that, of the copies specified in cdmi_data_redundancy, these copies shall be stored on this many separate infrastructures. When this data system metadata item is absent, or is present and is not set to a positive numeric string, this data system metadata item shall not be used.	Optional
cdmi_data_dispersion	JSON string	If this data system metadata item is present and set to a positive numeric string, it indicates that the client is requesting a minimum desired distance (in km) between the infrastructures supporting the multiple copies of data. This metadata is used to separate the (cdmi_infrastructure_redundancy number of) infrastructures by a minimum geographic distance to prevent data loss due to site disasters. When this data system metadata item is absent, or is present and is not set to a positive numeric string, this data system metadata item shall not be used.	Optional
cdmi_geographic_placement	JSON array of JSON strings	If this data system metadata item is present and set to zero or more geopolitical identifiers, it indicates that the client is requesting restrictions on the geographic regions where the object is permitted to be stored. Each geopolitical identifier shall be in the form of either a string containing a valid ISO 3166 country/country-subdivision code, which indicates that storage is permitted within that geopolitical region, or in the form of a string starting with the "!" character in front of a valid ISO 3166 country/country-subdivision code, which excludes that country/country-subdivision from the previous list of geopolitical regions. The list is evaluated, in order, from left to right, with evaluation of each candidate storage location stopping when the candidate location is a permitted or prohibited region or is contained within a permitted or prohibited region. In addition to the ISO 3166 codes, "*" shall indicate all regions. If a candidate location does not match any of the entries in the list, the candidate location shall be considered to be prohibited. • When this data system metadata item is absent, this data system metadata item shall not be used. • When this data system metadata item is present and does not contain valid geopolitical identifiers, the create, update, or deserialize operation shall fail with an HTTP status code of 400 Bad Request. • When this data system metadata item is present and valid, but no available storage locations are permitted, the create, update, or deserialize operation shall fail with an HTTP status code of 403 Forbidden.	Optional

Continued on next page

Table 142 – continued from previous page

Metadata name	Type	Description	Requirement
`cdmi_retention_id`	JSON string	If this data system metadata item is present and not an empty string, it indicates that the client is requesting that the string be used to tag a given object as being managed by a specific retention policy. This data system metadata item is not required to place an object under retention, but is useful when needing to be able to perform a query to find all objects under a specific retention policy. When this data system metadata item is absent, or is present and an empty string, this data system metadata item shall not be used.	Optional
`cdmi_retention_period`	JSON string	If this data system metadata item is present and contains a valid ISO 8601:2004 time interval (as described in), it indicates that the client is requesting that an object be placed under retention (see 18.3). When this data system metadata item is absent, this data system metadata item shall not be used. When this data system metadata item is present but does not contain a valid ISO 8601:2004 time interval, the create, update, or deserialize operation shall fail with an HTTP status code of `400 Bad Request`. If this data system metadata item is updated and the new end date is before the current end date, the update operation shall fail with an HTTP status code of `403 Forbidden`.	Optional
`cdmi_retention_autodelete`	JSON string	If this data system metadata item is present and set to "`true`", it indicates that the client is requesting that an object under retention be automatically deleted when retention expires. When this data system metadata item is absent, or is present and is not set to "`true`", this data system metadata item shall not be used.	Optional
`cdmi_hold_id`	JSON array of JSON strings	If this data system metadata item is present and not an empty array, it indicates that the client is requesting that an object be placed under hold (see 18.4). Each string in the array shall contain a unique user-specified hold identifier. When this data system metadata item is absent, or is present and is an empty JSON array, this data system metadata item shall not be used. If this data system metadata item is updated, and a previously existing hold string has been removed or changed in the update, the update operation shall fail with an HTTP status code of `403 Forbidden`. (See 18.4 concerning releasing holds.)	Optional

Continued on next page

Table 142 – continued from previous page

Metadata name	Type	Description	Requirement
cdmi_encryption	JSON string	If this data system metadata item is present and not an empty string, it indicates that the client is requesting that the object be encrypted while at rest. If encrypted, all data and metadata related to the object shall be encrypted. Supported algorithm/mode/length values are provided by the cdmi_encryption capability. When this data system metadata item is absent, this data system metadata item shall not be used. If this data system metadata item is present but does not contain a valid encryption algorithm/mode/length string, the system is free to choose to ignore the data system metadata, to fail with an HTTP status code of 400 Bad Request, or to select an encryption algorithm/mode/length of the system's choice. Supported encryption algorithms are expressed as a string in the form of ALGORITHM_MODE_KEYLENGTH, where: • "ALGORITHM" is the encryption algorithm (e.g., "AES" or "3DES"). • "MODE" is the mode of operation (e.g., "XTS", "CBC", or "CTR"). • "KEYLENGTH" is the key size in bytes (e.g., "128", "192", "256"). To improve interoperability between CDMI implementations, the following designators should be used for the more common encryption combinations: • "3DES_ECB_168" for the three-key TripleDES algorithm, the Electronic Code Book (ECB) mode of operation, and a key size of 168 bits; • "3DES_CBC_168" for the three-key TripleDES algorithm, the Cipher Block Chaining (CBC) mode of operation, and a key size of 168 bits; • "AES_CBC_128" for the AES algorithm, the CBC mode of operation, and a key size of 128 bits; • "AES_CBC_256" for the AES algorithm, the CBC mode of operation, and a key size of 256 bits; • "AES_XTS_128" for the AES algorithm, the XTS mode of operation, and a key size of 128 bits; and • "AES_XTS_256" for the AES algorithm, the XTS mode of operation, and a key size of 256 bits.	Optional

Continued on next page

Table 142 – continued from previous page

Metadata name	Type	Description	Requirement
cdmi_value_hash	JSON string	If this data system metadata item is present and not an empty string, it indicates that the client is requesting that the system hash the object value using the hashing algorithm and length requested. The result of the hash shall be provided in the cdmi_hash storage system metadata item. Supported algorithm/length values are provided by the cdmi_value_hash storage system capability. When this data system metadata item is absent, this data system metadata item shall not be used. If this data system metadata item is present but does not contain a valid hash algorithm/length string, the system is free to choose to ignore the data system metadata, to fail with an HTTP status code of 400 Bad Request, or to select a hash algorithm/length of the system's choice. Supported hash algorithms are expressed as a string in the form of ALGORITHM LENGTH, where: • "ALGORITHM" is the hash algorithm (e.g., "SHA"). • "LENGTH" is the hash size in bytes (e.g., "160", "256"). To improve interoperability between CDMI implementations, the following designators should be used for the more common encryption combinations: • "SHA160" for SHA-1, and • "SHA256" for SHA-2.	Optional
cdmi_latency	JSON string	If this data system metadata item is present and set to a positive numeric string, it indicates that the client is requesting a desired maximum time to first byte, in milliseconds. This metadata is the desired latency (in milliseconds) to the first byte of data, as measured from the edge of the cloud and factoring out any propagation latency between the client and the cloud. For example, this metadata may be used to determine, in an interoperable way, from what type of storage medium the data may be served. When this data system metadata item is absent, or is present and is not set to a positive numeric string, this data system metadata item shall not be used.	Optional
cdmi_throughput	JSON string	If this data system metadata item is present and set to a positive numeric string, it indicates that the client is requesting a desired maximum data rate on retrieve, in bytes per second. This metadata is the desired bandwidth to the data, as measured from the edge of the cloud and factoring out any bandwidth capability between the client and the cloud. This metadata is used to stage the data in locations where there is sufficient bandwidth to accommodate a maximum usage. When this data system metadata item is absent, or is present and is not set to a positive numeric string, this data system metadata item shall not be used.	Optional

Continued on next page

Table 142 – continued from previous page

Metadata name	Type	Description	Requirement
cdmi_sanitization_method	JSON string	If this data system metadata item is present and not an empty string, it indicates that the client is requesting that the system use a specific sanitization method to delete data such that the data is unrecoverable after an update or delete operation. Supported sanitization method values are provided by the cdmi_sanitization_method capability. When this data system metadata item is absent, this data system metadata item shall not be used. If this data system metadata item is present but does not contain a valid sanitization method string, the system is free to choose to ignore the data system metadata, to fail with an HTTP status code of 400 Bad Request, or to select a sanitization method of the system's choice. Supported sanitization methods are defined as system-specific strings.	Optional
cdmi_RPO	JSON string	If this data system metadata item is present and set to a positive numeric string, it indicates that the client is requesting a largest acceptable duration in time between an update or create and when the object may be recovered, specified in seconds. This metadata is used to indicate the desired backup frequency from the primary copy or copies of the data to the secondary copy or copies. It is the maximum acceptable time period before a failure or disaster during which changes to data may be lost as a consequence of recovery. When this data system metadata item is absent, or is present and is not set to a positive numeric string, this data system metadata item shall not be used.	Optional
cdmi_RTO	JSON string	If this data system metadata item is present and set to a positive numeric string, it indicates that the client is requesting the largest acceptable duration in time to restore data, specified in seconds. This metadata is used to indicate the desired maximum acceptable duration to restore the primary copy or copies of the data from a secondary backup copy or copies. When this data system metadata item is absent, or is present and is not set to a positive numeric string, this data system metadata item shall not be used.	Optional
cdmi_enc_key_id	JSON string	If this data system metadata item is present and not an empty string, it indicates that the client is requesting that the system associate with the object a key identifier (e.g. KMIP Identifier) for the symmetric key used to encrypt and decrypt the object.	Optional
cdmi_enc_value_sign_id	JSON string	If this data system metadata item is present and not an empty string, it indicates that the client is requesting that the system associate with the object a key identifier (e.g. KMIP Identifier) for the private key used for signing the value of the object.	Optional
cdmi_enc_value_verify_id	JSON string	If this data system metadata item is present and not an empty string, it indicates that the client is requesting that the system associate with the object a key identifier (e.g. KMIP Identifier) for the public key or certificate chain used for verifying the signature of the value of the object.	Optional

Continued on next page

Table 142 – continued from previous page

Metadata name	Type	Description	Requirement
cdmi_enc_object_sign_id	JSON string	If this data system metadata item is present and not an empty string, it indicates that the client is requesting that the system associate with the object a key identifier (e.g. KMIP Identifier) for the private key used for signing the entire object.	Optional
cdmi_enc_object_verify_id	JSON string	If this data system metadata item is present and not an empty string, it indicates that the client is requesting that the system associate with the object a key identifier (.e.g. KMIP Identifier) for the public key or certificate chain used for verifying the signature of the entire object.	Optional
cdmi_versioning	JSON string	If this data system metadata item is present and not an empty string, it indicates that the client is requesting that versioning be enabled for the data object, and what level of versioning is requrested. • If set to the value "value", versions shall be created when the value is updated. • If set to the value "user", versions shall be created when the value and/or user metadata is updated. • If set to the value "all", versions shall be created when any update is performed against the version-enabled data object. This data system metadata item shall not be present in data object versions.	Optional
cdmi_versions_count	JSON string	If this data system metadata item is present and not an empty string, it indicates that the client is requesting limits on the maximum number of historical versions to be retained. • If cdmi_versions_count is not present, no limits should be placed on the number of versions that are retained. • If cdmi_versions_count is present and has a value of zero, only the current version should be retained. • If cdmi_versions_count is present and has a value greater than zero, up to the specified number of historical versions should be retained. • If the number of historical versions exceeds the value specified, historical versions should be deleted from the oldest to the newest until the number of historical versions equals the value contained in cdmi_versions_count.	Optional
cdmi_versions_age	JSON string	If this data system metadata item is present and not an empty string, it indicates that the client is requesting limits on the maximum age of the oldest historical version requested to be retained. • If cdmi_versions_age is not present, no limit should be placed on the age of versions that are retained. • If cdmi_versions_age is present, historical versions should be retained until their age in seconds since creation is greater than the value contained in cdmi_versions_age. • If the age of a historical version exceeds the value specified, that historical version should be deleted.	Optional

Continued on next page

Table 142 – continued from previous page

Metadata name	Type	Description	Requirement
cdmi_versions_size	JSON string	If this data system metadata item is present and not an empty string, it indicates that the client is requesting limits on the maximum amount of space to be used to retain historical versions. • If cdmi_versions_size is not present, no limit should be placed on the size of versions that are retained. • If cdmi_versions_size is present, historic versions should be retained until the total storage consumption in bytes of the historical versions exceeds the value contained in cdmi_versions_size. • If the total size consumed by historical versions exceeds the value specified, historical versions should be deleted from the oldest to the newest until the total storage consumption of historical versions is equal or less than the value contained in cdmi_versions_size.	Optional

16.4 Support for provided data system metadata

For each metadata item in a data system, there is an actual value that the cloud service is able to achieve at this time, as shown in Table 143. Data system-provided metadata items are read only. Updates of these metadata items shall be ignored.

Table 143: Provided values of data system metadata

Metadata name	Type	Description	Requirement
cdmi_data_redundancy_provided	JSON string	Contains the current number of complete copies of the data object at this time	Optional
cdmi_immediate_redundancy_provided	JSON string	If present and set to "true", indicates if immediate redundancy is provided for the object	Optional
cdmi_infrastructure_redundancy_provided	JSON string	Contains the current number of independent storage infrastructures supporting the data currently operating	Optional
cdmi_data_dispersion_provided	JSON string	Contains the current lowest distance (km) between any two infrastructures hosting the data	Optional
cdmi_geographic_placement_provided	JSON array of JSON strings	Contains an ISO-3166 identifier that corresponds to a geopolitical region where the object is stored	Optional
cdmi_retention_period_provided	JSON string	Contains an ISO-8601 time interval (as described in 5.6) specifying the period the object is protected by retention	Optional
cdmi_retention_autodelete_provided	JSON string	Contains "true" if the object will automatically be deleted when retention expires	Optional
cdmi_hold_id_provided	JSON array of JSON strings	Contains the user-specified hold identifiers for active holds	Optional
cdmi_encryption_provided	JSON string	Contains the algorithm used for encryption, the mode of operation, and the key size. (See cdmi_encryption in 16.3 for the format.)	Optional
cdmi_value_hash_provided	JSON string	Contains the algorithm and length being used to hash the object value. See cdmi_value_hash in 16.3 for the format.	Optional
cdmi_latency_provided	JSON string	Contains the provided maximum time to first byte	Optional
cdmi_throughput_provided	JSON string	Contains the provided maximum data rate on retrieve	Optional
cdmi_sanitization_method_provided	JSON string	Contains the sanitization method used. See cdmi_sanitization_method in 16.3 for the format.	Optional
cdmi_RPO_provided	JSON string	Contains the provided duration, in seconds, between an update and when the update may be recovered	Optional
cdmi_RTO_provided	JSON string	Contains the provided duration, in seconds, to restore data	Optional

Continued on next page

Table 143 – continued from previous page

Metadata name	Type	Description	Requirement
cdmi_authentication_methods_provided	JSON array of JSON strings	Contains a list of authentication methods enabled for the domain. See cdmi_authentication_methods in 16.3 for the format.	Optional
cdmi_versioning_provided	JSON string	Contains the value "value", "user", or "all" if versioning is enabled for the data object.	Optional
cdmi_versions_count_provided	JSON string	Contains the maximum number of historical versions that will be retained.	Optional
cdmi_versions_age_provided	JSON string	Contains the oldest age of a historical version that will be retained, in seconds before the current time.	Optional
cdmi_versions_size_provided	JSON string	Contains the maximum amount of space that can be used to retain historical versions, in bytes.	Optional

16.5 Support for user metadata

All CDMI objects that support metadata shall permit the inclusion of arbitrary user-defined metadata items, with the restriction that the name of a user-defined metadata item shall not start with the prefix "`cdmi\_`".

- The maximum number of user-defined metadata items is specified by the capability `cdmi_metadata_maxitems`.
- The maximum size of each user-defined metadata item is specified by the capability `cdmi_metadata_maxsize`.
- The maximum total size of user-defined metadata items for an object is specified by the capability `cdmi_metadata_maxtotalsize`.

16.6 Metadata update operations

CDMI permits a client to replace all metadata items or to perform operations against one or more individual metadata items.

Replacing all metadata items is accomplished by including the metadata field in the update request body JSON and not specifying specific metadata items in the update URI.

Adding, updating, and removing specific metadata items is accomplished by specifying the specific metadata item names in the update URI:

- To add a new metadata item to an existing object, the metadata item name shall be included in the update request URI, and the metadata item shall be included in the metadata field in the update request body JSON.
- To update the value of an existing metadata item, the metadata item name shall be included in the update request URI, and the metadata item shall be included in the metadata field in the update request body JSON.
- To remove an existing metadata item, the metadata item name shall be included in the update request URI, and the metadata item shall not be included in the metadata field in the update request body JSON.

When individual metadata items are specified in the update URI, metadata items included in the metadata field in the request body JSON that are not referred to in the update URI shall be ignored.

Clause 17

Access control

17.1 Overview

Access control defines the mechanisms by which access to objects are permitted or denied. The CDMI International Standard supports the following options for access control:

- No access control
- Access Control List (ACL) based access control (See 17.2.1)
- Domain based access control (See 10.2.5)
- Delegated access control (See clause 24)
- Vendor-defined access control extensions
- Combinations of the above

17.2 Access control flow

Fig. 11 illustrates the control flow for access control in an example CDMI implementation. As every aspect of access control is optional within a CDMI server, each different implementations will typically implement appropriate subsets of the illustrated access control flow, in a manner approprate to the internal architecture of their implementation.

The full contol flow can include 24 steps:

1. The CDMI client initiates a CDMI operation by sending a CDMI request to a CDMI server. As part of the request, the CDMI client includes information about its identity and information to prove this identity (credentials). The method by which these credentials are presented and formatted is not specified in this International Standard, however, some guidance is provided in 5.4.3.
2. If the CDMI server supports Domains (see clause 10), the CDMI server obtains the domain associated with the object the CDMI operation is being performed against. If the CDMI system does not support domains, steps 2 - 8 are skipped.
3. The CDMI server obtains required information about the domain associated with the object.
4. Domain Information is returned for further use.
5. Domain information is used to resolve CDMI client credentials.
6. If the Domain is configured to delegate identity resolution to an external system (such as Active Directory), credentials are sent to this external system for resolution.
7. If the Domain is configured to use local membership, credentials are compared against the configured domain members (see 10.4).
8. The resolved principle (user, group, indication of validity) is returned for further use.
9. If the CDMI server supports ACLs (see 17.2.1), the CDMI server evaluates the object ACL. If the CDMI system does not support ACLs, steps 9 - 15 are skipped.
10. The CDMI ACL processing subsystem obtains the ACL for the object.
11. The CDMI server obtains ACL metadata associated with the object.
12. If the object is in a container, the CDMI server obtains ACL metadata for parent containers.
13. The obtained ACL metadata is returned for further use.
14. The CDMI ACL processing subsystem evaluates the resolved principals against the resolved ACL.
15. The evaluated permission mask is returned for further use.
16. If the CDMI server supports Delegated Access Control (DAC) (see clause 24), the CDMI server obtains DAC metadata associated with the object the CDMI operation is being performed against. If the CDMI system does not support DAC, steps 16 - 22 are skipped.
17. The CDMI server obtains DAC metadata associated with the object.
18. DAC metadata is returned for further use.
19. If DAC metadata is present and indicates that DAC is to be used, the specified delegation is performed.
20. The external DAC provider is contacted, including the evaluated Object permission mask.
21. If a valid DAC response is received, the dac_applied_mask replaces the evaluated Object permission mask.
22. The DAC results and Object permission mask is returned for further use.
23. The Object permission mask is used to determine if the requested operation is permitted.
24. The operation is permitted or denied, and the corresponding response returend to the CDMI Client.

Steps 2 - 8, 9 - 15, and 16 - 18 may be performed in parallel.

17.2.1 General mechanisms

CDMI uses the well-known mechanism of an Access Control List (ACL) as defined in the NFSv4 standard (see RFC 3530 [1]). ACLs are lists of permissions-granting or permissions-denying entries called Access Control Entries (ACEs).

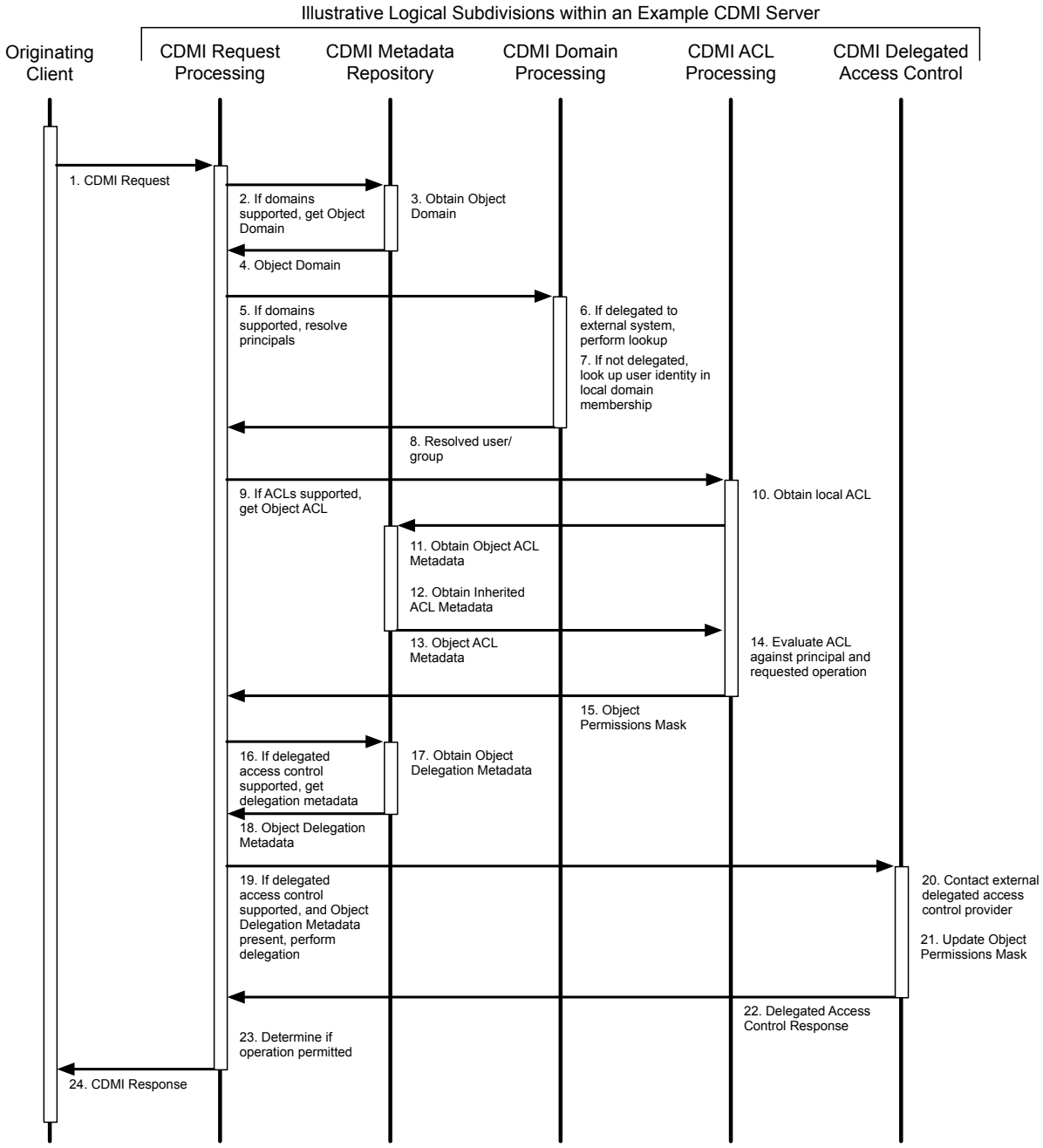


Fig. 11: Access control flow

17.2.2 ACL and ACE structure

An ACL is an ordered list of ACEs. The two types of ACEs in CDMI are `ALLOW` and `DENY`. An `ALLOW` ACE grants some form of access to a principal. Principals are either users or groups and are represented by identifiers. A `DENY` ACE denies access of some kind to a principal. For instance, a `DENY` ACE may deny the ability to write the metadata or ACL of an object but may remain silent on other forms of access. In that case, if another ACE ALLOWs write access to the object, the principal is allowed to write the object's data, but nothing else.

ACEs are composed of four fields: `type`, `who`, `flags` and `access_mask`, as per RFC 3530 [1]. The `type`, `flags`, and `access_mask` shall be specified as either unsigned integers in hex string representation or as a comma-delimited list of bit mask string form values taken from *ACE types*, *ACE flags*, and *ACE masks bits*.

17.2.3 ACE types

Table 144 defines the following ACE types, as specified in section 5.11.1 of RFC 3530 [1].

Table 144: ACE types

String form	Description	Constant	Bit mask
"`ALLOW`"	Allow access rights for a principal	"`CDMI_ACE_ACCESS_ALLOW`"	0x00000000
"`DENY`"	Deny access rights for a principal	"`CDMI_ACE_ACCESS_DENY`"	0x00000001
"`AUDIT`"	Generate an audit record when the principal attempts to exercise the specified access rights	"`CDMI_ACE_SYSTEM_AUDIT`"	0x00000002

The reason that the string forms may be safely abbreviated is that they are local to the ACE structure type, as opposed to constants, which are relatively global in scope.

The client is responsible for ordering the ACEs in an ACL. The server shall not enforce any ordering and shall store and evaluate the ACEs in the order given by the client.

17.2.4 ACE who

The special "`who`" identifiers need to be understood universally, rather than in the context of a particular external security domain (see *Who identifiers*). Some of these identifiers may not be understood when a CDMI client accesses the server, but they may have meaning when a local process accesses the file. The ability to display and modify these permissions is permitted over CDMI, even if none of the access methods on the server understands the identifiers.

Table 145: Who identifiers

Who	Description
"`OWNER@`"	The owner of the file
"`GROUP@`"	The group associated with the file
"`EVERYONE@`"	The world
"`ANONYMOUS@`"	Access without authentication
"`AUTHENTICATED@`"	Any authenticated user (opposite of "`ANONYMOUS@`")
"`ADMINISTRATOR@`"	A user with administrative status, e.g., "`root`"
"`ADMINUSERS@`"	A group whose members are given administrative status

To avoid name conflicts, these special identifiers are distinguished by an appended "`@`" (with no domain name).

17.2.5 ACE flags

CDMI allows for nested containers and mandates that objects and subcontainers be able to inherit access permissions from their parent containers. However, it is not enough to simply inherit all permissions from the parent; it might be desirable, for example, to have different default permissions on child objects and subcontainers of a given container. The flags in Table 146 govern this behavior.

Table 146: ACE flags

String form	Description	Constant	Bit mask
"NO_FLAGS"	No flags are set	"CDMI_ACE_FLAGS_NONE"	0x00000000
"OBJECT_INHERIT"	An ACE on which "OBJECT_INHERIT" is set is inherited by objects as an effective ACE: "OBJECT_INHERIT" is cleared on the child object. When the ACE is inherited by a container, "OBJECT_INHERIT" is retained for the purpose of inheritance, and additionally, "INHERIT_ONLY" is set.	"CDMI_ACE_FLAGS_ OBJECT_INHERIT_ACE"	0x00000001
"CONTAINER_INHERIT"	An ACE on which "CONTAINER_INHERIT" is set is inherited by a subcontainer as an effective ACE. Both "INHERIT_ONLY" and "CONTAINER_INHERIT" are cleared on the child container.	"CDMI_ACE_FLAGS_ CONTAINER_INHERIT_ACE"	0x00000002
"NO_PROPAGATE"	An ACE on which "NO_PROPAGATE" is set is not inherited by any objects or subcontainers. It applies only to the container on which it is set.	"CDMI_ACE_FLAGS_ NO_PROPAGATE_ACE"	0x00000004
"INHERIT_ONLY"	An ACE on which "INHERIT_ONLY" is set is propagated to children during ACL inheritance as specified by "OBJECT_INHERIT" and "CONTAINER_INHERIT". The ACE is ignored when evaluating access to the container on which it is set and is always ignored when set on objects.	"CDMI_ACE_FLAGS_ INHERIT_ONLY_ACE"	0x00000008
"IDENTIFIER_GROUP"	An ACE on which "IDENTIFIER_GROUP" is set indicates that the "who" refers to a group identifier.	"CDMI_ACE_FLAGS_ IDENTIFIER_GROUP"	0x00000040
"INHERITED"	An ACE on which "INHERITED" is set indicates that this ACE is inherited from a parent directory. A server that supports automatic inheritance will place this flag on any ACEs inherited from the parent directory when creating a new object.	"CDMI_ACE_FLAGS_ INHERITED_ACE"	0x00000080

17.2.6 ACE mask bits

The mask field of an ACE contains a 32 bit mask, as specified in section 5.11.2 of RFC 3530 [1]. Table 146 defines the impact of each bit in an ACE mask field.

Table 147: ACE masks bits

String form	Description	Constant	Bit mask
"READ_OBJECT"	If true, indicates permission to read the value of an object. If false: • A CDMI GET that requests all fields shall return all permitted fields with the value field excluded. • A CDMI GET that requests specific fields shall return requested permitted fields with the value field excluded. • A CDMI GET for only the value field shall return an HTTP status code of 403 Forbidden. • A non-CDMI GET shall return an HTTP status code of 403 Forbidden.	"CDMI_ACE_READ_OBJECT"	0x00000001
"LIST_CONTAINER"	If true, indicates permission to list the children of an object. If false: • A CDMI GET that requests all fields shall return all permitted fields with the children field and childrenrange field excluded. • A CDMI GET that requests specific fields shall return the requested permitted fields with the children field and childrenrange field excluded. • A CDMI GET for only the children field and/or childrenrange field shall return an HTTP status code of 403 Forbidden.	"CDMI_ACE_LIST_CONTAINER"	0x00000001
"WRITE_OBJECT"	If true, indicates permission to modify the value of an object If false, a PUT that requests modification of the value of an object shall return an HTTP status code of 403 Forbidden.	"CDMI_ACE_WRITE_OBJECT"	0x00000002
"ADD_OBJECT"	If true, indicates permission to add a new child data object or queue object. If false, a PUT or POST that requests creation of a new child data object or new queue object shall return an HTTP status code of 403 Forbidden.	"CDMI_ACE_ADD_OBJECT"	0x00000002

Continued on next page

Table 147 – continued from previous page

String form	Description	Constant	Bit mask
"APPEND_DATA"	If true, indicates permission to append data to the value of a data object. If "APPEND_DATA" is true and "WRITE_OBJECT" is false, a PUT that requests modification of any existing part of the value of an object shall return an HTTP status code of 403 Forbidden.	"CDMI_ACE_APPEND_DATA"	0x00000004
"ADD_SUBCONTAINER"	If true, indicates permission to create a child container object or domain object. If false, a PUT that requests creation of a new child container object or new domain object shall return an HTTP status code of 403 Forbidden.	"CDMI_ACE_ADD_SUBCONTAINER"	0x00000004
"READ_METADATA"	If true, indicates permission to read the metadata of an object. If false: • A CDMI GET that requests all fields shall return all permitted fields with the metadata field excluded. • A CDMI GET that requests specific fields shall return the requested permitted fields with the metadata field excluded. • A CDMI GET for only the metadata field shall return an HTTP status code of 403 Forbidden.	"CDMI_ACE_READ_METADATA"	0x00000008
"WRITE_METADATA"	If true, indicates permission to modify the metadata of an object. If false, a CDMI PUT that requests modification of the metadata field of an object shall return an HTTP status code of 403 Forbidden.	"CDMI_ACE_WRITE_METADATA"	0x00000010
"EXECUTE"	If true, indicates permission to execute an object.	"CDMI_ACE_EXECUTE"	0x00000020
"TRAVERSE_CONTAINER"	If true, indicates permission to traverse a container object or domain object. If false, all operations against all children below the container shall return an HTTP status code of 403 Forbidden.	"CDMI_ACE_TRAVERSE_CONTAINER"	0x00000020
"DELETE_OBJECT"	If true, indicates permission to delete a child data object or child queue object from a container object. If false, all DELETE operations shall return an HTTP status code of 403 Forbidden.	"CDMI_ACE_DELETE_OBJECT"	0x00000040

Continued on next page

Table 147 – continued from previous page

String form	Description	Constant	Bit mask
"DELETE_SUBCONTAINER"	If true, indicates permission to delete a child container object from a container object or to delete a child domain object from a domain object. If false, all DELETE operations shall return an HTTP status code of 403 Forbidden.	"CDMI_ACE_DELETE_SUBCONTAINER"	0x00000040
"READ_ATTRIBUTES"	If true, indicates permission to read the attribute fields[1] of an object. If false: • A CDMI GET that requests all fields shall return all non-attribute fields and shall not return any attribute fields. • A CDMI GET that requests at least one non-attribute field shall only return the requested non-attribute fields. • A CDMI GET that requests only non-attribute fields shall return an HTTP status code of 403 Forbidden.	"CDMI_ACE_READ_ATTRIBUTES"	0x00000080
"WRITE_ATTRIBUTES"	If true, indicates permission to change attribute fields[#a]_ of an object. If false, a CDMI PUT that requests modification of any non-attribute field shall return an HTTP status code of 403 Forbidden.	"CDMI_ACE_WRITE_ATTRIBUTES"	0x00000100
"WRITE_RETENTION"	If true, indicates permission to change retention attributes of an object. If false, a CDMI PUT that requests modification of any non-hold retention metadata items shall return an HTTP status code of 403 Forbidden.	"CDMI_ACE_WRITE_RETENTION"	0x00000200
"WRITE_RETENTION_HOLD"	If true, indicates permission to change retention hold attributes of an object. If false, a CDMI PUT that requests modification of any retention hold metadata items shall return an HTTP status code of 403 Forbidden.	"CDMI_ACE_WRITE_RETENTION_HOLD"	0x00000400
"DELETE"	If true, indicates permission to delete an object. If false, all DELETE operations shall return an HTTP status code of 403 Forbidden.	"CDMI_ACE_DELETE"	0x00010000

Continued on next page

Table 147 – continued from previous page

String form	Description	Constant	Bit mask
"READ_ACL"	If true, indicates permission to read the ACL of an object. If false: • A CDMI GET that requests all metadata items shall return all permitted metadata items with the "cdmi_acl" metadata item excluded. • A CDMI GET that requests specific metadata items shall return the requested permitted metadata items with the "cdmi_acl" metadata item excluded. • A CDMI GET for only the cdmi_acl metadata item shall return an HTTP status code of 403 Forbidden. If "READ_ACL" is true and "READ_METADATA" is false, then to read the ACL, a client CDMI GET for only the "cdmi_acl" metadata item shall be permitted.	"CDMI_ACE_READ_ACL"	0x00020000
"WRITE_ACL"	If true, indicates permission to write the ACL of an object. If false: • If "WRITE_ACL" is false, a CDMI PUT that requests modification of the "cdmi_acl" metadata item shall return an HTTP status code of 403 Forbidden. • If "WRITE_ACL" is true and "WRITE_METADATA" is false, then to write the ACL, a client CDMI PUT for only the "cdmi_acl" metadata item shall be permitted.	"CDMI_ACE_WRITE_ACL"	0x00040000
"WRITE_OWNER"	If true, indicates permission to change the owner of an object. If false: • If "WRITE_OWNER" is false, a CDMI PUT that requests modification of the "cdmi_owner" metadata item shall return an HTTP status code of 403 Forbidden. • If "WRITE_OWNER" is true and "WRITE_METADATA" is false, then to write the owner, a client CDMI PUT for only the "cdmi_owner" metadata item shall be permitted.	"CDMI_ACE_WRITE_OWNER"	0x00080000

Continued on next page

Table 147 – continued from previous page

String form	Description	Constant	Bit mask
"SYNCHRONIZE"	If true, indicates permission to access an object locally at the server with synchronous reads and writes.	"CDMI_ACE_SYNCHRONIZE"	0x00100000

Implementations shall use the correct string form to display permissions, if the object type is known. If the object type is unknown, the "object" version of the string shall be used.

17.2.7 ACL evaluation

When evaluating whether access to a particular object O by a principal P is to be granted, the server shall traverse the object's logical ACL (its ACL after processing inheritance from parent containers) in list order, using a temporary permissions bitmask m, initially empty (all zeroes), and apply the following algorithm:

- If the object still does not contain an ACL, the algorithm terminates and access is denied for all users and groups. This condition is not expected, as CDMI implementations should require an inheritable default ACL on all root containers.
- ACEs that do not refer to the principal P requesting the operation are ignored.
- If an ACE is encountered that denies access to P for any of the requested mask bits, access is denied and the algorithm terminates.
- If an ACE is encountered that allows access to P, the permissions mask m for the operation is XORed with the permissions mask from the ACE. If m is sufficient for the operation, access is granted and the algorithm terminates.
- If the end of the ACL list is reached and permission has neither been granted nor explicitly denied, access is denied and the algorithm terminates, unless the object is a container root. In this case, the server shall:
  - allow access to the container owner, "ADMINISTRATOR@", and any member of "ADMINUSERS@"; and
  - log an event indicating what has happened.

When permission for the desired access is not explicitly given, even "ADMINISTRATOR@" and equivalents are denied for objects that aren't container roots. When an admin needs to access an object in such an instance, the root container shall be accessed and its inheritable ACEs changed in a way as to allow access to the original object. The resulting log entry then provides an audit trail for the access.

When a root container is created and no ACL is supplied, the server shall place an ACL containing the following ACEs on the container:

```
"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"
    }
]
```

As ACLs are storage system metadata, they are stored and retrieved through the metadata field included in a PUT or GET request. The syntax is as follows, using the constant strings from *ACE types*, *ACE flags*, and *ACE masks bits*:

```
ACL = { ACE [, ACE ...] }
ACE = { acetype , identifier , aceflags , acemask }
acetype = uint_t | acetypeitem
```

(continues on next page)

[1] The value fields, children fields, and metadata field are considered to be non-attribute fields. All other fields are considered to be attribute fields.

(continued from previous page)

```
identifier  = utf8string_t
aceflags    = uint_t | aceflagsstring
acemask     = uint_t | acemaskstring

acetypeitem = aceallowedtype | acedeniedtype | aceaudittype
aceallowedtype = "CDMI_ACE_ACCESS_ALLOWED_TYPE" | 0x0
acedeniedtype  = "CDMI_ACE_ACCESS_DENIED_TYPE" | 0x01
aceaudittype   = "CDMI_ACE_SYSTEM_AUDIT_TYPE" | 0x02

aceflagsstring = aceflagsitem [| aceflagsitem ...]
aceflagsitem   = aceobinherititem | acecontinherititem | acenopropagateitem |
↪aceinheritonlyitem

aceobinheritiitem   = "CDMI_ACE_OBJECT_INHERIT_ACE" | 0x01
acecontinheritiitem = "CDMI_ACE_CONTAINER_INHERIT_ACE" | 0x02
acenopropagateitem = "CDMI_ACE_NO_PROPAGATE_INHERIT_ACE" | 0x04
aceinheritonlyitem = "CDMI_ACE_INHERIT_ONLY_ACE" | 0x08

acemaskstring  =   acemaskitem [| acemaskitem ...]
acemaskitem    =   acereaditem | acewriteitem | aceappenditem | acereadmetaitem |
↪acewritemetaitem | acedeleteitem | acedelselfitem | acereadaclitem | acewriteaclitem |
↪aceexecuteitem | acereadattritem | acewriteattritem | aceretentionitem

acereaditem       = "CDMI_ACE_READ_OBJECT" | "CDMI_ACE_LIST_CONTAINER" | 0x01
acewriteitem      = "CDMI_ACE_WRITE_OBJECT" | "CDMI_ACE_ADD_OBJECT" | 0x02
aceappenditem     = "CDMI_ACE_APPEND_DATA" |  "CDMI_ACE_ADD_SUBCONTAINER" |  0x04
acereadmetaitem   = "CDMI_ACE_READ_METADATA" | 0x08
acewritemetaitem  = "CDMI_ACE_WRITE_METADATA" | 0x10
acedeleteitem     = "CDMI_ACE_DELETE_OBJECT" | "CDMI_ACE_DELETE_SUBCONTAINER" | 0x40
acedelselfitem    = "CDMI_ACE_DELETE" | 0x10000
acereadaclitem    = "CDMI_ACE_READ_ACL" | 0x20000
acewriteaclitem   = "CDMI_ACE_WRITE_ACL" | 0x40000
aceexecuteitem    = "CDMI_ACE_EXECUTE" | 0x80000
acereadattritem   = "CDMI_ACE_READ_ATTRIBUTES" | 0x00080
acewriteattritem  = "CDMI_ACE_WRITE_ATTRIBUTES" | 0x00100
aceretentionitem  = "CDMI_ACE_SET_RETENTION" | 0x10000000
```

When ACE masks are presented in numeric format, they shall, at all times, be specified in hexadecimal notation with a leading "0x". This format allows both servers and clients to quickly determine which of the two forms of a given constant is being used. When masks are presented in string format, they shall be converted to numeric format and then evaluated using standard bitwise operators.

When an object is created, no ACL is supplied, and an ACL is not inherited from the parent container (or there is no parent container), the server shall place an ACL containing the following ACEs on the object:

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

17.2.8 Example ACE mask expressions

Example 1:

```
"READ_ALL" | 0x02
```

evaluates to 0x09 | 0x02 == 0x0

Example 2:

```
0x001F07FF
```

evaluates to 0x001F07FF == “ALL_PERMS”

Example 3:

```
"RW_ALL" | DELETE
```

evaluates to 0x000601DF | 0x00100000 == 0x000701DF

17.2.9 Canonical format for ACE hexadecimal quantities

ACE mask expressions may be evaluated and converted to a string hexadecimal value before transmission in a CDMI JSON body. Applications or utilities that display them to users should convert them into a text expression before display and accept user input in text format as well.

The following technique should be used to decompose masks into strings. A table of masks and string equivalents should be maintained and ordered from greatest to least:

Table 148: ACE bit mask/string

Hex form	Object string form	Container string form
0x001F07FF	“ALL_PERMS”	“ALL_PERMS”
0x0006006F	“RW_ALL”	“RW_ALL”
0x0000001F	“RW”	“RW”
	…	…
0x00000002	“WRITE_OBJECT”	“ADD_OBJECT”
0x00000001	“READ_OBJECT”	“LIST_CONTAINER”

Given an access mask M, the following is repeated until M == 0:

1. Select the highest mask m from the table such that M & m == m.
2. If the object is a container, select the string from the 3rd column; otherwise, select the string from the 2nd column.
3. Bitwise subtract m from M, i.e., set M = M xor m.
4. The complete textual representation is then all the selected strings concatenated with “, ``" between them, e.g., "``ALL_PERMS, WRITE_OWNER”. The strings should appear in the order they are selected.

A similar technique should be used for all other sets of hex/string equivalents.

This algorithm, properly coded, requires only one (often partial) pass through the corresponding string equivalents table.

17.2.10 JSON format for ACLs

ACE flags and masks are members of a 32-bit quantity that is widely understood in its hexadecimal representations. The JSON data format does not support hexadecimal integers, however. For this reason, all hexadecimal integers in CDMI ACLs shall be represented as quoted strings containing a leading “0x”.

ACLs containing one or more ACEs shall be represented in JSON as follows:

```
{
    "cdmi_acl" : [
        {
            "acetype" : "0xnn",
            "identifier" : "<user-or-group-name>",
            "aceflags" : "0xnn",
            "acemask" : "0xnn"
        },
        {
            "acetype" : "0xnn",
            "identifier" : "<user-or-group-name>",
            "aceflags" : "0xnn",
            "acemask" : "0xnn"
        }
    ]
```

(continues on next page)

(continued from previous page)

```
}

ACEs in such an ACL shall be evaluated in order as they appear.
```

3327 EXAMPLE 1: An example of an ACL embedded in a response to a GET request is as follows:

```
HTTP/1.1 200 OK
Content-Type: application/cdmi-object

{
    "objectType" : "/application/cdmi-object",
    "objectID" : "00007ED9001086A99CC6487FEE373D82",
    "objectName" : "MyDataItem.txt",
    "parentURI" : "/MyContainer/",
    "domainURI" : "/cdmi_domains/MyDomain/",
    "capabilitiesURI" : "/cdmi_capabilities/dataobject/",
    "completionStatus" : "Complete",
    "mimetype" : "text/plain",
    "metadata" : {
        "cdmi_size" : "17",
        "cdmi_acl" : [
            {
                "acetype" : "0x00",
                "identifier" : "EVERYONE@",
                "aceflags" : "0x00",
                "acemask" : "0x00020089"
            }
        ],
        ...
    },
    "valuerange" : "0-16",
    "value" : "Hello CDMI World!"
}
```

Clause 18

Retention and hold management

18.1 Overview

A cloud storage system may optionally implement retention management disciplines into the system management functionality of the cloud-based storage system. The implementation of retention and hold capabilities is indicated by the presence of the cloud storage system-wide capabilities for retention and hold capabilities.

Retention management includes implementing a retention policy, defining a hold policy to enable objects to be held for specific purposes (e.g., litigation), and defining how the rules for deleting objects are affected by placing either a retention policy and/or a hold on an object. CDMI™ object deletion is not a capability of retention management, *per se*, but rather is a general system capability. However, this clause describes what happens when placing either a retention policy and/or a hold on an object.

Retention management may be applied to the following object types:

- data objects,
- queue objects, and
- container objects.

18.2 Retention management disciplines

CDMI retention, deletion, and hold management affect any CDMI client that creates or deletes CDMI objects, as these disciplines mandate how a cloud storage system manages CDMI objects when they are created and until they are deleted.

CDMI retention management is comprised of three management disciplines: retention, hold, and deletion:

- CDMI retention uses retention time criteria to determine the time period during which object deletion from the CDMI-based system is prohibited. No changes to the object are allowed, even after the retention period has expired, except as specified below.
- CDMI hold prohibits object deletion and modification until all holds on the object have been released.
- A CDMI-based system shall not allow the deletion of a CDMI object before the CDMI retention time criteria are met or while holds exist. Any deletion attempts (e.g., by a CDMI application) shall return an error.
- After the CDMI retention time criteria have been met and all holds have been released, CDMI retention and holds shall no longer be a reason to prohibit object deletion.
- Once the retention period has started or if holds exist, changes to the object data and metadata shall not be allowed, with the exception of extensions to the retention and hold data system metadata. The retention data system metadata may be added or the retention period extended, and the hold data system metadata may be added or extended with additional holds. Any other attempt to modify the object shall return an error.

18.3 CDMI retention

18.3.1 Overview

CDMI retention only allows one retention policy to be applied to an object at a time.

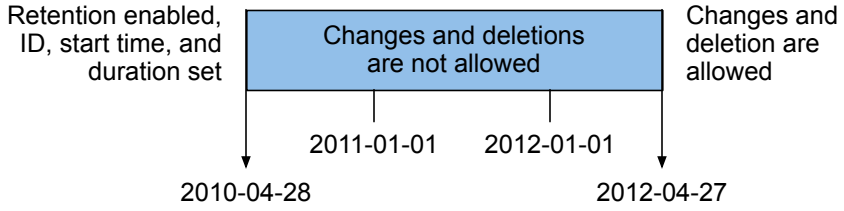
Retention management uses time criteria to determine the time period during which CDMI object deletion from the CDMI-based system shall be prohibited. CDMI retention criteria shall be specified by the following data system metadata:

- a retention criteria identifier—a CDMI client-specified string that shall identify the retention records class (`cdmi_retention_id`); and
- a retention start time and retention period time—the start time, when used together with period, indicating when retention shall no longer be enforced (`cdmi_retention_period`).

When a CDMI client attempts to delete an object, the cloud storage system shall evaluate all such retention criteria and return an error, if any retention criteria have not been met.

When copying objects with a retention policy, retention properties shall not be transferred from the source CDMI object to the destination object, and the destination object shall not have a retention policy.

Fig. 12 shows how to establish time-based retention with a retention identifier. The value of the object data system metadata for the retention period shall not be reduced. Removing holds is outside the scope of the CDMI International Standard.



Example: Retention start date of 2010-04-28 with a duration of 730 days. No holds.

Fig. 12: Object retention

A specific HTTP error code (403) shall be returned on operations to objects that are under retention period when the cloud storage system attempts to change or delete the object before the retention period criteria are met.

A cloud storage system shall not prevent metadata changes that increase the retention period, as there are valid business reasons to change a retention period for an object.

18.3.2 Examples

EXAMPLE 1: Place an existing object under retention:

```
--> PATCH /cdmi/2.0.0/MyContainer/MyDataObject.txt?metadata=cdmi_retention_id&
↪metadata=cdmi_retention_period HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-object
-->
--> {
-->   "metadata" : {
-->     "cdmi_retention_id" : "1",
-->     "cdmi_retention_period" : "2010-04-28T00:00:00.000000Z/2012-04-27T00:00:00.
↪000000Z"
-->   }
--> }

<-- HTTP/1.1 204 No Content
```

EXAMPLE 2: Increase the duration of retention on an existing object under retention:

```
--> PATCH /cdmi/2.0.0/MyContainer/MyDataObject.txt?metadata=cdmi_retention_period
↪HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-object
-->
--> {
-->   "metadata" : {
-->     "cdmi_retention_period" : "2011-04-28T00:00:00.000000Z/2013-04-27T00:00:00.
↪000000Z"
-->   }
--> }

<-- HTTP/1.1 204 No Content
```

3383 EXAMPLE 3: Decrease the duration of retention on an existing object under retention:

```
--> PATCH /cdmi/2.0.0/MyContainer/MyDataObject.txt?metadata=cdmi_retention_period
↪HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-object
-->
--> {
-->   "metadata" : {
-->     "cdmi_retention_period" : "2011-04-28T00:00:00.000000Z/2012-01-27T00:00:00.
↪000000Z"
-->   }
--> }

<-- HTTP/1.1 403 Forbidden
```

18.4 CDMI hold

18.4.1 Overview

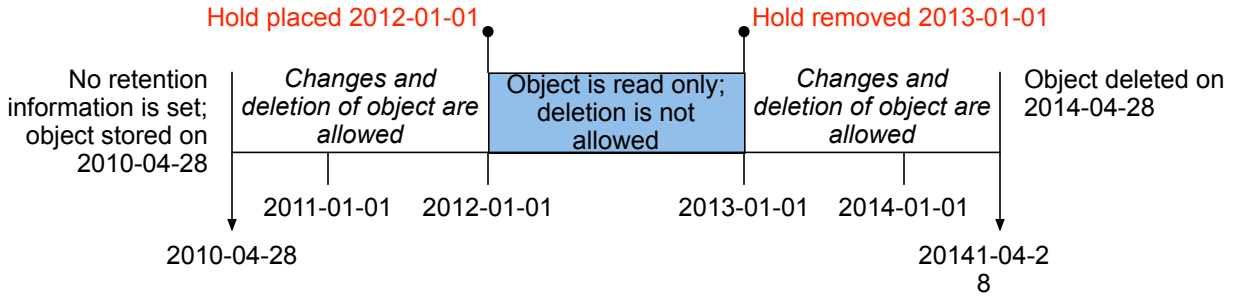
CDMI hold enforces read-only data object access and prohibition of object deletion. A cloud storage system shall allow multiple holds to be applied to a single object to satisfy multiple hold orders. While an object is on hold, a cloud storage system shall strictly enforce read-only access to the object and prohibit object deletion.

When copying objects that are on hold, hold properties shall not be transferred from the source CDMI object to the destination object, and the destination object shall not be on hold.

Hold management uses a hold indicator to determine the time periods during which CDMI object revision (data and metadata) and deletion from the CDMI-based system shall be prohibited. CDMI hold criteria shall be specified by data system metadata, specifically, a hold criteria identifier that is a client-specified string that shall identify the holds and their order.

A CDMI client may place an object on hold by adding a hold identifier to the `cdmi_hold_id` data system metadata item. When an object is on hold, CDMI clients shall be subject to failures or unexpected state changes on operations, which would otherwise be successful if the object was not on hold.

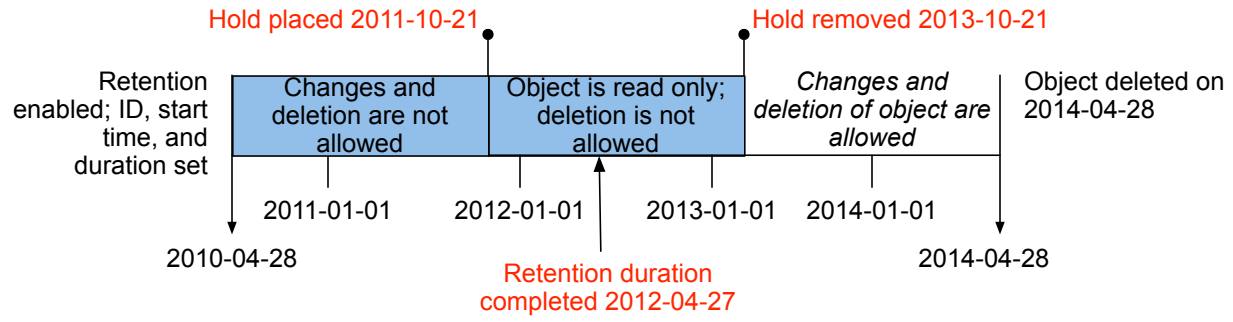
Fig. 13 shows how placing a hold on an object affects its read-only and deletion capability.



Example: Hold placed on the object on 2012-01-01 and removed on 2013-01-01

Fig. 13: Object hold

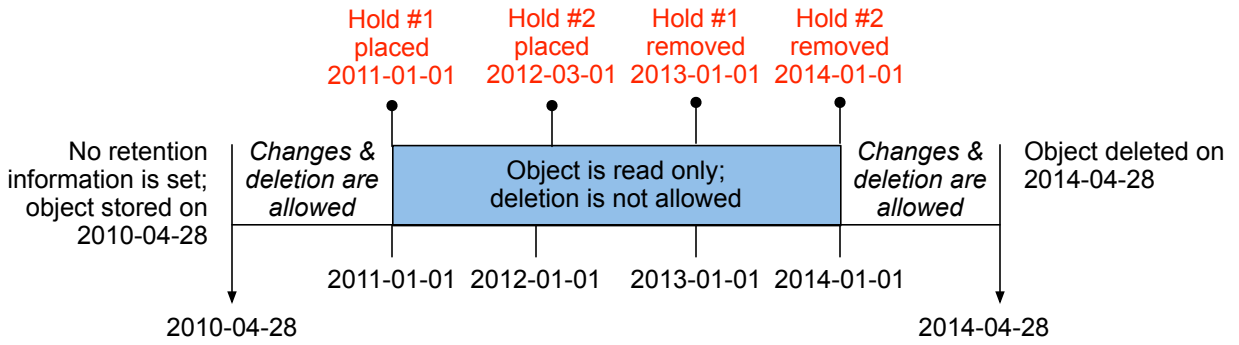
Fig. 14 shows how to establish time-based retention with a retention identifier that has a hold placed on the object. The value of the object data system metadata for the retention period shall not be reduced, and the value of the object data system metadata for hold identifiers shall not permit holds to be removed. Removing holds is outside the scope of the CDMI International Standard.



Example: Start date of 2010-04-28 with a duration of 730 days; hold placed on the object

Fig. 14: Object hold on object with retention

Fig. 15 shows how placing multiple holds on an object affects its read-only and deletion capability.



Example: Start date of 2010-04-28 with a duration of 730 days; holds placed on the object

Fig. 15: Object with multiple holds

A cloud storage system shall maintain an on-hold object in read-only mode with respect to the application access to data and metadata and shall prohibit deletion, either automated or explicit.

- CDMI clients shall tolerate these object on-hold failures or state changes.
- Releases from hold are not part of this International Standard and are typically performed out of band using an additionally secured non-CDMI mechanism provided by the implementation.

A specific HTTP error code (403) shall be returned on operations to objects that are under a hold when the system attempts to change the object or attempts to delete the object before the hold is removed. This failure should be a an error to the application.

18.4.2 Examples

EXAMPLE 1: Place an existing object under hold:

```
--> PATCH /cdmi/2.0.0/MyContainer/MyDataObject.txt?metadata=cdmi_hold_id HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-object
-->
--> {
-->   "metadata": {
-->     "cdmi_hold_id": {
-->       "case_7": ""
-->     }
-->   }
--> }

<-- HTTP/1.1 204 No Content
```

EXAMPLE 2: Attempt to remove a hold for an object under hold:

```
--> PATCH /cdmi/2.0.0/MyContainer/MyDataObject.txt?metadata=cdmi_hold_id HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-object
-->
--> {
-->   "metadata": {
-->     "cdmi_hold_id": {}
-->   }
--> }

<-- HTTP/1.1 403 Forbidden
```

EXAMPLE 3: Add a second hold to an object under hold:

```
--> PATCH /cdmi/2.0.0/MyContainer/MyDataObject.txt?metadata=cdmi_hold_id HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-object
-->
--> {
-->   "metadata":{
-->     "cdmi_hold_id": {
-->       "case_7": "",
-->       "case_15": ""
-->     }
-->   }
--> }

<-- HTTP/1.1 204 No Content
```

18.5 CDMI auto-deletion

18.5.1 Overview

CDMI deletion controls cloud storage system actions with respect to object deletion. A cloud storage system may automatically delete a CDMI object after the retention time and hold criteria have been met. (See `cdmi_retention_autodelete` in *Data system metadata*.)

CDMI objects shall be automatically deleted by the system at the retention period expiration by setting the `cdmi_retention_autodelete` data system metadata item. The `cdmi_retention_autodelete` data system metadata item indicates to the system that the object shall be made unavailable for access after the retention criteria have been satisfied. The system shall ensure that the object is no longer available through the CDMI interface. If the system has satisfied the retention requirement and a hold is established for the object, the object shall not be made unavailable or deleted. When a hold and retention have been applied to an object, both need to be satisfied (retention period expired and no holds existing) for objects to be automatically deleted from the system.

EXAMPLE 1: Place an object under retention with autodelete:

```
--> PATCH /cdmi/2.0.0/MyContainer/MyDataObject.txt?metadata=cdmi_retention_period&
↪metadata=cdmi_retention_autodelete HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: application/cdmi-object
-->
--> {
-->   "metadata":{
-->     "cdmi_retention_period": "2011-04-28T00:00:00.000000Z/2013-04-27T00:00:00.
↪000000Z",
-->     "cdmi_retention_autodelete": "true"
-->   }
--> }

<-- HTTP/1.1 204 No Content
```

18.6 Retention security considerations

The accuracy and integrity of the retention start and elapsed times depend on the accuracy and integrity of the clock that is used to set their values. Equally important is the relative accuracy and security of the clock that determines if the retention period has elapsed when compared to the clock that sets the start time property. Relative time differences between these two clocks may lead to undesirable retention and deletion management behavior.

It is important to have a reliable source from which the system clock is set. A stratum 1 time is directly connected to a reference clock and is at the top of the time server hierarchy. Relative time differences between the system clock and the reference clock may lead to undesirable retention timestamps and difficulties with time action events.

EXAMPLE 1: An object is created in a cloud storage system at time 0 with a period of 8 years and `autodelete` of `true`. At time 1 year, the system clock is adjusted forward to 9 years. Now, because the system time is 9 years, the retention time criterion is satisfied, even though only 1 year has actually elapsed. And, since `autodelete` is `true`, the system automatically deletes the object.

The specification for accuracy and integrity of timekeeping is not within the scope of CDMI. However, to prevent undesirable retention and deletion management consequences, systems should maintain accurate clock time, with zero or minimal deviation to clock integrity.

Clause 19

Scope specification

19.1 Overview

CDMI™ provides a standardized mechanism to define sets of objects that match certain characteristics. This mechanism is known as a CDMI scope specification. Scope specifications are typically used to provide a CDMI client with a way to indicate in what set of CDMI objects it is interested.

Each JSON object within the scope specification represents a set of conditions that shall all be true in order for an object to be considered to match against the scope (a logical AND relationship). For queries, a matching object would be returned in the query results. An empty scope specification is considered to evaluate to true. Multiple JSON objects are used to express logical OR relationships, where if any JSON object in the scope evaluates to true, then the object shall be considered to have matched against the scope.

Each JSON object is constructed using the same structure that CDMI objects use. To show this structure, assume the following result from a GET for a data object:

```
HTTP/1.1 200 OK
Content-Type: application/cdmi-object

{
    "objectType" : "application/cdmi-object",
    "objectID" : "00007E7F0010EB9092B29F6CD6AD6824",
    "objectName" : "MyDataObject.txt",
    "parentURI" : "/MyContainer/",
     "parentID" : "00007E7F00102E230ED82694DAA975D2",
    "domainURI" : "/cdmi_domains/MyDomain/",
    "capabilitiesURI" : "/cdmi_capabilities/dataobject/",
    "completionStatus" : "Complete",
    "mimetype" : "text/plain",
    "metadata" : {
        "cdmi_size" : "108263",
        ...
    },
    "valuerange" : "0-108262",
    "value" : "..."
}
```

19.2 Examples

Each field inside a scope specification JSON object represents a condition that shall be met for a field.

EXAMPLE 1: A query to find all objects belonging to the domain "/cdmi_domains/MyDomain/" is structured as follows:

```
[
    {
        "domainURI" : "== /cdmi_domains/MyDomain/"
    }
]
```

EXAMPLE 2: To query for all objects belonging to the domain "/cdmi_domains/MyDomain/" AND are also located within the container "MyContainer", the scope specification is structured as follows:

```
[
    {
        "parentURI" : "== /MyContainer/",
        "domainURI" : "== /cdmi_domains/MyDomain/"
    }
]
```

EXAMPLE 3: To query for all objects created within a certain time range, the scope specification is structured as follows:

```
[
    {
        "metadata": {
            "cdmi_ctime": [
                ">=2012-01-01T00:00:00",
                "<=2013-01-01T00:00:00"
            ]
        }
    }
]
```

When multiple matching expressions are specified for a given field or metadata item, all matching expression must evaluate true for an object to be considered a query result.

EXAMPLE 4: To query for all objects that belong to the domain "MyDomain" OR are located within the container "MyContainer", the query is structured as follows:

```
[
    {
        "parentURI" : "== /MyContainer/",
    },
    {
        "domainURI" : "== /cdmi_domains/MyDomain/"
    }
]
```

Queries may match on any field within an object that a cloud storage system is capable of returning as a result of an object GET.

EXAMPLE 5: To query metadata items, the metadata object is included as an object within the query request. This query is shown as follows:

```
[
    {
        "metadata" : {
            "colour" : "== blue"
        }
    }
]
```

This approach allows matching against arbitrarily nested metadata structures. When a JSON object is included in the scope specification, matches are performed within that object, and when a JSON array is included in the scope specification, matches are performed within that array. Matching against the contents of arrays of objects is indicated by having an object within the array, as illustrated in Example 5.

EXAMPLE 6: To query all objects with an ACE associated with the user "jdoe":

```
[
    {
        "metadata" : {
            "cdmi_acl" : [
                {
                    "identifier" : "== jdoe"
                }
            ]
        }
    }
]
```

EXAMPLE 7: To query the value of objects, the value field is included within the query request. Values are always represented using base 64 encoding in queries.

```
{
   [
       {
           "value": "== Ymx1ZQ=="
       }
   ]
}
```

Query against the value of objects is optional and is indicated by the presence of the cdmi_query_value capability.

19.3 Query matching expressions

Query matching expressions are structured as “<operator>” or “<operator><sp><constant>”, and are defined in Table 149.

Table 149: Query matching expressions

Matching Expression	Description
“field”:“*”	The exists matching expression tests for the existence of the field. If the field is present, even if empty, the condition shall be considered to be met.
“field”:“!*”	The not exists matching expression tests for the non-existence of the field. If the field is absent, the condition shall be considered to be met.
“field”:“== constant”	The equals matching expression tests for the equality of the value of the field and a specified constant value. The equality test is case sensitive. If the constant value matches the value of the field, the condition shall be considered to be met.
“field”:“#== constant”	The numeric equals matching expression tests for the numeric equality of the value of the field and a specified constant value.
“field”:“!= constant”	The not equals matching expression tests for the non-equality of the value of the field and a specified constant value. The not-equals test is case sensitive. If the constant value does not match the value of the field, the condition shall be considered to be met.
“field”:“#!= constant”	The numeric equals matching expression tests for non-equality of the numeric equality of the value of the field and a specified constant value.
“field”:“> constant”	The greater than matching expression tests if the value of the field is lexicographically greater than a specified constant value. The greater than test is case sensitive. If the constant value is greater than the value of the field, the condition shall be considered to be met.
“field”:“#> constant”	The numeric greater than matching expression tests if the numeric value of the field is greater than a specified constant value.
“field”:“>= constant”	The greater than or equals to matching expression tests if the value of the field is lexicographically greater than or equal to a specified constant value. The greater than or equals to test is case sensitive. If the constant value is greater than or equal to the value of the field, the condition shall be considered to be met.
“field”:“#>= constant”	The numeric greater than or equals to matching expression tests if the numeric value of the field is greater than or equal to a specified constant value.
“field”:“< constant”	The less than operator tests if the value of the field is lexicographically less than a specified constant value. The less than test is case sensitive. If the constant value is less than the value of the field, the condition shall be considered to be met.
“field”:“#< constant”	The numeric less than operator tests if the numeric value of the field is less than a specified constant value.
“field”:“<= constant”	The less than or equals to matching expression tests if the value of the field is lexicographically less than or equal to a specified constant value. The less than or equal test is case sensitive. If the constant value is less than or equal to the value of the field, the condition shall be considered to be met.
“field”:“#<= constant”	The numeric less than or equals to matching expression tests if the numeric value of the field is less than or equal to a specified constant value.
“field”:“starts constant”	The starts with matching expression tests if the field value starts with a specified constant value. If the constant value is equal to the start of the value of the field, the condition shall be considered to be met.
“field”:“!starts constant”	The not starts with matching expression tests if the field value does not start with a specified constant value. If the constant value is not equal to the start of the value of the field, the condition shall be considered to be met.
“field”:“ends constant”	The ends with matching expression tests if the field value ends with a specified constant value. If the constant value is equal to the end of the value of the field, the condition shall be considered to be met.

Continued on next page

Table 149 – continued from previous page

Matching Expression	Description
"field":"!ends constant"	The not ends with matching expression tests if the field value does not end with a specified constant value. If the constant value is not equal to the end of the value of the field, the condition shall be considered to be met.
"field":"contains constant"	The contains matching expression tests if the field value contains a specified constant value. If the constant value is found as a substring within the value of the field, the condition shall be considered to be met. The contains operator is only supported if the cdmi_query_contains capability is present.
"field":"!contains constant"	The not contains matching expression tests if the field value does not contain a specified constant value. If the constant value is not found as a substring within the value of the field, the condition shall be considered to be met. The not contains operator is only supported if the cdmi_query_contains capability is present.
"field":"tag constant"	The tag matching expression tests if the field value contains a specified constant tag value. The leading space character after the "tag" and before the constant value is not included in the comparison. The tag test is not case sensitive. If the constant value is found as a tag substring within the value of the field, the condition shall be considered to be met. Tag substrings start at the beginning of the value or a ",", and end at the next "," or the end of the string. Whitespace before and after "," characters shall be stripped for the purpose of comparisons. Tag matching expressions are only supported if the cdmi_query_tags capability is present.
"field":"!tag constant"	The not tag matching expression tests if the field value does not contain a specified constant tag value. The leading space character after the "!tag" and before the constant value is not included in the comparison. The not tag test is not case sensitive. If the constant value is not found as a tag substring within the value of the field, the condition shall be considered to be met. Tag substrings start at the beginning of the value or a ",", and end at the next "," or the end of the string. Whitespace before and after "," characters shall be stripped for the purpose of comparisons. Tag matching expressions are only supported if the cdmi_query_tags capability is present.
"field":"=~ constant"	The regular expression matching expression tests if the field value matches a specified constant regular expression value. If the regular expression evaluates to true against the value, the condition shall be considered to be met. Regular expression strings shall be processed according to the POSIX Extended Regular Expression (ERE) standard, as specified in IEEE 1003.1-2017 [41]. Regex matching expressions are only supported if the cdmi_query_regex capability is present.
"field":"!~ constant"	The not regular expression matching expression tests if the field value does not match a specified constant regular expression value. If the regular expression evaluates to false against the value, the condition shall be considered to be met. Regular expression strings shall be processed according to the POSIX Extended Regular Expression (ERE) standard, as specified in IEEE 1003.1-2017 [41]. Regex matching expressions are only supported if the "cdmi_query_regex" capability is present.

Numeric constant strings shall be processed according to the JSON number representation described in RFC 4627 [5]. A numeric matching expression shall be considered to be non-matching against a non-numeric field value.

All fields in objects that are not included in the scope specification shall be ignored for the purpose of matching objects.

When a URI is used as the constant for the equals and not equals operators against the parentURI, domainURI, and capabilitiesURI, either a URI by path or URI by object ID may be specified and are considered interchangeable.

19.3.1 Examples

EXAMPLE 1: In a query to find all objects belonging to a specific domain, the following two query scopes are considered identical:

```
[
    {
        "domainURI" : "== /cdmi_domains/MyDomain/"
    }
]
```

and

```
[
    {
        "domainURI" : "== /cdmi_objectid/00007E7F001074C86AD256DA5C67180D/"
    }
]
```

EXAMPLE 2: Likewise, a query to find all objects with a given parent container would have two equivalent forms:

```
[
    {
        "parentURI" : "== /MyContainer/"
    }
]
```

and

```
[
    {
        "parentURI" : "== /cdmi_objectid/00007ED900100E358C3B312DB652C201/"
    }
]
```

If an object ID is used in a query scope in the `objectID` field or the `parentID` field, all object IDs shall be processed such that they are case insensitive.

Clause 20

Results specification

20.1 Overview

CDMI™ provides a standardized mechanism to define subsets of object contents. This mechanism is known as a CDMI results specification. Results specifications are typically used to provide a CDMI client with a way to indicate on what subset of the contents of CDMI objects it intends to retrieve or operate.

Each JSON object within the results specification represents a set of fields that are returned for each matching object.

The results JSON object shall be constructed using the same structure as is used for CDMI objects. To show this, assume the following result from a GET for a data object:

```
HTTP/1.1 200 OK
Content-Type: application/cdmi-object

{
    "objectType" : "application/cdmi-object",
    "objectID" : "00007E7F0010EB9092B29F6CD6AD6824",
    "objectName" : "MyDataObject.txt",
    "parentURI" : "/MyContainer/",
     "parentID" : "00007E7F00102E230ED82694DAA975D2",
    "domainURI" : "/cdmi_domains/MyDomain/",
    "capabilitiesURI" : "/cdmi_capabilities/dataobject/",
    "completionStatus" : "Complete",
    "mimetype" : "text/plain",
    "metadata" : {
        "cdmi_size" : "108263",
            ...
    },
    "valuerange" : "0-108262",
    "value" : "..."
}
```

20.2 Examples

Each field inside a results specification JSON object indicates that the field shall be included in the results.

EXAMPLE 1: The following results specification requests that the `objectID` and `cdmi_size` metadata fields be returned in the results:

```
{
    "cdmi_results_specification" : {
        "objectID" : "",
        "metadata" : {
            "cdmi_size" : ""
        }
    }
}
```

EXAMPLE 2: If an object is matched, the result JSON is enqueued as follows:

```
{
    "objectID" : "00007E7F0010EB9092B29F6CD6AD6824",
    "metadata" : {
        "cdmi_size" : "108263"
    }
}
```

For most common use cases, clients request either the `objectID`, the `objectName` and `parentURI`, or all three fields in the `cdmi_results_specification`. If the `parentURI` or `objectName` is requested, the field shall only be returned for objects existing in a container object.

EXAMPLE 3: To request all metadata items be returned for each matching object, the following `cdmi_results_specification` shall be used:

```
{
    "cdmi_results_specification" : {
        "metadata" : ""
    }
}
```

EXAMPLE 4: To request all fields and all metadata items be returned for each matching object, the following `cdmi_results_specification` shall be used:

```
{
    "cdmi_results_specification" : ""
}
```

The `value` field is always returned in base 64 encoding when included in a query result, where the `valuetransferencoding` field indicates the encoding that should be expected if a GET to read the object is performed.

Clause 21

Notification queues

21.1 Overview

A cloud storage system may optionally implement notification functionality. The implementation of notification is indicated by the presence of the cloud storage system-wide capabilities for notification, and requires support for CDMI™ queues.

Notification queues allow CDMI clients to efficiently discover what changes have occurred to the system. As queue data is persistent, no session state needs to be retained by the client. If different notification queues are used for different clients, then each client operates independently from the others (e.g., a storage management application may use a notification queue to keep its database current without having to do full scans of a container to discover what data objects have been added, modified, or removed).

When a client wishes to receive notifications, it may first check if the system is capable of providing notifications by checking for the presence of the `cdmi_notification` capability in the root container capabilities. If this capability is not present, creating a notification queue shall be successful, but no notifications shall be enqueued into the notification queue.

To create a notification queue, the client creates a regular CDMI queue and adds metadata instructing the storage system to treat the queue as a notification queue. This added metadata also instructs the system about what types of notifications shall be generated and what information shall be included with each notification.

After the notification queue is created, all subsequent matching events after the queue creation time shall result in notification results being enqueued into the queue. CDMI does not mandate any specific ordering of events, and clients must be able to handle events that arrive out of order.

21.2 Metadata

21.2.1 Required metadata

When creating a notification queue, the metadata described in Table 150 shall be provided. Attempts to change metadata in this table shall result in an HTTP status code of `403 Forbidden`. After a notification queue has been created, with the exception of `cdmi_queue_type`, the metadata items in this table may not be changed. `cdmi_queue_type` may only be removed, indicating to the system that the notification queue shall no longer receive notifications and shall be treated as a regular CDMI queue object.

Table 150: Required metadata for a notification queue

Metadata name	Type	Description	Requirement
`cdmi_queue_type`	JSON string	Queue type indicates how the cloud storage system shall manage the queue object. The type of `cdmi_notification_queue` is defined for notification queues.	Mandatory

Continued on next page

Table 150 – continued from previous page

Metadata name	Type	Description	Requirement
cdmi_notification_events	JSON array of JSON strings	The notification events metadata contains a JSON array that indicates which events generate notifications. Defined values are: • cdmi_create_processing - Notifications are generated when a new object is created but is still in the “Processing” completion status. • cdmi_create_complete - Notifications are generated when a new object is created immediately or when a new object in the process of being created transitions from the “Processing” completion status. When an object transitions from “Processing” completion status, the “cdmi_event_result” is the HTTP result code that would have been returned if the create operation was not delayed. • cdmi_read - Notifications are generated when an object is read. • cdmi_modify_processing - Notifications are generated when an existing object is modified but is still in the “Processing” completion status. • cdmi_modify_complete - Notifications are generated when an existing object is modified and is in the “Complete” completion status. This notification is also generated when an existing object being modified transitions from “Processing” to “Complete”. When an object transitions from “Processing” completion status, the “cdmi_event_result” is the HTTP result code that would have been returned if the modify operation was not delayed. • cdmi_rename - Notifications are generated when an object is renamed as part of a move operation. • cdmi_copy - Notifications are generated for the newly created copied object when the copy is completed. • cdmi_reference - Notifications are generated when a reference is created. • cdmi_delete - Notifications are generated when an object is deleted. • cdmi_export - Notifications are generated when a container is exported. • cdmi_snapshot - Notifications are generated when a container snapshot is created. • <implementor-specific events> Clients may include the desired notification event types in the cdmi_notification_events JSON array. If all notifications events are desired, an empty JSON array shall be used.	Mandatory
cdmi_scope_specification	JSON array of JSON objects	The scope specification determines the set of objects on which operations trigger the generation of notifications. If notifications are desired for all objects, include an empty JSON array. See clause 19 for how to construct a scope specification.	Mandatory

Continued on next page

Table 150 – continued from previous page

Metadata name	Type	Description	Requirement
cdmi_results_specification	JSON object	The results specification contains the JSON fields to be returned for each object that matches the notification scope specification. See clause 20 for how to construct a results specification. In addition to the fields defined in clause 20, for notifications, four additional fields are defined: • cdmi_event - Indicates the event as specified in the "cdmi_notification_events" field that triggered the notification; • cdmi_event_result - Indicates the status result of the event that triggered the notification. The status is the same as the status that was returned over the HTTP request, i.e., 200 OK, 404 Not Found, etc.; • cdmi_event_time - Indicates the time of the event that triggered the notification. The time will be formatted in ISO-8601 time (see 5.6 and ISo 8601-1:2019 [32]); and • cdmi_event_user - Indicates the principal (ACL name) of the user that caused the event that triggered the notification. If the system triggered the event, the name will be left as an empty string.	Mandatory

21.2.2 Examples

EXAMPLE 1: The metadata associated with a notification queue is as follows:

```
{
    "metadata" : {
        "cdmi_queue_type" : "cdmi_notification_queue",
        "cdmi_notification_events" : [
            "cdmi_create_complete",
            "cdmi_read",
            "cdmi_modify_complete",
            "cdmi_delete"
        ],
        "cdmi_scope_specification" : [
            {
                "domainURI" : "== /cdmi_domains/MyDomain/",
                "parentURI" : "starts /sandbox",
                "metadata" : {
                    "cdmi_size" : ">+100000"
                }
            }
        ],
        "cdmi_results_specification" : {
            "cdmi_event" : "",
            "cdmi_event_result" : "",
            "cdmi_event_time" : "",
            "objectID" : "",
            "metadata" : {
                "cdmi_size" : ""
            }
        }
    }
}
```

When notification results are stored in a notification queue, each enqueued value shall consist of a JSON object of MIME type "application/json". This JSON object contains the specified values requested in the cdmi_results_specification of the notification queue metadata.

EXAMPLE 2: A notification result JSON object is as follows:

```
{
    "cdmi_event" : "cdmi_read",
    "cdmi_event_result" : "200 OK",
    "cdmi_event_time" : "2010-11-15T13:12:52.342324Z",
    "objectID" : "00007E7F0010EB9092B29F6CD6AD6824",
    "metadata" : {
        "cdmi_size" : "108263"
    }
}
```

Objects shall only be included in the notification results if the user who created the notification queue is able to read the matching object.

If the administrator created the notification queue, then all matching objects that the administrator is allowed to read are included in the results. If user "jdoe" created the notification queue, then only matching objects that "jdoe" is allowed to read are included in the results.

21.2.3 System-created metadata

Table 151 describes the system-created metadata that provides details on the status of the notification queue.

Table 151: Notification status metadata

Metadata name	Type	Description	Requirement
cdmi_notification_status	JSON string	A string indicating the state of the notification queue. Defined values are: • Processing - Indicates that the notification queue is scanning for results; • Halted - Indicates that new notifications will no longer be enqueued; • Current - Indicates that the notification queue contained all notifications that can be found at this time; and • Error - Indicates that the notification queue metadata is not valid, or other errors were encountered that prevented notification messages from being enqueued. Arbitrary vendor-defined text may follow the string "Error". If this metadata item does not exist, then notifications have not yet started being enqueued.	Mandatory

Clause 22

Query queues

22.1 Overview

A cloud storage system may optionally implement metadata and/or full-text query functionality. The implementation of query is indicated by the presence of the cloud storage system-wide capabilities for query and requires support for CDMI™ queues.

Query queues allow CDMI clients to efficiently discover what content matches a given set of metadata query criteria or full-content search criteria. Clients create or update a query queue by specifying metadata that defines the matching criteria (known as the query scope), along with what results should be returned for matching objects (known as the query results). The cloud service shall then perform the query using the content existing at the time the query is being processed, storing the query results in the query queue. As query results are found, they are added to the queue, and when the query is complete, the `cdmi_query_status` metadata of the queue is changed to indicate that the query has completed. Any matching objects created or modified while the query is being performed may or may not be included in the query results (e.g., as a consequence of eventual consistency).

When a client wishes to perform queries, it may first check if the system is capable of providing query functionality by checking for the presence of the cdmi_query capability in the root container capabilities. If this capability is not present, creating a query queue shall be successful, but no query results shall be enqueued into the query queue.

When creating a query queue, the metadata described in Table 152 shall be provided. Attempts to change metadata in this table shall result in an HTTP status code of `403 Forbidden`. After a query queue has been created, with the exception of `cdmi_queue_type`, the metadata items in this table cannot be changed. If the value of `cdmi_queue_type` is changed from "`cdmi_query_queue`", this change indicates to the system that an in-process query shall be stopped, the query queue shall no longer receive query results, and the query queue shall be treated as a regular CDMI queue object. To start a new query with an existing queue, the value of the `cdmi_queue_type` shall be changed back to "`cdmi_query_queue`". This international standard does not define a mechanism to pause a running query or resume a stopped query.

Table 152: Required metadata for a query queue

Metadata name	Type	Description	Requirement
`cdmi_queue_type`	JSON string	The queue type indicates how the cloud storage system shall manage the queue object. The type of "`cdmi_query_queue`" is defined for query queues.	Mandatory
`cdmi_scope_specification`	JSON array of JSON objects	The scope specification determines which objects are included in the query results. This scope specification is similar to a "`WHERE`" clause in SQL-like languages. To query all objects, specify an empty JSON array. See Clause 19 for how to construct a scope specification.	Mandatory
`cdmi_results_specification`	JSON object	The results specification contains the JSON fields to be returned for each object that matches the query. This results specification is similar to a "`SELECT`" clause in SQL-like languages. See Clause 20 for how to construct a results specification.	Mandatory

22.1.1 Examples

EXAMPLE 1: An example of the metadata associated with a query queue is as follows:

```
{
    "metadata" : {
        "cdmi_queue_type" : "cdmi_query_queue",
        "cdmi_scope_specification" : [
            {
                "domainURI" : "== /cdmi_domains/MyDomain/",
                "parentURI" : "starts /sandbox",
                "metadata" : {
                    "cdmi_size" : "#> 100000"
                }
            }
        ],
        "cdmi_results_specification" : {
            "objectID" : "",
            "metadata" : {
                "cdmi_size" : ""
            }
        }
    }
}
```

When results are stored in a query queue, each enqueued value shall consist of a JSON object of MIME type "`application/json`". This JSON object contains the specified values requested in the `cdmi_results_specification` of the query queue metadata.

EXAMPLE 2: An example of a query result JSON object is as follows:

```
{
    "objectID" : "00007E7F0010EB9092B29F6CD6AD6824",
    "metadata" : {
        "cdmi_size" : "108263"
    }
}
```

Table 153 describes the system-created metadata that provides details on the status of the query queue.

Table 153: Query status metadata

Metadata name	Type	Description	Requirement
`cdmi_query_status`	JSON string	When present, this metadata item indicates the state of the query queue. Defined values are: • Processing - Indicates that the query queue is scanning for results; • Halted - Indicates that new query results will no longer be enqueued; • Current - Indicates that the query queue contained all query results that can be found at this time; and • Error - Indicates that the query queue metadata was not valid, or other errors were encountered that prevented all query results from being enqueued. Arbitrary vendor-defined text may follow the string "`Error`".	Mandatory

Objects shall only be included in the query results if the user who created the query queue is able to read the matching objects or metadata.

NOTE: If the administrator created the query queue, then all matching objects that the administrator is allowed to read are included in the results. If user "`jdoe`" created the query queue, then only matching objects that "`jdoe`" is allowed to read are included in the results.

22.2 Extending CDMI query

An implementor of a CDMI server may extend CDMI query by adding vendor-specific matching expressions. When an implementor adds vendor-specific metadata fields, these fields shall be queried using the standard query queue functionality.

An implementor of a CDMI server may extend CDMI query by allowing the creation of vendor-specific query queues with a type other than "`cdmi_query_queue`".

Clause 23

Encrypted objects

23.1 Overview

A cloud storage system may optionally implement additional operations against encrypted objects. Support for these operations are indicated by the presence of the cloud storage system-wide capabilities for encrypted objects.

Encrypted object operations include the ability to encrypt, re-encrypt, and decrypt objects that are already stored in the cloud (in-place), to sign and verify the signature of encrypted objects, and to access and update the plaintext associated with encrypted objects.

The CDMI International Standard does not specify the method by which keys are managed. Key management services are provided by an external key management system (KMS), and the use of the KMIP standard is given as an example of how a CDMI server interacts with an external KMS.

CDMI objects may contain values that are encrypted. Operations against an encrypted CDMI object are only supported if the encrypted object value is a valid CMS or JWE JSON format. The CMS or JWE JSON object shall include an embedded mimetype of the encrypted object. For JWE, the "`cty`" header shall be used for this purpose.

23.2 Encryption operations

23.2.1 State diagram

The state transition diagram for encrypted objects is shown in Fig. 16:

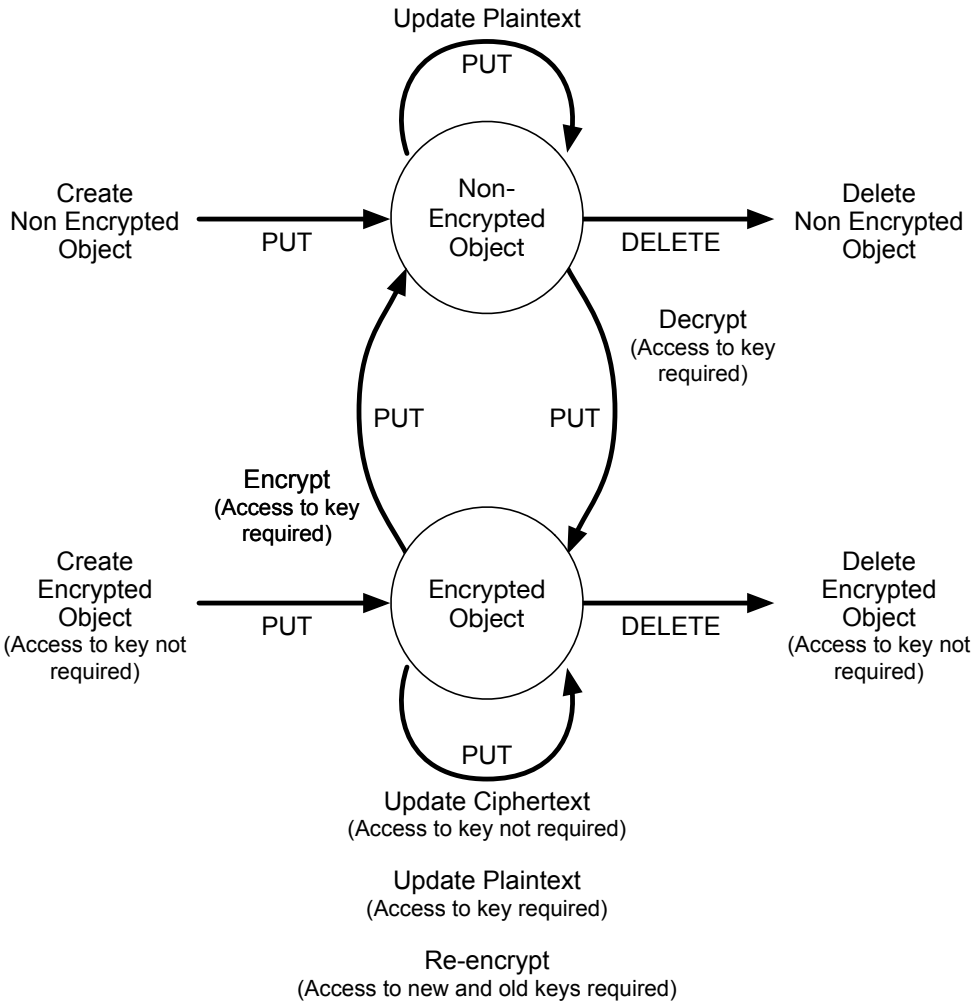


Fig. 16: Encrypted object state transistions

The eight encryption operations are defined in Section 23.2.2 through Section 23.2.9.

23.2.2 Create a new encrypted object

Client-encrypted objects shall be stored to a CDMI server using a standard HTTP or CDMI PUT operation, as described in clauses 7.2 and 8.3. The client shall indicate that an object is encrypted by specifying a mimetype of "`application/cms`" or "`application/jose+json`".

A client may register an encryption key, signing keys and/or verification keys with a Key Management System (KMS), and may indicate the Key IDs in `cdmi_enc_key_id`, `cdmi_enc_value_sign_id`, `cdmi_enc_object_sign_id`, `cdmi_enc_value_verify_id`, and/or `cdmi_enc_object_verify_id` metadata items. This allows the CDMI server to access the keys from the KMS on behalf of a client, when needed.

Creating an encrypted objects on a CDMI server does not require any encryption-specific capabilities to be supported, and is backwards compatible with earlier versions of the CDMI standard. This permits encrypted objects to be stored

and transferred by CDMI servers that do not support encryption-specific functionality.

23.2.3 Delete an encrypted object

Encrypted objects shall be deleted using a standard HTTP or CDMI DELETE operation, as described in clause 7.5 and clause 8.6. Any client with sufficient permissions shall be permitted to delete an encrypted object, regardless of if they can access the decryption keys.

23.2.4 Encrypt an unencrypted object

Existing unencrypted objects shall be encrypted in-place by performing a CDMI PATCH operation, as described in clause 8.5, that changes the object mimetype to "`application/cms`" or "`application/jose+json`" and specifies a `cdmi_enc_key_id` metadata item. The client may also specify a `cdmi_enc_value_sign_id` and/or `cdmi_enc_value_verify_id` metadata item to indicate that the object is to be signed, and to provide signature verification information.

The CDMI Server shall use the client's credentials (which are included in HTTP headers, and are out of scope of this International Standard) to retreive the encryption and signing keys, and encryption and signing algorithm information from the KMS, and shall use the keys to encrypt and sign the value of the object. The mimetype of the encrypted value is stored in the CMS wrapper, or in a "`cty`" field of the JWE JSON.

23.2.5 Decrypt an encrypted object

Existing encrypted objects shall be decrypted in-place by performing a CDMI PATCH operation, as described in clause 8.5, that changes the object mimetype from "`application/cms`" or "`application/jose+json`" to the original mimetype as specified in the CMS wrapper, or in the "`cty`" field of the JWE JSON. Specifying any other fields or metadata shall return a ``400 Bad Request`` result code.

The CDMI Server shall use the client's credentials (which are included in HTTP headers, and are out of scope of this International Standard) to retreive the encryption, signing and verification keys, and encryption, signing and verification algorithm information from the KMS, and shall use the keys to decrypt and verify the encrypted value and user metadata included in the object.

23.2.6 Re-encrypt an encrypted object

Existing encrypted objects shall be re-encrypted in-place by performing a CDMI PATCH operation, as described in clause 8.5, that retains the object mimetype of "`application/cms`" or "`application/jose+json`", or changes the object mimetype from "`application/cms`" to "`application/jose+json`", or vice-versa. The client shall also specify a new `cdmi_enc_key_id`, `cdmi_enc_value_sign_id` and/or `cdmi_enc_value_verify_id` metadata item to indicate the new key(s) to be used. Specifying any other fields or metadata shall return a `400 Bad Request` result code.

The CDMI Server shall use the client's credentials (which are included in HTTP headers, and are out of scope of this International Standard) to retreive both the original encryption and signing keys using the original metadata values, and the new encryption and signing keys using the new metadata values from the KMS, and shall use these keys to decrypt, verify, encrypt and sign the value of the object, as needed.

If an encrypted object does not have an existing `cdmi_enc_key_id` metadata item, does not have a "`kid`" header, and no keys are associated with the Object ID, the specified metadata shall be added to the object, and no re-encryption operation shall be performed.

23.2.7 Access ciphertext of an encrypted object

The ciphertext content of an encrypted object shall be read by performing an HTTP GET operation, as described in clause 6.3, with an Accept header value of "`application/cms`" or "`application/jose+json`", depending on the mimetype of the encrypted object.

The ciphertext content of an encrypted object shall also be read by performing a CDMI GET operation, as described in clause 8.4.

23.2.8 Access plaintext of an encrypted object

The plaintext value of an encrypted object shall be read by performing an HTTP GET operation, as described in clause 6.3, with an Accept header value other than "`application/cms`" or "`application/jose+json`", typically "`*/*`". Object plaintext cannot be transparently accessed using a CDMI GET.

The CDMI Server shall use the client's credentials (which are included in HTTP headers, and are out of scope of this International Standard) to retreive the encryption, signing and verification keys, and encryption, signing and verification algorithm information from the KMS, and shall use the keys to decrypt and verify the encrypted value included in the object.

When an encrypted object is decrypted for access, the plaintext shall not be retained or cached by the CDMI server.

23.2.9 Update plaintext of an encrypted object

The plaintext value of an encrypted object shall be modified by performing an HTTP PATCH operation, as described in clause 6.4, with an Content-Type header value other than "`application/cms`" or "`application/jose+json`", typically "`*/*`"., depending on the mimetype of the encrypted object. Object plaintext cannot be transparently modified using a CDMI GET.

The CDMI Server shall use the client's credentials (which are included in HTTP headers, and are out of scope of this International Standard) to retreive the encryption, signing and verification keys, and encryption, signing and verification algorithm information from the KMS, and shall use the keys to decrypt and verify the encrypted value, update the value, and re-encrypt/re-sign the updated value.

When an encrypted object is decrypted for update, the plaintext shall not be retained or cached by the CDMI server.

23.2.10 Other CDMI operations

Other operations specifed by this International Standard (such as copying, serializing, querying, etc.) treat an encrypted value the same way as a non-encrypted value.

23.3 Example uses of encrypted objects

Encrypted objects can be used with CDMI systems in the following ways:

- **Passthrough** – A client may store an encrypted object in any format in a CDMI server, with the ciphertext being accessible to the server and to other authorized clients. No access to the plaintext is provided. Passthrough use is compatible with all CDMI systems and is useful when the clients manage all security-related operations and want to protect against potentially untrustworthy clouds.
- **Server-side encryption and signing** – A client may instruct a CDMI server that supports encrypted object operations to take an existing CDMI object and encrypt or encrypt and sign it in place into CMS or JWE JSON representation, where the value of the object is persistently stored from that point on in an encrypted format. Server-side encryption and signing is useful when clients trust the CDMI server and want to increase object security without having to re-upload the data.
- **Server-side decryption** – A client may instruct a CDMI server that supports encrypted object operations to take an existing CDMI object and decrypt it in place from a CMS or JWE JSON representation, where the value of the object is persistently stored from that point on in a decrypted format. Server-side decryption is useful when a client trusts the CDMI server and wants to decrease object security without having to re-upload the data.
- **Client access decryption** – A CDMI server may automatically attempt to decrypt an encrypted object when accessed via HTTP. Client access decryption is useful to provide transparent access to authorized HTTP clients without requiring modifications to the HTTP clients.
- **Cloud access decryption** – A CDMI server may automatically decrypt encrypted objects when it has access to the decryption keys. Cloud access decryption is useful for cloud-resident data processing performed by the CDMI server, such as virus scanning, query, and analytics.
- **Signature verification** – A CDMI server can automatically verify signatures that are attached to encrypted objects that include a signature. Signature verification is useful for detecting corruption or alteration before delivering data to a client.

23.4 KMS integration

The encryption key is obtained from the KMS using a unique identifier that is stored in the `cdmi_enc_key_id` metadata item associated with the encrypted object. If this metadata item is not present, the CDMI object ID shall be used to locate the key.

When a client requests that an operation be performed that requires accessing the key for the object, the CDMI server evaluates the credentials provided by the client to determine if the client is authorized to perform the requested operation. If the operation is permitted, the CDMI server retrieves the key from the KMS to complete the requested operation. To retrieve the key, the client may be required to provide additional information in the HTTP request that the CDMI server can then use to authenticate to the KMS.

The CDMI International Standard does not specify the mechanism by which the CDMI server communicates with the KMS. In this International Standard, the KMIP protocol is used as an example. CMS and JWE strings for algorithms, key lengths, etc., need to be mapped to the strings used by the KMS (see KMIP clause 9.1.3.2.7).

All keys are created and managed externally to the CDMI server, typically by the client or a system operating on behalf of the client. As a consequence, the CDMI server requires read-only access to the KMS. The CDMI server shall not cache keys.

23.5 CMS format

Any valid CMS-formatted data may be stored to a CDMI server. However, encrypted object operations are only defined for the following subset of valid CMS-formatted data.

For encryption operations, the CDMI server shall support the following:

- `EnvelopedData`
- `EncryptedContentInfo`
- `contentEncryptionAlgorithm` value listed in the `cdmi_cms_encryption` capability of that CDMI server

For signature operations, the CDMI server shall support the following:

- `AuthenticatedData`
- `SignedData`
- `digestAlgorithms` value listed in the `cdmi_cms_digest` capability of that CDMI server
- `SignerInfo`
- `signatureAlgorithm` value listed in the `cdmi_cms_signature` capability of that CDMI server

The following CMS-formatted data may be ignored: `recipientInfos`

23.6 JOSE format

Any valid JWE-formatted data may be stored to a CDMI server. However, encrypted object operations are only defined for a small subset of valid JWE-formatted data.

For encryption operations, the CDMI server shall support the following:

- JWE with Direct Encryption (Symmetric Key from KMS)
- JWE with Key Encryption (Public Key from KMS)

For signature operations, the CDMI server shall support the following:

- JWS RSA (Private Key from KMS)
- JWS ECDSA (Private Key from KMS)
- JWS HMAC-SHA2 (Symmetric Key from KMS)

The following JOSE-formatted data may be ignored:

- Multiple recipients, and
- Multiple signatures.

23.7 Signature/digest verification

If a signature is present as part of the CMS or JWE JSON value, the CDMI server shall verify that the signature of the value is valid before allowing plaintext access or modification.

If a whole-object signature is present, the CDMI server shall verify that the signature contained in the `cdmi_enc_signature` metadata item is valid before allowing any read operations for the object. Write operations are permitted for an object with an invalid or unverifiable whole-object signature.

When present, a whole-object signature shall be attached as a "`cdmi_enc_signature`" metadata item in JWS compact format, with the second field (the JWS payload field) replaced with an empty string as described in Appendix F of RFC 7515 [17].

For signature creation and verification, payload field shall be computed using the following process:

1. Create a serialized representation of the CDMI object, as described in clause 15
2. Remove the following metadata items, if present:
    - `cdmi_atime`
    - `cdmi_acount`
    - `cdmi_enc_signature`
    - Any `*_provided` metadata items
3. Sort all JSON objects in the serialized CDMI object according to the following rules:
    - Within each JSON object, name/value pair entries shall be sorted lexicographically by name
    - Within each JSON array, the initial order shall be preserved
4. Remove all JSON whitespace
5. Base64 URL encode, according to the JWS RFC 7515 [17]

23.8 Error handling

If a decryption or signature validation operation is requested against a CDMI object containing an invalid CMS or JWE JSON representation, an HTTP status code of `500 Internal Error` shall be returned to the client.

If a decryption or signature validation operation is requested against a CDMI object containing a valid CMS or JWE JSON representation that uses an unsupported algorithm or feature, an HTTP status code of `501 Not Implemented` shall be returned to the client.

If a decryption or signature validation operation is requested against a CDMI object containing a valid CMS or JWE JSON representation, but the required keys are temporarily unavailable given the credentials presented, an HTTP status code of `408 Request Timeout` shall be returned to the client.

If a decryption or signature validation operation is requested against a CDMI object containing a valid CMS or JWE JSON representation, but the required keys are unavailable given the credentials presented, an HTTP status code of `401 Unauthorized` shall be returned to the client.

If a decryption or signature validation operation is requested against a CDMI object containing a valid CMS or JWE JSON representation, valid keys are available, and signature verification fails, an HTTP status code of `403 Forbidden` shall be returned to the client.

Clause 24

Delegated access control

24.1 Overview

CDMI access control is based around Access Control Lists (ACLs) that are stored as object metadata. When a client requests to perform an operation against a CDMI object, the CDMI server shall validate the client's identity and credentials against the object ACL to determine if the operation is allowed. This request assumes that the CDMI server is trusted and capable of making these access control decisions.

Fig. 17 illustrates an ACL-based access control request:

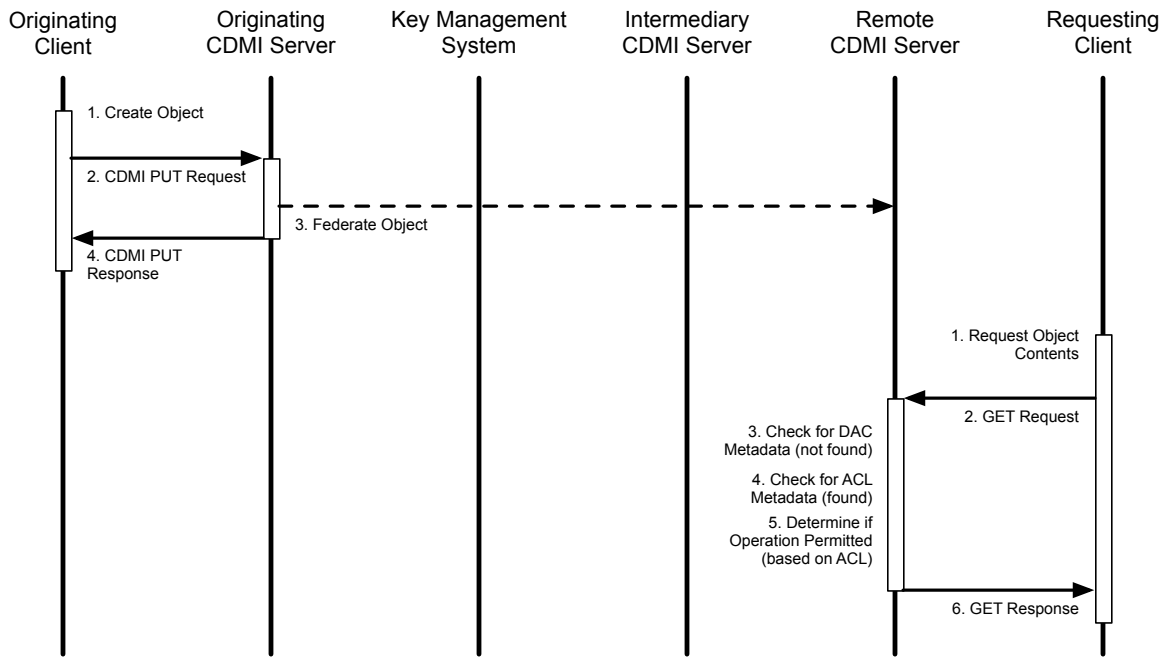


Fig. 17: Non-delegated (ACL-based) access control data flow

When an access control decision needs to be made by a third party (such as by the originating CDMI server in Fig. 17), access control is delegated. When `cdmi_dac_uri` and `cdmi_dac_certificate` object metadata is present, as specified in clause 16.2, Delegated Access Control (DAC) shall be used.

An example of an object with DAC metadata is shown below:

```
{
  "objectType": "application/cdmi-object",
  "objectName": "MyObject.txt",
  "capabilitiesURI": "/cdmi_capabilities/dataobject/",
  "objectID": "0000000800182ADB37303732323136662D343564622D3462",
  "mimetype": "text/plain",
  "metadata": {
    "cdmi_size": "33",
    "cdmi_ctime": "2017-04-05T11:01:25",
    "cdmi_atime": "2017-04-05T11:44:28",
    "cdmi_dac_uri": "https://cloud.example.com/dac/",
    "cdmi_dac_certificate": {
      "kty": "EC",
      "x": "goqhRgM4hyEh1p-fD1oU15QAgdKXsBZTQ_0B-IgSz6M",
      "y": "cd8RTm8uLTGblIzioAzv8dzIkM85c08o23eksJrDt2Y",
      "crv": "P-256"
    }
  },
  "valueTransferEncoding": "utf-8",
  "valueRange": "33",
  "value": "This is an unencrypted text file."
}
```

The process by which objects are federated between systems is outside the scope of access control delegation and involves how objects are replicated, synchronized, mirrored, or migrated between CDMI servers. These processes are typically under the control of policies or external policy management systems. Federation is typically performed by third-party systems that use CDMI features including notification, serialization, and the preservation of globally unique object identifiers, which forms the basis for client-transparent interoperability.

24.2 Delegated access control (DAC)

A cloud storage system may implement support for DAC, which is indicated by the presence of the cdmi_dac system-wide capability.

DAC enables requests for operations against an object to be allowed or denied by a third-party DAC provider, in addition to ACL access control. When required by object metadata, DAC access control verification shall be performed after ACL evaluation, but before ACL enforcement, as the DAC provider may overrule local ACL evaluation results. When an encrypted object is accessed, the DAC provider may provide the decryption key. The decryption key enables access to encrypted objects, even if the CDMI server cannot access the keys directly.

Clients often have different degrees to which they trust the CDMI server with which they are interacting. Table 154 describes the four ways that DAC shall interact with stored objects.

Table 154: Access modes for DAC

Mode of access	Degree of trust
Client-side decryption	CDMI server is not trusted with keys or to make delegated access control decisions. 1. Client requests encrypted object from CDMI Server 2. Client receives ciphertext from the CDMI Server 3. Client is responsible for getting decryption keys out of band 4. Client verifies signatures (if present) 5. Client verifies correct object 6. Client decrypts object This mode of access does not use any functionality indicated by the `cdmi_dac` capability and is supported by all CDMI servers.
Client-side decryption with DAC	CDMI server is not trusted with keys and is used to establish an opaque channel of communication between the client and the DAC provider for key delivery. 1. Client requests encrypted object from the CDMI Server, and includes custom DAC headers specifying information required for secure delivery of decryption key 2. Client receives ciphertext from the CDMI Server, along with custom DAC header from the DAC provider for the decryption key 3. Client is extracts decryption key from DAC provider headers 4. Client verifies signatures (if present) 5. Client verifies correct object 6. Client decrypts object This mode of access requires the `cdmi_dac` capability but does not require encrypted object support. In this mode, data is exchanged between the client and the DAC provider using one or more “CDMI-DAC-” headers, as described in clause 24.4.
Direct Client DAC	CDMI server is not trusted with keys, and client establishes channel of communication between the client and the DAC provider for key delivery. 1. Client requests encrypted object from CDMI Server 2. Client receives ciphertext from CDMI Server 3. Client sends DAC request directly to DAC Provider 4. Client receive DAC response directly from DAC Provider 5. Client verifies signatures (if present) 6. Client verifies correct object 7. Client decrypts object This mode of access requires the `cdmi_dac` capability but does not require encrypted object support.

Continued on next page

Table 154 – continued from previous page

Mode of access	Degree of trust
Server-side decryption with DAC	CDMI server is trusted with keys and to delegate access control decisions. DAC message exchange is used to get the decryption keys to decrypt the contents of the object, and keys are not revealed to the client. 1. Client requests encrypted object from CDMI Server 2. CDMI server contacts the DAC Provider to determine access control decision and gets decryption keys, where the keys are not revealed to the client. 3. CDMI server verifies signatures (if present) 4. CDMI server verifies correct object 5. CDMI server decrypts object 6. Client receives plaintext This mode of access requires DAC and encrypted object support.
Plaintext objects with DAC	CDMI server is trusted with plaintext and to not bypass delegated access control decisions. 1. Client requests non-encrypted object from CDMI Server 2. CDMI server contacts DAC provider to determine access control decision 3. CDMI server verifies signatures (if present) 4. CDMI server verifies correct object 5. Client receives plaintext This mode of access requires DAC support.

The `cdmi_dac_uri` metadata item indicates where delegated access control requests shall be submitted, and the `cdmi_dac_certificate` metadata item indicates how securely communication with the delegated access control provider shall be established. Both of these metadata items shall be present for DAC to be enabled for a given object.

DAC requests are submitted to a DAC provider using two typical methods:

- **Direct** - The DAC request shall be submitted directly to the absolute URI specified in the `cdmi_dac_uri metadata` item. This approach requires the host specified in the URI to be accessible from the CDMI server, and for the CDMI server making the request to have sufficient permissions to PUT the DAC request to that location.
- **Indirect** - The DAC request shall be sent to the DAC provider using an indirect route. Indirect routing is useful when the `cdmi_dac_uri` does not specify a host. An example of indirect routing is when the `cdmi_dac_uri` contains a mailto URI; the Internet mail system is then responsible for delivering the DAC request.

  In other cases, the certificate included with the DAC request (taken from the `cdmi_dac_certificate` metadata) may be used by intermediary CDMI servers to determine the further routing of the DAC request. For example, DAC requests using a E.U.-issued certificate can be forwarded to a different intermediary CDMI server to those requests using a U.S.-issued certificate. How certificate fields are used to determine routing is not defined in this International Standard.

Both direct and indirect routing may be synchronous or asynchronous. If a DAC response is not received within the CDMI server or client timeout windows, the client request may time out; however a subsequent request may be processed locally if the DAC response allows response caching. When the CDMI server times out while waiting for a DAC response, it shall return an HTTP status code of `504 Gateway Timeout`.

24.3 Delegated access control message exchange

When a client requests to access or modify an object containing DAC metadata on a CDMI server that supports DAC, the CDMI server shall create and send a DAC request as specified in clause 24.5. Upon receiving a DAC response as specified in clause 24.7, the CDMI server shall allow or deny the operation based on the contents of the response.

Fig. 18 provides an example of access control delegation for a non-encrypted object. The black solid lines show indirect routing, and gray dashed lines show direct routing.

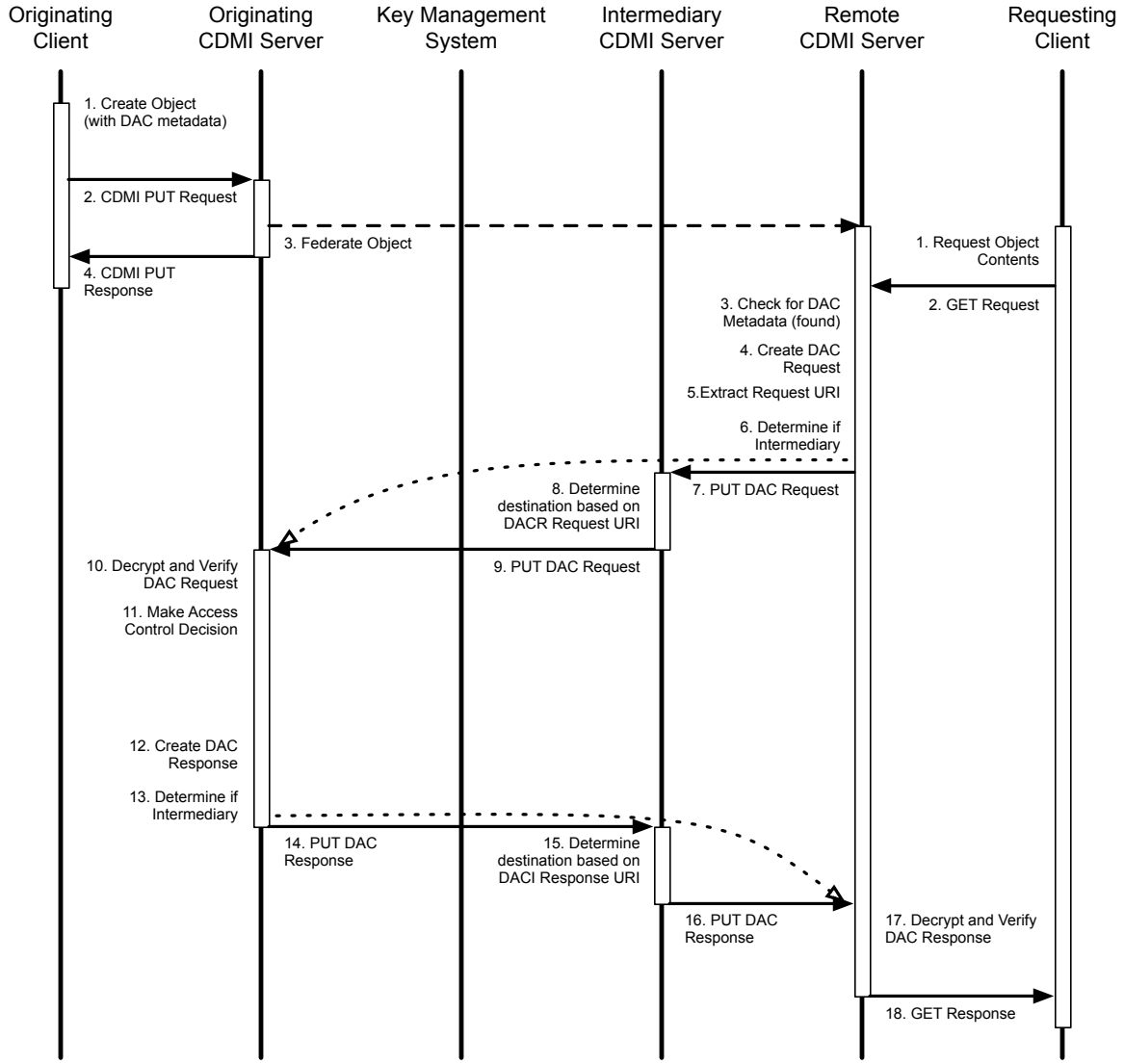


Fig. 18: Delegated access control data flow example for non-encrypted object

For non-encrypted objects, an originating client indicates that DAC is requested by including the DAC metadata items. It is important to emphasize that for non-encrypted objects, DAC cannot be guaranteed to be enforced, as when an object with DAC metadata is accessed from a CDMI server that does not support DAC; only ACL-based access control shall be evaluated.

Fig. 19 provides a second example of access control delegation for an encrypted object. The black solid lines show indirect routing, and gray dashed lines show direct routing.

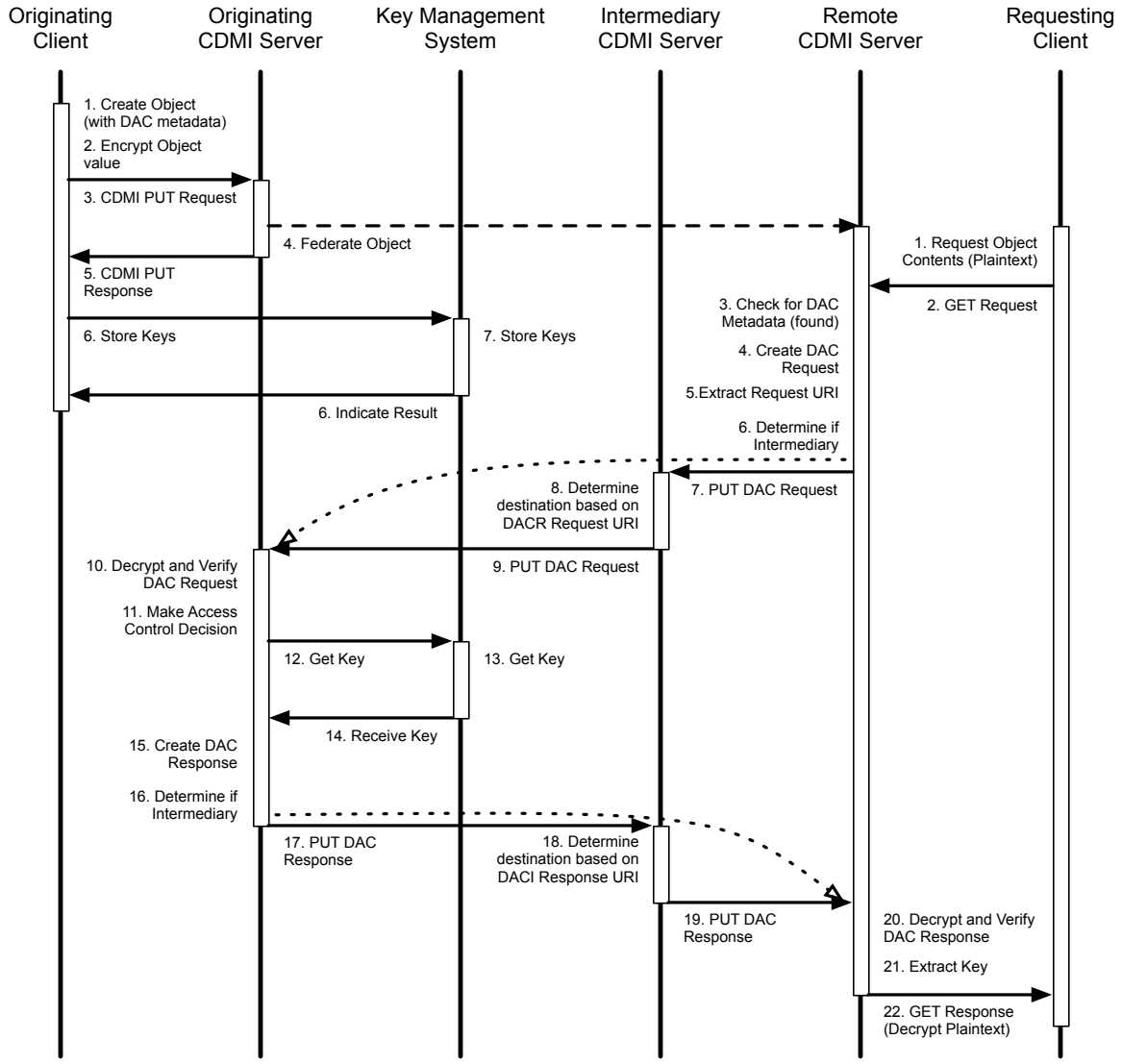


Fig. 19: Delegated access control data flow example for encrypted object

For encrypted objects, as access to the decryption keys are provided in the DAC response, the plaintext is inaccessible unless the CDMI server supports DAC.

When the DAC provider processes the DAC request, if the operation is allowed and the key is requested by the CDMI server, the object key, if present, shall be obtained and sent back as part of the DAC response. Upon receiving the DAC response, the CDMI server shall extract the key to perform the client operation.

24.4 Client header passthrough

The Delegated Access Control extension provides facilities to allow client-provided HTTP request headers to be passed through to the DAC provider, and for the DAC provider to pass HTTP response headers back to the client. These headers are identified by the "`CDMI-DAC-`" prefix.

The contents and full names of these headers are not defined in this International Standard. However, it is anticipated that these headers shall be used to allow the client to provide additional information that may be required for the access control decision-making process, for audit purposes, or for secure key exchange.

For example, when an operation is allowed by a DAC provider, the object key may be encrypted using the public key from a client-provided certificate (verified by the DAC provider), which is included in a "`CDMI-DAC-`" request header, with the encrypted object key being sent back to the client in a "`CDMI-DAC-`" response header. In this scenario, the CDMI server cannot decrypt the ciphertext but can securely pass on the encrypted object key to the client. The client can then use its private key to decrypt the response header to get the object key, which can then be used to decrypt the object.

24.5 DAC request

When a CDMI server that supports DAC needs to contact the DAC provider as specified in the DAC metadata, it shall construct a DAC request, as specified in Table 155.

Table 155: DAC request

Field name	Type	Description	Requirement
dac_request_version	JSON string	Indicates the version of the DAC request. This field shall be set to the value "1".	Mandatory
dac_request_id	JSON string	Contains a system-specified identifier that is used to match up the corresponding DAC response. This identifier shall be unique within the window that multiple DAC responses may be received.	Mandatory
server_identity	JSON object	A JSON object, containing a JWE JWK which shall include a public key that is used to submit a DAC response, and should contains a X.509 certificate or certificate chain used to verify the identity of the CDMI server that is generating the DAC request. This ensures that only the CDMI Server that generated the DAC request can read the DAC response.	Mandatory
client_identity	JSON object	A JSON object containing the following JSON entities: JSON String, "acl_name", containing the ACL name of the client requesting the operation. JSON Array, "acl_group", containing the ACL group(s) of the client requesting the operation.	Optional
acl_effective_mask	JSON string	A text or hexadecimal string representation of the ACE mask determined by ACL evaluation for the requested operation, as defined in 17.2.6.	Mandatory
client_headers	JSON object	A JSON object containing a JSON string for each HTTP header in the operation request that starts with "CDMI-DAC-", where the JSON string name is the header name, and the JSON string value is the header value. These headers can be used for tunneling information from the client to the DAC provider.	Mandatory
cdmi_objectID	JSON string	Contains the object ID of the object the operation is performed against.	Mandatory
cdmi_enc_key_id	JSON string	Contains the encryption key identifier (for example, a KMIP identifier) for the symmetric key that is used to encrypt and decrypt the object, which is used to indicate that the CDMI server is requesting the encryption key.	Optional
cdmi_operation	JSON string	Contains a string indicating which operation is being requested to be performed against the object. The following operations are defined: • "cdmi_read" • "cdmi_modify" • "cdmi_delete"	Mandatory
dac_response_uri	JSON string	An optional URI that specifies where to send the DAC response. This URI is required for asynchronous DAC requests, such as when sent via email URIs. If this field is omitted, the DAC response shall be based on the context of the request, for example, as a message body returned for the request PUT when using HTTPS, or an email reply when using a mailto URI.	Optional

3877 An example of a DAC request is shown below:

```
{
   "dac_request_version": "1",
   "dac_request_id": "037130fa-da72-44f0-8a31-62073263ac95",
   "server_identity": {
      "kty": "EC",
      "x": "joyfi05KEI3hcOhJeOfny_TWsZ9FFS1zUydFQhm3G78",
      "y": "Nsk3jX1ph0FH8APR2k0XSu6pDZYyF7f_Okplf7hZ_8k",
      "crv": "P-256"
   },
   "client_identity": {
      "acl_name": "anonymous",
      "acl_group": ["users"]
   },
   "acl_effective_mask": "READ_ALL",
   "client_headers": {
      "cdmi-dac-header1": "This is a test header"
   },
   "cdmi_objectID": "0000000800182ADB37303732323136662D343564622D3462",
   "cdmi_operation": "cdmi_read"
}
```

24.6 Packaged DAC request

A JSON DAC request shall be encrypted in JWE format, where the recipient is the public key of the DAC provider certificate (as specified in the DAC object `cdmi_dac_certificate` metadata), and is JWS signed using the private key of the CDMI server that corresponds to the server identity certificate included in the DAC request. The certificate of the DAC provider from the object is then attached as specified in Table 156.

Table 156: Packaged DAC request

Field name	Type	Description	Requirement
`dac_request`	JSON object	JOSE encrypted and signed request	Mandatory
`dac_request_dest_certificate`	JSON object	The cdmi_dac_certificate metadata value, which is used to indicate where the DAC request is being sent via indirect routing.	Mandatory
`dac_request_dest_uri`	JSON string	The `cdmi_dac_uri` metadata value, which is used to indicate where the DAC request is being sent via direct routing, or used to indicate the first location when being sent via indirect routing.	Mandatory

An example of a packaged DAC request is shown below[1]:

```
{
   "dac_request": {
      "protected":
         "eyJqd2siOiJ7XCJrdHlcIjpcIkVDXCIsXCJ4XCI6XCJqb3lmaTA1S0VJM2hjT2hK
          ZU9mbnlfVFdzWjlGRlMxelV5ZEZRaG0zRzc4XCIsXCJ5XCI6XCJOc2szalgxcGgw
          Rkg4QVBSMmswWFN1NnBEWll5RjdmX09rcGxmN2haXzhrXCIsXCJjcnZcIjpcIlAt
          MjU2XCJ9IiwiYWxnIjoiRVMyNTYifQ",
      "payload":
         "eyJwcm90ZWN0ZWQiOiJleUpoYkdjaU9pSkZRMFJJTFVWVElpd2laVzVqSWpvaVFU
          STFOa2REVFNJc0ltVndheUk2ZXlKcmRIa2lPaUpGUXlJc0luZ2lPaUpuUkZOek1F
          cFRXbU5VVVRsWGVGWXRiRXhSVTJ4elFsY3lXazFvTm1kb1JrcDJTVmt4TWt4d1dW
          TlJJaXdpZVNJNklsTkZNV1pXWkVkalZtdGtPVVZCVmpaVGMyeE9NVzQyUkdsdlpV
          dHVZV3BLWmpsdWVFOVljRlpoYmtFaUxDSmpjbllpT2lKUUxUSTFOaUo5ZlEiLCJl
          bmNyeXB0ZWRfa2V5IjoiIiwiaXYiOiJLRDlGRlBOcFh2cWNIYTdIIiwiY2lwaGVy
          dGV4dCI6Im42NlpmUzBXRmhjN3ZzT3Rnc1o5SXJtWU5paDI4RDVzT1psTk96dEdO
          TW5hakFRSGZTMGozcUhrMUxPME9IbFBYMnVfYXVWcVN2aDF2Z1IxSFlnOEl3TmFq
          TFZfS29ZMndGXzlkaDRtWFJlVXA4R1hpbm05MFE0ZWZmY1BLRm1IcEo0dE94TTVS
          VjlLN2VvdWNxSkxzczJKbHc1ZUJhOVQ5WjFyS1pvQmIxVURSLVVmRW91QlNZRFA3
          NUl1SEFRSWU4UW5qOW04QjFHb18tNTFPNndKb2d6cHh5Ulhpd3g2SWdoYlhSYmNX
          MWQ5bVRtZkR3UFBoSE4zTUp1UGUxbVBpelNLWnJ3NWNQM21NZmhKWmNoT3gyZkZt
          Q3NMME5zSkphQWo3WEs0elFiMGVBd0RSS1BzeTJ6MnZCZzFQT1lhUHppOVphNjRK
          RHgyZ3hWRTA2Y0xERGx3TXY4dW9CbFU1TVdyZF9YRGdScUZsSFl1T19aZEtxQkRp
          MVQ1SW5HeDc2YzdCcmVObzFIbnVqV200M0FsanpPRmIyTHBhdU5PQnlETl9oVXFi
          WGRISTZOWnZBUDU0MzVteHZDRi1SYUpMZGxFUENNeGhneXNFdy1oRGxoQmtFYUpf
          U0JtZUZtem5ITGFkZUNDYzI3cWNuOUlZVlZBMHZMZVY2N2xzbnZMY3VyOHI0OFlt
          SXRmZGNZbFVOLTh2c0xhSlZzbHhMSzc0VjdjdWNhbFNubWJvYktWTVV6TnZuU29K
          NHpldXBYZzItbl92WnIwbkZlSUFWelIxZmJvUVA0c1F4bXNSUWJNY2d4bmpSM21E
          eTJsQzY5dFN1TDJGYm1qUnZiYWM3XzFRa01CIiwidGFnIjoiNWlRcGVTdTlfb00y
          X2UtSTM3NjJpQSJ9",
      "signature":
         "rGz9Cku3csTIJ_p3qmHzUrPSLb1ZSD3ZlfaJDw0F-dNmJs6sgzizFC_jf5VgDVuo
          GT-wH2b2zVuP_O1HDcKPDQ"
   },
   "dac_request_dest_certificate": {
      "kty": "EC",
      "x": "goqhRgM4hyEh1p-fD1oU15QAgdKXsBZTQ_0B-IgSz6M",
      "y": "cd8RTm8uLTGblIzioAzv8dzIkM85c08o23eksJrDt2Y",
      "crv": "P-256"
   },
   "dac_request_dest_uri": "https://cloud.example.com/dac/"
}
```

Once created, the packaged DAC request shall be submitted using the DAC request URI specified in the DAC ob-

[1] Decrypt with `"d": "NnU0IEyV4JSyLoKwIzKN1FAxDvL6qqawAHlPkpwBMSY"`.

ject metadata, for example, as an HTTP PUT operation of type "`application/json`", or via an SMTP email. The `dac_request_dest_certificate` and `dac_request_dest_uri` may be used to route the request through intermediary hops, as needed.

24.7 DAC response

When a DAC provider receives a DAC request, it shall decrypt the request using its private key, verify the signature of the CDMI server, and shall evaluate the request. Based on the information provided, the DAC provider shall allow or deny operations by modifying or replacing the ACL mask that was initially determined by the CDMI server.

To indicate the result of the DAC request to the requesting CDMI server, the DAC provider shall construct a DAC response, as specified in Table 157.

Table 157: DAC response

Field name	Type	Description	Requirement
dac_response_version	JSON string	Indicates the version of the DAC response. This field shall be set to the value "1".	Mandatory
dac_response_id	JSON string	Contains the system-specified identifier specified in the corresponding dac_request_id.	Mandatory
dac_applied_mask	JSON string	A text or hexadecimal string representation of the ACE mask that shall be used, as defined in 17.2.6.	Mandatory
dac_object_key	JSON object	The key for the object in JWK format (See RFC 7517 [16]). This key is only disclosed when cdmi_enc_key_id is included in the DAC request and the DAC provider allows access.	Optional
dac_response_headers	JSON object	A series of headers that start with "CDMI-DAC-" to be returned to the client. These headers can be used to pass information from the DAC provider back to the client.	Optional
dac_key_cache_expiry	JSON string	The complete date/time when the object key is no longer to be cached, specified in ISO 8601 date/time format. If this field is not included, the key shall not be cached.	Optional
dac_response_cache_expiry	JSON string	The complete date/time when the DAC response is no longer to be cached, specified in ISO 8601 date/time format. If this field is not included, the response shall not be cached.	Optional
dac_redirect_objectID	JSON string	Indicates an alternate CDMI Object ID used to access the requested object. If present, the CDMI server shall send an HTTP Redirect to the client.	Optional
dac_audit_uri	JSON string	Indicates a URI to a CDMI queue where audit logging messages associated with the operations shall be submitted. When present, audit logging messages shall be generated for receiving the response, performing the operation, and determining when to purge the key. The format of these audit messages is not defined by this International Standard.	Optional

An example of a DAC response is shown below:

```
{
   "dac_response_version": "1",
   "dac_response_id": "037130fa-da72-44f0-8a31-62073263ac95",
   "dac_applied_mask": "ALL_PERMS",
   "dac_response_headers": {
      "CDMI-DAC-AuthInfo": "No key requested."
   },
   "dac_response_cache_expiry": "2017-04-06T15:06:01.554Z"
}
```

24.8 Packaged DAC response

The above JSON (DAC response) shall be encrypted in JWE format where the recipient is the public key of the CDMI server certificate (as specified in the DAC request), and is JWS-signed using the private key of the DAC provider that corresponds to the DAC provider identity certificate associated with the object (`cdmi_dac_certificate`), or with a different signing, included in a jku/jwk/x5u or x5c JOSE header to allow retrieval of the public signing verification key.

The certificate of the CDMI server is then attached as specified in Table 158.

Table 158: Packaged DAC response

Field name	Type	Description	Requirement
`dac_response`	JSON object	JOSE encrypted and signed response	Mandatory
`dac_response_dest_certificate`	JSON object	The contents of the DAC request `server_identity` field.	Mandatory
`dac_response_dest_uri`	JSON string	The contents of the DAC request `dac_response_uri` field, if present	Optional

An example of a packaged DAC response is shown below[2]:

```
{
   "dac_response": {
      "protected":
         "eyJqd2siOiJ7XCJrdHlcIjpcIkVDXCIsXCJ4XCI6XCJnb3FoUmdNNGh5RWgxcC1m
          RDFvVTE1UUFnZEtYc0JaVFFfMEItSWdTejZNXCIsXCJ5XCI6XCJjZDhSVG04dUxU
          R2JsSXppb0F6djhkeklrTTg1YzA4bzIzZWtzSnJEdDJZXCIsXCJjcnZcIjpcIlAt
          MjU2XCJ9IiwiYWxnIjoiRVMyNTYifQ",
      "payload":
         "eyJwcm90ZWN0ZWQiOiJleUpoYkdjaU9pSkZRMFJUTFVWVElpd2laVzVqSWpvaVFU
          STFOa2REVFNJc0ltVndheUk2ZXlKcmRIa2lPaUpGUXlJc0luZ2lPaUpNVUVReWRX
          WmlkMUpmT0hoU2FWRlRNMWN3YUZSbU5tWnlXWEZDU0hWYU4xQTVUbEEzVFdaVFEy
          MDRJaXdpZVNJNklqWmhiMWgxUzJFeVVqZHNTMW93YlU5UlJUQmFlV0pQU2pKWlYy
          bzNOM1l3Wm5GWU1ESnBiRE5EVUVVaUxDSmpjbllpT2lKUUxUSTFOaUo5ZlEiLCJl
          bmNyeXB0ZWRfa2V5IjoiIiwiaXYiOiJYMFhTUDNZVTNBUkJwQ1NlIiwiY2lwaGVy
          dGV4dCI6Ik5DQXE1dnBCeUVaVERJcHlWem5Gemxtbm1JU09sVk5uNGpSUUtKWjB5
          c0s0dzZPcDYtNE94cGtvVVY5WUFvbDhmdUVOeFFMdjFBQUpDWXB3M0ZFelRrMEpG
          VmU1NWE0UlNIVkhnSlJmMEhiWjlxbk5aOHY0d1JUaXBGS0RsakpvLUhXOG82bzlM
          czV2YmRVTGJPRk9Db3RTTGZuekdSQ3lMV3Z2TUZaS3BHXzM1b21PeFpNcW1oN2Ro
          c3IxMmF6cHdkSnJKX084TTFkVHdDaWZxeURlWWFpNGM4M3U4TUhieDdETldRWkhH
          QnIzT1J0bDhaWGJTQW9OQ09fVWRpdU8zWXZmWmNiWU51TTY2UXBZbDFobENSaDJO
          eEZtLW12VUR0a1VoaXR5cTdyZ3BSbWZoYndKNklCaGdpdyIsInRhZyI6Ijh3YWx6
          T0Q4U3hWTC1STXY3OXlTZGcifQ",
      "signature":
         "8-09XlWUUDsXXqoEh5EKIAYEOTR-vtAYqauW1aNfdv2Io9B4RCuAL13zi7i27vbo
          TYvHxnFa7K6HJPygsAVn5g "
   },
   "dac_response_dest_certificate": {
      "kty": "EC",
      "x": "joyfi05KEI3hcOhJeOfny_TWsZ9FFS1zUydFQhm3G78",
      "y": "Nsk3jX1ph0FH8APR2k0XSu6pDZYyF7f_Okplf7hZ_8k",
      "crv": "P-256"
   }
}
```

Once created, the packaged DAC response shall be returned as the response to the HTTPS/HTTP request, or submitted using the DAC response URI specified in the DAC request, for example, as an HTTP PUT operation or via an SMTP email. The `dac_response_dest_certificate` and `dac_response_dest_uri` may also be used to route the request through intermediary hops if needed, as determined by the routing system, which is out of scope of this standard.

When the CDMI server receives a packaged DAC response message, it shall decrypt it using its private key and shall verify the signature. If the decryption and signature verification are successful, the CDMI server shall use the provided `dac_applied_mask` in place of the ACL computed mask.

If the CDMI server supports key or DAC response caching, cache expiry values shall be honored. Cached responses and keys may only be used for identical client operations, where the client identity, objectID, operation, and “CDMI-

[2] Decrypt with “d”: “huCoV1iC24rZ3uF5q-1HHlGb2UcC6Ue9oNezEQNzUB8”

DAC-" request headers are identical. Otherwise, the cached response shall be expired. If an audit URI is present in the cached response, audit messages shall also be generated for all operations allowed using the cached response.

The CDMI server shall also implement audit logging when specified in the DAC response. If the CDMI server does not support audit logging and it is required by a DAC response, the operation shall be denied.

If a `dac_redirect_objectID` field is returned in the DAC response, the CDMI server shall return an HTTP redirect to the specified Object ID. This redirect allows a DAC provider to create a client-operation-specific instance of the object that is encrypted with a single-use key.

24.9 Error handling

In the following scenarios, the following HTTP response codes shall be returned to a client:

- When a DAC response denies the requested operation, an HTTP status code of `403 Forbidden` shall be returned to the client along with any `dac_response_headers` included in the response.
- When a DAC response includes a `dac_redirect_objectID`, an HTTP status code of `302 Found` shall be returned to the client along with any `dac_response_headers` included in the response.
- When a DAC request to access or modify an encrypted object is allowed, but the key is not included in the DAC response, an HTTP status code of `401 Unauthorized` shall be returned to the client along with any `dac_response_headers` included in the response.
- When a DAC request to access or modify an encrypted object is allowed, but cannot be performed due to lack of support for an encryption algorithm, signing algorithm, or key type, an HTTP status code of `501 Not Implemented` shall be returned along with any `dac_response_headers` included in the response.
- When a DAC request times out, an HTTP status code of `500 Internal Server Error` shall be returned to the client.
- When a DAC request cannot be sent or routed because the DAC metadata is not supported or valid, an HTTP status code of `501 Not Implemented` shall be returned to the client.
- When a DAC request cannot be sent or routed because an upstream system is unavailable, an HTTP status code of `500 Internal Server Error` shall be returned to the client.

24.10 Examples

The following examples illustrate the primary ways that DAC requests are performed.

EXAMPLE 1: GET ciphertext of encrypted object with delegated access control

The following CDMI operation is performed against an encrypted CDMI object with delegated access control metadata:

```
--> GET /MyContainer/MyEncryptedObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cms, application/jose+json
```

The CDMI server verifies local access controls and determines that the request can proceed. The following DAC request is generated:

```
{
   "dac_request_version": "1",
   "dac_request_id": "5b801b19-479e-446d-882a-8483f7c4905c",
   "server_identity": {
      "kty": "EC",
      "x": "joyfi05KEI3hcOhJeOfny_TWsZ9FFS1zUydFQhm3G78",
      "y": "Nsk3jX1ph0FH8APR2k0XSu6pDZYyF7f_Okplf7hZ_8k",
      "crv": "P-256"
   },
   "client_identity": {
      "acl_name": "anonymous",
      "acl_group": ["guest"]
   },
   "acl_effective_mask": "READ_ALL",
   "client_headers": {},
   "cdmi_objectID": "0000000800182F9E64313363323731622D363536662D3465",
   "cdmi_operation": "cdmi_read"
}
```

This request is first JWE encrypted with the key in cdmi_dac_certificate. The result is JWS signed, using either the key in server_identity, or a different key embedded in the JWS header.

The DAC provider verifies, decryptes and processes the request and returns the following DAC response:

```
{
   "dac_response_version": "1",
   "dac_response_id": "5b801b19-479e-446d-882a-8483f7c4905c",
      "dac_applied_mask": "ALL_PERMS",
   "dac_response_headers": {
      "CDMI-DAC-AuthInfo": "No key requested."
   }
}
```

The `CDMI-DAC-AuthInfo` indicates a custom header.

Since the operation is allowed by the DAC provider, the following response is sent:

```
<-- HTTP/1.1 200 OK
<-- Content-Type: application/jose+json
<-- Content-Length: 290
<-- CDMI-DAC-AuthInfo: No key requested.
<--
<-- <JOSE+JSON Encrypted Object>
```

EXAMPLE 2: GET ciphertext of encrypted object with passthrough key access

The following CDMI operation is performed against an encrypted CDMI object with delegated access control metadata:

```
--> GET /cdmi/2.0.0/MyContainer/MyEncryptedObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cms, application/jose+json
--> Authorization: Basic am9lOnBhc3N3b3Jk
--> CDMI-DAC-N: <vendor-specific header that indicates key passthrough>
```

The CDMI server verifies local access controls and determines that the request can proceed. The following DAC request is generated. The CDMI-DAC-N is a custom header that indicates that the client wants to obtain the object decryption key via header pass-through.

To demonstrate the power of such custom headers: the CDMI-DAC-N request header could contain a cell phone number. The matching response header would then contain a password-based encryption of the object key, while the password will be delivered via a message to the cell phone. It is up to the vendor to come up with and implement such mechanisms.

```
{
   "dac_request_version": "1",
   "dac_request_id": "77b54650-183f-4053-8512-be08f7c6c50e",
   "server_identity": {
      "kty": "EC",
      "x": "joyfi05KEI3hcOhJeOfny_TWsZ9FFS1zUydFQhm3G78",
      "y": "Nsk3jX1ph0FH8APR2k0XSu6pDZYyF7f_Okplf7hZ_8k",
      "crv": "P-256"
   },
   "client_identity": {
      "acl_name": "joe",
      "acl_group": ["users"]
   },
   "acl_effective_mask": "READ_ALL",
   "client_headers": {
      "CDMI-DAC-N": "<copy from headers>"
   },
   "cdmi_objectID": "0000000800182F9E643133633233731622D363536662D3465",
   "cdmi_operation": "cdmi_read"
}
```

This request is first JWE encrypted with the key in cdmi_dac_certificate. The result is JWS signed, either using the key in server_identity, or a different key embedded in the JWS header. Replication of these encrypted messages is not useful and will be skipped.

The DAC provider processes the request, obtains the object decryption key and embeds it as a dac_response_header, then returns the following DAC response:

```
{
   "dac_response_version": "1",
   "dac_response_id": "5b801b19-479e-446d-882a-8483f7c4905c",
   "dac_applied_mask": "ALL_PERMS",
   "dac_response_headers": {
      "CDMI-DAC-AuthInfo": "Key successfully retrieved from keyserver."
      "CDMI-DAC-N": "<vendor-specific decryption key info>"
   }
}
```

Since the operation is allowed by the DAC provider, the following response is sent:

```
<-- HTTP/1.1 200 OK
<-- Content-Type: application/jose+json
<-- Content-Length: 290
<-- CDMI-DAC-AuthInfo: Key successfully retrieved from keyserver.
<-- CDMI-DAC-N: <vendor-specific decryption key info>
<--
<-- <JOSE+JSON Encrypted Object>
```

The client can now parse the key in the CDMI-DAC-N header and use it to decrypt the ciphertext.

EXAMPLE 3: GET plaintext of encrypted object with delegated access control

The following CDMI operation is performed against an encrypted CDMI object with delegated access control metadata:

```
--> GET /cdmi/2.0.0/MyContainer/MyEncryptedObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Accept: */*
--> Authorization: Basic am9lOnBhc3N3b3Jk
```

The CDMI server verifies local access controls and determines that the request can proceed. The following DAC request is generated:

```
{
   "dac_request_version": "1",
   "dac_request_id": "b79d7619-1bbd-45a1-b2d3-5753f7fc5155",
   "server_identity": {
      "kty": "EC",
      "x": "joyfi05KEI3hcOhJeOfny_TWsZ9FFS1zUydFQhm3G78",
      "y": "Nsk3jX1ph0FH8APR2k0XSu6pDZYyF7f_Okplf7hZ_8k",
      "crv": "P-256"
   },
   "client_identity": {
      "acl_name": "joe",
      "acl_group": ["users"]
   },
   "acl_effective_mask": "READ_ALL",
   "client_headers": {},
   "cdmi_objectID": "0000000800182F9E64313363323731622D363536662D3465",
   "cdmi_operation": "cdmi_read",
   "cdmi_enc_key_id": "0000000800182F9E64313363323731622D363536662D3465"
}
```


3970 The DAC provider processes the request, obtains the object decryption key and returns the following DAC
3971 response:


```
{
   "dac_response_version": "1",
   "dac_response_id": "b79d7619-1bbd-45a1-b2d3-5753f7fc5155",
   "dac_applied_mask": "ALL_PERMS",
   "dac_object_key": {
      "kty": "oct",
      "kid": "0000000800182F9E64313363323731622D363536662D3465",
      "use": "enc",
      "alg": "dir",
      "k": "vBX811eh8ydyI08by7L13kZKNmfRHTAMZa5vJqMCHQU"
   },
   "dac_response_headers": {
      "CDMI-DAC-AuthInfo": "Key successfully obtained from KMS."
   }
   "dac_key_cache_expiry": "2017-04-05T14:58:58Z",
   "dac_response_cache_expiry": "2017-04-05T14:58:58Z"
}
```


3972 Since the operation is allowed by the DAC provider and the key is provided, the object is decrypted by the
3973 CDMI server and the following response is sent:

```
<-- HTTP/1.1 200 OK
<-- Content-Type: text/plain
<-- Content-Length: 252
<--
<-- <Decrypted contents of Encrypted Value>
```

3974 EXAMPLE 4: RSA Example

3975 In this example, there are two hospitals (A and B), that both have CDMI servers, and federate objects
3976 between them. At some point, the following encrypted object has been made at hospital A. It containts
3977 a cdmi_dac_certificate and cdmi_dac_uri that indicate how access can be requested at hospital A. The
3978 cerificate contains a 2048-bit RSA encryption key, with a matching X.509 certification chain that can be
3979 used to verify the certificate.


```
{
   "objectType": "application/cdmi-object",
   "objectName": "MyEncryptedObject.txt",
   "capabilitiesURI": "/cdmi_capabilities/dataobject/",
   "objectID": "000000080018F34436313131393061372D613735302D3438",
   "mimetype": "application/jose+json",
   "metadata": {
      "cdmi_size": "306",
      "cdmi_dac_uri": "https://cdmi.hos-a.fr:9001/dac/",
      "cdmi_atime": "2017-04-06T14:06:34",
      "cdmi_enc_key_id": "encryption_key_1",
```


(continues on next page)

(continued from previous page)


```
      "cdmi_dac_certificate": {
         "kty": "RSA",
         "kid": "cdmi.hos-a.fr_encrypt_public",
         "key_ops": [
           "wrapKey",
           "unwrapKey",
           "encrypt",
           "decrypt"
         ],
         "n":
            "uL7ANgD80H5sNqo3nHzovPRxgncQLhz0oQvGMVvULCkrYXMaXZ5sNv7fT6UdMSZi
             T-e0sthapmEqrpeV9RKHsiF3COGl2YndUHixpEkHp8y1ggcH6iTzoBsgXMZ7OLW-
             mJ2RW3rodT7k-tcozYYsTSM5egMPQSAKgt0nMnPmdNRnEyA2_NJ8Y7lNkEXyja0Q
             JLstzkP8-cKS0BkEquLQEMbZVRM6U5uG69cj1i9OWvuRzPoaATKyt6Cc4f6PUu9L
             OyCBUAs9dXsRrt3B8H1qe7io7FAAcOpcUDKDnLFXS1THc37DK_zEyKZcMttjCvEl
             Ovt-cIaokdnxJeggv9AFGQ",
         "e": "AQAB",
         "x5c": [
           "MIIDMDCCAhigAwIBAgIBBDANBgkqhkiG9w0BAQsFADBCMQswCQYDVQQGEwJubDER
            MA8GA1UEChMIbGllc2RvbmsxDTALBgNVBAsTBGNkbWkxETAPBgNVBAMTCHJzYS1y
            b290MCAXDTE2MTAyNzEyNDUwMFoYDzk5OTkxMjMxMjM1OTU5WjA1MQswCQYDVQQG
            EwJmcjEOMAwGA1UEChMFaG9zLWExFjAUBgNVBAMTDWNkbWkuaG9zLWEuZnIwggEi
            MA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQC4vsA2APzQfmw2qjecfOi89HGC
            dxAuHPShC8YxW9QsKSthcxpdnmw2/t9PpR0xJmJP57Sy2FqmYSqul5X1EoeyIXcI
            4aXZid1QeLGkSQenzLWCBwfqJPOgGyBcxns4tb6YnZFbeuh1PuT61yjNhixNIzl6
            Aw9BIAqC3Scyc+Z01GcTIDb80nxjuU2QRfKNrRAkuy3OQ/z5wpLQGQSq4tAQxtlV
            EzpTm4br1yPWL05a+5HM+hoBMrK3oJzh/o9S70s7IIFQCz11exGu3cHwfWp7uKjs
            UABw6lxQMoOcsVdLVMdzfsMr/MTIplwy22MK8SU6+35whqiR2fEl6CC/0AUZAgMB
            AAGjPDA6MAwGA1UdEwEB/wQCMAAwHQYDVR0OBBYEFBAIGICMR5H6KLKMlZAEhCCc
            KwE9MAsGA1UdDwQEAwIEMDANBgkqhkiG9w0BAQsFAAOCAQEAANYSSryUU6112pYM
            r83M3GWnjzu16B+4KgimZ8kbey94zNPdwmwQdSe0Xmg+1Otc6VUB40ouNnwK8efB
            aWBtXwCA7Nb715nTqo2+rn+X+A0mGrYaKkToPEe8ZYwDcOlOpNC9JFE+QgP9/CJa
            AaWrf95W+4kra2WnA4Bhqu2WWXnQkL47/nKcGVZgQAH+mVnxPaI0gELYdonXU/S2
            8HqxoyjpGL/vmyc46zUbxYsgx/jiE7J0fJVP6Yk/3dlNYCCpLtV8VmzFAQAeCcn8
            AWowFcd09a4SY09rn1MUv/rrvXpzflfn9j7PtRRFj2e/KhitmOH1zKDuYzREpUOu
            TDlPIQ==",
           "MIIDQDCCAiigAwIBAgIBATANBgkqhkiG9w0BAQsFADBCMQswCQYDVQQGEwJubDER
            MA8GA1UEChMIbGllc2RvbmsxDTALBgNVBAsTBGNkbWkxETAPBgNVBAMTCHJzYS1y
            b290MCAXDTE2MTAyNzEyNDQwMFoYDzk5OTkxMjMxMjM1OTU5WjBCMQswCQYDVQQG
            EwJubDERMA8GA1UEChMIbGllc2RvbmsxDTALBgNVBAsTBGNkbWkxETAPBgNVBAMT
            CHJzYS1yb290MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAsrUj46dx
            5ojlaZk7YtOL6e+Q6JoG7gVMaXkJn1Szlx9ND/8w4PeO1SQ2skukdOHAlQRmxdft
            zhccNTM5hmbcn8TAfWSYqQF1R7s78bVjtmat6AQP1vSgiyZ8Ak+iYZEq3c2zVyYQ
            HKKxWxmFZt1HT8/H/B3bXveXQcERKE+Tq66h8pqVcocQUtzRFsEYmv0bR1rghtoq
            H8nhB5xnebgVlXjApW+et2SE7r6Fjv1aAbGI89ouJ1gsMPeX56P8AUjacFtNKc44
            Obu6HRXY/jm6f2mlEUm84EUsJ+9b5+S2x4qPttJDfSCasWYYz4mFJ8MwmFiBGUwf
            geT2bUm6t7qqbQIDAQABoz8wPTAPBgNVHRMBAf8EBTADAQH/MB0GA1UdDgQWBBR+
            tEB2udkEXxX0k15GztF/4ol03jALBgNVHQ8EBAMCAQYwDQYJKoZIhvcNAQELBQAD
            ggEBAIjx1f9rJ2B+mDSA3L2GRhjrPRjfI6Un3Z51CeW9gO9PMQ5ws5pDJyB79dE/
            Q8Uf1e8pZyjchTsRa8GRdnKyndN2imayOVUvPoTd3/ZSmfkurcbj3I4VW8sjHP7C
            E8fmUS8Xprdpo2SxV7oneJC0vt5eyh8mgfJ/qSbwVaiXuH1Wxi6duAvdxddMXAxQ
            KPG1KKVM7CYfCdpX/HagCOHzcto+374zFqqnQ1Kx5rbgvxNSgm/PDDOMwP03+bbT
            R63KSK1VbdtLBuS4jgaPabwyxQz/FciwTu/HLOQn8TNqDWyoIbs+eQX2Mds2Apul
            8XH2+CakjBLMLL3Tlj2x+6tKR9o="
         ]
      },
      "cdmi_ctime": "2017-04-06T14:06:31"
   },
   "valueTransferEncoding": "json",
   "value": {
      "protected":
         "eyJraWQiOiJlbmNyeXB0aW9uX2tleV8xIiwiYWxnIjoiQTI1NktXIiwiY3R5Ijoi
          dGV4dC9wbGFpbiIsImVuYyI6IkEyNTZHQ00ifQ ",
      "encrypted_key":
         "329yyozEo3JPCpXGPKyI_fa5hhFH9dmfB7kulg1Q6NhoVAvdmDMclg",
      "iv": "9Gr5Hxzcs9hxPmPM",
      "ciphertext": "-sJkcHcdQUXChEBlZm7UZya1RR2_IcpRocC-BmQfAuA3",
      "tag": "VIFJDcMdZngtpLWWDX8vFw"
```


(continues on next page)

(continued from previous page)

```
    }
}
```

This encrypted object has been federated to the CDMI server at hospital B. Now, one of its clients wants to transparently access the plaintext of this object by performing the following operation:

```
--> GET /cdmi/2.0.0/MyContainer/MyEncryptedObject.txt HTTP/1.1
--> Host: cdmi.hos-b.us:9002
--> Accept: */*
```

The CDMI server at hospital B will look up the object and find out that it is an encrypted object with DAC information attached. As a result it will generate the following (plain) DAC request:

```
{
   "dac_request_version": "1",
   "dac_request_id": "73da04e1-2182-447e-8342-f4b9f06ec936",
   "server_identity": {
      "kty": "RSA",
      "kid": "cdmi.hos-b.us_encrypt_public",
      "key_ops": [
         "wrapKey",
         "unwrapKey",
         "encrypt",
         "decrypt"
      ],
      "n":
         "oQMQkY85UzwO7K6H0QQNfAiRMN3ZfhK0aXEKx7YwvrCU9IKOquZ1OYZ9Cv8556_8
          E8yZm02JDWOBOaSsGHU835jvXf12f4MywKGWj5FtIGL-j9kXF6SWq3zuLVYlXpMI
          KsJngHMVFca_-ZhZ2vLsrnDR1aCNEC48gR26ewp6WX1ptnSc1W4x3Mj-ONMVzxVE
          7XNlwYysTgDtonmTQD-YG6_KhhAPx0YowMbUWv_cMQvXsi7MMDyZn6fxfq4zQmQ2
          V5RtUy5msd6K3beDzS4LmZhsJmjU7YnhOj0pZby4Zckm43npjXPAuwPhzK2OW7qb
          fkv0qm4rsFWUcuNh8lBsDw",
      "e": "AQAB",
      "x5c": [
         "MIIDMDCCAhigAwIBAgIBBTANBgkqhkiG9w0BAQsFADBCMQswCQYDVQQGEwJubDER
          MA8GA1UEChMIbGllc2RvbmsxDTALBgNVBAsTBGNkbWkxETAPBgNVBAMTCHJzYS1y
          b290MCAXDTE2MTAyNzEyNDUwMFoYDzk5OTkxMjMxMjM1OTU5WjA1MQswCQYDVQQG
          EwJ1czEOMAwGA1UEChMFaG9zLWIxFjAUBgNVBAMTDWNkbWkuaG9zLWIudXMwggEi
          MA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQChAxCRjzlTPA7srofRBA18CJEw
          3dl+ErRpcQrHtjC+sJT0go6q5nU5hn0K\/znnr\/wTzJmbTYkNY4E5pKwYdTzfmO
          9d\/XZ\/gzLAoZaPkW0gYv6P2RcXpJarfO4tViVekwgqwmeAcxUVxr\/5mFna8uy
          ucNHVoI0QLjyBHbp7CnpZfWm2dJzVbjHcyP440xXPFUTtc2XBjKxOAO2ieZNAP5g
          br8qGEA\/HRijAxtRa\/9wxC9eyLswwPJmfp\/F+rjNCZDZXlG1TLmax3ordt4PN
          LguZmGwmaNTtieE6PSllvLhlySbjeemNc8C7A+HMrY5bupt+S\/SqbiuwVZRy42H
          yUGwPAgMBAAGjPDA6MAwGA1UdEwEB\/wQCMAAwHQYDVR0OBBYEFH7NJvMIfTQtZn
          nyiIdLNkjCgwSIMAsGA1UdDwQEAwIEMDANBgkqhkiG9w0BAQsFAAOCAQEAdiADIv
          0v09SUDcPL+BKysvchn\/Sgx5KBu7n9KFwE31Dhx2zvT6ruL8kXdekPH9cfrDafW
          6I\/vnbzAVj02i5pM2cHayj13fTOWSVwpcQuvkoIF9eVIWONkemMMf7M7jpTw07z
          7S2T5usaDmMNpqj8y5pRpQo3PnBVxpEZJ0XaSdfuiHtVLDq8gDZCq6Hc2tt7JM3W
          njnQgs+1lSGRuqWOcpmVONIoqvhiolNDZV35Z7puRwqck1N2f1qyHHGBWXfCN9U4
          ci6q1BnWBIFV+hURge8NSbpqawolaNueUbTcKjN3JsMC4ZxhMF9rN3uuPn+UAYkA
          yQkcSmGSMM07wcAkMg==",
         "MIIDQDCCAiigAwIBAgIBATANBgkqhkiG9w0BAQsFADBCMQswCQYDVQQGEwJubDER
          MA8GA1UEChMIbGllc2RvbmsxDTALBgNVBAsTBGNkbWkxETAPBgNVBAMTCHJzYS1y
          b290MCAXDTE2MTAyNzEyNDQwMFoYDzk5OTkxMjMxMjM1OTU5WjBCMQswCQYDVQQG
          EwJubDERMA8GA1UEChMIbGllc2RvbmsxDTALBgNVBAsTBGNkbWkxETAPBgNVBAMT
          CHJzYS1yb290MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAsrUj46dx
          5ojlaZk7YtOL6e+Q6JoG7gVMaXkJn1Szlx9ND\/8w4PeO1SQ2skukdOHAlQRmxdf
          tzhccNTM5hmbcn8TAfWSYqQF1R7s78bVjtmat6AQP1vSgiyZ8Ak+iYZEq3c2zVyY
          QHKKxWxmFZt1HT8\/H\/B3bXveXQcERKE+Tq66h8pqVcocQUtzRFsEYmv0bR1rgh
          toqH8nhB5xnebgVlXjApW+et2SE7r6Fjv1aAbGI89ouJ1gsMPeX56P8AUjacFtNK
          c44Obu6HRXY\/jm6f2m1EUm84EUsJ+9b5+S2x4qPttJDfSCasWYYz4mFJ8MwmFiB
          GUwfgeT2bUm6t7qqbQIDAQABoz8wPTAPBgNVHRMBAf8EBTADAQH\/MB0GA1UdDgQ
          WBBR+tEB2udkEXxX0k15GztF\/4ol03jALBgNVHQ8EBAMCAQYwDQYJKoZIhvcNAQ
          ELBQADggEBAIjx1f9rJ2B+mDSA3L2GRhjrPRjfI6Un3Z51CeW9gO9PMQ5ws5pDJy
          B79dE\/Q8Uf1e8pZyjchTsRa8GRdnKyndN2imayOVUvPoTd3\/ZSmfkurcbj3I4V
          W8sjHP7CE8fmUS8Xprdpo2SxV7oneJC0vt5eyh8mgfJ\/qSbwVaiXuH1Wxi6duAv
          dxddMXAxQKPG1KKVM7CYfCdpX\/HagCOHzcto+374zFqqnQ1Kx5rbgvxNSgm\/PD
```

(continues on next page)

(continued from previous page)


```
          DOMwP03+bbTR63KSK1VbdtLBuS4jgaPabwyxQz\/FciwTu\/HLOQn8TNqDWyoIbs
          +eQX2Mds2Apul8XH2+CakjBLMLL3Tlj2x+6tKR9o="
      ]
   },
   "client_identity": {
      "acl_name": "anonymous",
      "acl_group": ["guests"]
   },
   "acl_effective_mask": "READ_ALL",
   "client_headers": {},
   "cdmi_objectID": "000000080018F34436313131393061372D613735302D3438",
   "cdmi_operation": "cdmi_read",
   "cdmi_enc_key_id": "encryption_key_1"
}
```


This plain DAC request will be JWE encrypted using the key found in the object's `cdmi_dac_certificate` (key id 'cdmi.hos-a.fr_encrypt_public'). Then it will be JWS signed using hospital B's private signing key. Since this signing key is not equal to the encryption key (in `server_identity`) it is embedded in the JOSE protected header of the JWS (note: Base64 decode of the protected header reveals the signing key; Base64 decode of the payload reveals the JWE.)


```
{
   "dac_request": {
      "protected":
         "eyJraWQiOiJjZG1pLmhvcy1iLnVzX3NpZ25fcHJpdmF0ZSIsImp3ayI6IntcImt0
          eVwiOlwiUlNBXCIsXCJraWRcIjpcImNkbWkuaG9zLWIudXNfc2lnbl9wcml2YXRl
          XCIsXCJrZXlfb3BzXCI6W1widmVyaWZ5XCIsXCJzaWduXCJdLFwiblwiOlwicEVR
          aFFUMVF6QmdrV2RiVW56eVkwbkZmWjRVYXJnbFVPcGFxeGlXYXk5cGhnQ0x6Tmtj
          RHZ4eVdIdHFRSWE0ZHpvMDVaXzBiOXhNTElrYUI5MTJheV83MllOZHpmMWVfaUVX
          Mi1OdVB4MHVSaFV3SzQ4WUo2MFlwVTdpN2ZpQWNKeVJoU1dlWGtnQXQyRndUYnkt
          SjlsNW9DV1dZemRfc0U3a2NMSkc0QmkwSEtQbVhrUEVwbXpOamhsU0VsdnlodHFL
          djRERG1JRk1JTDNrUGJueGNfX0RwenAyaVVpdGhvUFhpY1pJQXMtUDIybGRGMkRE
          X0tzbW9SU3RQR2NuTEVYbWpKcXhoRU13Qm5UZE14TjdQNnh6bk5iQVNtdXNnR21F
          XzJXdVUyS09yLVBtYm5wTnNLcm14SHRhT2trc2pZdjFyVGhzRmkxNUZmSVQyQ1dU
          MnVRXCIsXCJlXCI6XCJBUUFCXCIsXCJ4NWNcIjpbXCJNSUlETURDQ0FoaWdBd0lC
          QWdJQkF6QU5CZ2txaGtpRzl3MEJBUXNGQURCQ01Rc3dDUVlEVlFRR0V3SnViREVS
          TUE4R0ExVUVDaE1JYkdsbGMyUnZibXN4RFRBTEJnTlZCQXNUQkdOa2JXa3hFVEFQ
          QmdOVkJBTVRDSEp6WVMxeWIyOTBNQ0FYRFRFMk1UQXlOekV5TkRVd01Gb1lEems1
          T1RreE1qTXhNak0xT1RVNVdqQTFNUXN3Q1FZRFZRUUdFd0oxY3pFT01Bd0dBMVVF
          Q2hNRmFHOXpMV0l4RmpBVUJnTlZCQU1URFdOa2JXa3VhRzl6TFdJdWRYTXdnZ0Vp
          TUEwR0NTcUdTSWIzRFFFQkFRVUFBNElCRHdBd2dnRUtBb0lCQVFDa1JDRkJQVkRN
          R0NSWjF0U2ZQSmpTY1Y5bmhScXVDVlE2bHFyR0packwybUdBSXZNMlJ3Ty9ISlll
          MnBBaHJoM09qVGxuL1J2M0V3c2lSb0gzWFpyTC92WmcxM04vVjcrSVJiYjQyNC9I
          UzVHRlRBcmp4Z25yUmlsVHVMdCtJQnduSkdGSlo1ZVNBQzNZWEJOdkw0bjJYbWdK
          WlpqTjMrd1R1Undza2JnR0xRY28rWmVROFNtYk0yT0dWSVNXL0tHMm9xL2dNT1ln
          VXdndmVROXVmRnovOE9uT25hSlNLMkdnOWVKeGtnQ3o0L2JhVjBYWU1QOHF5YWhG
          SzA4Wnljc1JlYU1tckdFUXpBR2ROMHpFM3MvckhPYzFzQkthNnlBYVlUL1phNVRZ
          bzZ2NCtadWVrMndxdWJFZTFvNlNTeU5pL1d0T0d3V0xYa1Y4aFBZSlpQYTVBZ01C
          QUFHalBEQTZNQXdHQTFVZEV3RUIvd1FDTUFBd0hRWURWUjBPQkJZRUZCeHdnVzB4
          TFV3QlRSaUlTMVZ2di9KVmNPMlFNQXNHQTFVZER3UUVBd0lIZ0RBTkJna3Foa2lH
          OXcwQkFRc0ZBQU9DQVFFQVI5bzMxQmR6N080d21GcFE0eEZjalFkSktSaFBiNndk
          RHdyOTM5OXd5OFUxV0VFOEpEQ0FvbW9nakJlQ2RLUWVqWE1oVjR1Vkx1YllpSitj
          WVRxZXh0NTJNb0pJRmUySnozNC9LbFVkYU5ENE5Jdm5teC9mWS83Qk9qQWFkY2Ny
          L0NQVUxmczE5OHUybG9GNUVSYWtPM2lGajhwWFY1MjlDSFFmekpsaGh0c0VoL3p2
          TXgydXpNTlpaWFlkVWxyQ0NBZGFGbWtFRDBHUHM4SGZDR1VXUytlQVNiN1ZnQXBC
          N0NYb1ZJLzVKa2JzL0ZreEF3TWlxSmE2RUpyYkRkSFl0N2prVktwcmVpMWlxRVBj
          Ri9MU0pNZXhGVjJVcDVISk9OMG1QMGFDclYwbFE2Nys3Mjg2Mkw1VmFjM0tjQTk0
          dVBZVTUxVEJITFNBVXRLTDI5Uk9oZz09XCIsXCJNSUlEUURDQ0FpaWdBd0lCQWdJ
          QkFUQU5CZ2txaGtpRzl3MEJBUXNGQURCQ01Rc3dDUVlEVlFRR0V3SnViREVSTUE4
          R0ExVUVDaE1JYkdsbGMyUnZibXN4RFRBTEJnTlZCQXNUQkdOa2JXa3hFVEFQQmdO
          VkJBTVRDSEp6WVMxeWIyOTBNQ0FYRFRFMk1UQXlOekV5TkRRd01Gb1lEems1T1Rr
          eE1qTXhNak0xT1RVNVdqQkNNUXN3Q1FZRFZRUUdFd0p1YkRFUk1BOEdBMVVFQ2hN
          SWJHbGxjMlJ2Ym1zeERUQUxCZ05WQkFzVEJHTmtiV2t4RVRBUEJnTlZCQU1UQ0hK
          ellTMXliMjkwTUlJQklqQU5CZ2txaGtpRzl3MEJBUUVGQUFPQ0FROEFNSUlCQ2dL
          Q0FRRUFzclVqNDZkeDVvamxhWms3WXRPTDZlK1E2Sm9HN2dWTWFYa0puMVN6bHg5
          TkQvOHc0UGVPMVNRMnNrdWtkT0hBbFFSbXhkZnR6aGNjTlRNNWhtYmNuOFRBZldT
          WXFRRjFSN3M3OGJWanRtYXQ2QVFQMXZTZ2l5WjhBaytpWVpFcTNjMnpWeVlRSEtL
          eFd4bUZadDFIVDgvSC9CM2JYdmVYUWNFUktFK1RxNjZoOHBxVmNvY1FVdHpSRnNF
```


(continues on next page)

(continued from previous page)


```
        WW12MGJSMXJnaHRvcUg4bmhCNXhuZWJnVmxYakFwVytldDJTRTdyNkZqdjFhQWJH
        STg5b3VKMWdzTVBlWDU2UDhBVWphY0Z0TktjNDRPYnU2SFJYWS9qbTZmMm0xRVVt
        ODRFVXNKKzliNStTMng0cVB0dEpEZlNDYXNXWVl6NG1GSjhNd21GaUJHVXdmZ2VU
        MmJVbTZ0N3FxYlFJREFRQUJvejh3UFRBUEJnTlZIUk1CQWY4RUJUQURBUUgvTUIw
        R0ExVWREZ1FXQkJSK3RFQjJ1ZGtFWHhYMGsxNUd6dEYvNG9sMDNqQUxCZ05WSFE4
        RUJBTUNBUVl3RFFZSktvWklodmNOQVFFTEJRQURnZ0VCQUlqeDFmOXJKMkIrbURT
        QTNMMkdSaGpyUFJqZkk2VW4zWjUxQ2VXOWdPOVBNUTV3czVwREp5Qjc5ZEUvUThV
        ZjFlOHBaeWpjaFRzUmE4R1Jkbkt5bmROMmltYXlPVlV2UG9UZDMvWlNtZmt1cmNi
        ajNJNFZXOHNqSFA3Q0U4Zm1VUzhYcHJkcG8yU3hWN29uZUpDMHZ0NWV5aDhtZ2ZK
        L3FTYndWYWlYdUgxV3hpNmR1QXZkeGRkTVhBeFFLUEcxS0tWTTdDWWZDZHBYL0hh
        Z0NPSHpjdG8rMzc0ekZxcW5RMUt4NXJiZ3Z4TlNnbS9QRERPTXdQMDMrYmJUUjYz
        S1NLMVZiZHRMQnVTNGpnYVBhYnd5eFF6L0ZjaXdUdS9ITE9RbjhUTnFEV3lvSWJz
        K2VRWDJNZHMyQXB1bDhYSDIrQ2FrakJMTUxMM1RsajJ4KzZ0S1I5bz1cIl19Iiwi
        YWxnIjoiUlMyNTYifQ",
    "payload":
       "eyJwcm90ZWN0ZWQiOiJleUpyYVdRaU9pSmpaRzFwTG1odmN5MWhMbVp5WDJWdVkz
        SjVjSFJmY0hWaWJHbGpJaXdpWVd4bklqb2lVbE5CTFU5QlJWQWlMQ0psYm1NaU9p
        SkJNalUyUjBOTkluMCIsImVuY3J5cHRlZF9rZXkiOiJ0WkQ4ZkpUT1hGZ3NTVi11
        RXN1ZS1VYURnVVpQX3FFZWFVUFFyQmpObUF3UlpMSXlONklUc3hhclRqbDR3YjdH
        UXltN0prOWw4QXlld0Fkakltby1yOUNULUhSbmR5Nlc2Nkd4bW11TGhwOV9ncjl2
        WEtjeXh3QUhHeGIyZGo3dmlIQjlIRElNazZEQzB1Ri1naWZlM3VaeEhjdVVfYkRG
        VGZBU0M2TEwyLTlIQ29OMzlSM2VZcC1FeWc0clhwZ2t6SkpnLTdRSUpHV3pvRFF1
        d2VfeEM3Vmh5Q25vdDc2bW9RbEpSM1I5Snh6M2M4TE96ckp0YTY3YjdhS3Zodzk5
        eUNPX3REb3d0WDZpNUd1cmZVbE5GN2VucmUwOWd3cUlLT3ZxQWRDTVdlSU5YWnVO
        WjlZX3Y5RjVvWlF1M21BcEdib0dVdVRvRWxUMFFKd1dwaUR6bUEiLCJpdiI6InVJ
        ODQyQ3ZsRUpBYTd4Y1YiLCJjaXBoZXJ0ZXh0IjoiVjBCcWtLTV93TmV6UVJuOGxv
        b3VOdFhPem9tSjJLbElYaU1NT0VQSk1NTVZpd0lKY05IWHdBV01wUnBoMlRrMnlS
        dUoweHBZbmRPdVBtUHdPVGljMWR4YXZvaGJEdHloTEFQS09TNWIzSk9lQTgxbWU3
        enRaeFdxOFRDVVRFVkhBVTM5dERzZUQyZ0tUME0yOFR1dXpEZHhJcEtUMXJQRXZr
        WThIQVBjakhZQ2pTeVpmcEhJSlpkUUhEclo0YUZBN0xETkkxWkxPOEhXMmtJbFB6
        UzQ0WFdKQzYwS243THJvSXRyZ0hCWnJfZEFpNWlIQTdmU0VCSFVSRFpPQ0pfdzlr
        TldLMDZfVTN3S2xVNU9DeFFYNzFMRGxCQllENlpaa2lwaExoRl91RUxvZFdKREEo1
        Zmg4VnhDRUJ6ZVdqckR1M3UzMXNldHJEbmRIaFNsT2VyeEswNFZuQXNETlczRXNT
        a2twLXp5cFh4d2ZBRVQ0VWxiR3V0UV9FX2xsX2M4VVZjSnJhRXJoZFgzX2VNajdS
        YThud2VEdGxZWjlDLWVlQkdJYzE1OTZLaVZzQlJ0c1dPZm5SLWQzdVZFREMwZnVv
        alp1U3M1Mm1mWjh5dU1yeE9QWDJodzFVVGlCeHRuODdyQjRNc1ppSkwwc2VIelIz
        T3d4eEFMUFJRSzFzQVNweXJRaW1jTVJBSUtGVzJUUDFjSjRmYTlIdmNTbzN0eTUx
        cENSVGlmUV9CSFlLM01LaGdWMWVnZ0I2Mmw4SEs2NkE0dFlpcjJyUnhkNEZPb1U0
        bXN0bVQ3aFpmOVhBRzFBMzNwSHNxSEFINng4LVJDc0FwOHlGeHd1SHg3X0s2ZzNo
        ZzJNSTNHZUpBTGxMRXhyeUJ5NEF4OXQ5ZEg0SWkzZWZjZlh2LVROcmFIS0tHd3VM
        QUItby1jVG1WUWZfU3dGSEtHZGNjSWgwazFYT3VXdkFmekducjJKMnhBVjBUQzdz
        YWNyZHBWNzgyWkM2T2dSSVliTEs0RnhUYXpuMXY1dlZjRms4dU40NFgxV2VuM09f
        RmR1UllLZ0ZPbUFyTHd4WXBMYXJSS1QzX3FPcjlOQnYtQ3FycEJPV2EwVlVtaXF5
        TWxTb2VHR2plaHcwLVNuMVBTa3dWb25jNTVYTWxYcEVZc2hJb3dHN0xnb0dBakxl
        MU13cjkxNWxmWnZsajFpckljbGk5MGtIWVBKTjdZVTJEMVhaM2hEV3FLTTRLWG1w
        RnJaVk5qdi0yUWN0ZGt1Vk1MaHQwN2JESU51aUpDZENsS0llbWVVZlhZMW1IakxB
        N1J6Tjg0SjN4UTIzVXN5eFdWLTFYRTJueEFmcmYzMG5KZG5HTi1KbEhqdFMtQW5E
        cXljUGQ4VjhBcUw5UURrUm1tZmsxRnBwTUdEaW9XS1lVM2JTTGhyTHh0RzMweUQ2
        bzVlb0UtUFN0aW9XcGVsN1lZcVg0aGxlNGNoNWVhWkQyRF93Ni1VOUxLbmQ2NnZq
        OVptNjU2YVZrYy1SSGhNWEpQczhMeElCM2hrQ1BSMi1KN2lUaTBZYTh2d016OFJ5
        LVQ2Z285b3VVU2F3MHJYbi0wNm9HNC00aXZhMnZ2UkZGaXFOS25KYTdrM1BrWDF2
        cXVqQUl5Ti1ESkI2cmZNZnZxTlZmV0phNXBuNTZLY1g4TFBPeUFiSGo4WUNWT3Fo
        QldtbGt1SjM4WUtpY1dWRkJfTlpSZEpseDRTUXdoQ2pLYlJtdmxENXJqeEk4NTRY
        NTdzQkNQbmpaTzdvZDRpZ2lVbnlfZjNyRkZQd1lieVFmdWxkM3hMRVdSWWZCWUpu
        YWRvNzB0S1VXMllOcWFzUzR3SW9ibFhFSnR5a2RPTHdQLVkybGRVMXdVbThsUHdp
        VXpNZGtrY2xDWkQ2Wkx6LU9GdDdLaE9EbUxPTUdPNWlhczlzeG9DaG4zYzFMcnFQ
        amd1Ui1RWFNoaUZrNmtYVU5CMnlIVEt3dmZZT0ZjUjhEbDdleVJCQ3NVUUY4STdT
        ZkFWNkJrMm82ZU1jdEJGTXluWFgydU9nWXEwOWtUM0ppckhzbGl3dThZbzM1bHpW
        ZGViWnlqREIxbWl2LUNKR091ZjRxbEZxRzNiYVBCQmhNa0ZFS2lWU0FJZmdDaFlu
        dUc3WkxvNmVWMW4zMnRGeDFzdEdUcnpuMkl3alNzektPMTJIY3VkcUZnTUREQ28w
        T19OREZ4Qmh1RFB3aVNadnBGNHJWV201QktEbjBOQzNFcy1zOExGaDZJUFVTMW1l
        VFB2Umx0cERJZTljQTZqdHhCQURacVphdGtreXB5ZzZGRDZzUjhUQVRDRGo0WWhT
        Rzk4MVBmRFZ1S09FVjlqaTdzd3lkLW9ST2Q1UXZCa1lKUkxhckpEOGhEdWZIcHdy
        MlViNENBb1JKcnJWdGM4TmJDcmlLbmZGWVBQal8yN3p1RFZKZXBxQTdpOVVTY0ZD
        T25vUEx3NEFpVmxqQVRPVU93MmNzR0FqM19tLThPR2cxM3BTUHNjWDlSUm5TV1lY
        cjdFdnFJaUVXNDZaNTR6NlJTNHg0Z1dqZWFzNUpHdmRsQkhoQk42bFd4QTFLa0Vs
        el90NlVOSVhGSWpZRTdyVF9oUzJsWmstWGNPelZuM0gxTllZU3NpTVVLVDNvWm4t
        UmliVVhoMGZjbVNWTHFRMnVuX2dQV3MyT0VUeHB3U1Y2dk9leWRjOFcwQVdhSVph
```


(continues on next page)

(continued from previous page)


```
          Vm43bW1MNnY2aEctTVFLSl92S3pfaTNxbkRxRUVzVjhDQVE3NTN1cWpHUVBEMUs4
          VGtZeTNucnhncEh1ODY3UDJNUGpVQWFxeldTV1dNN2FQeVhCTkZNeFQ5V3Q5ZmVw
          VEFheDdhLVo1VGcwS1pxS0lDQVVNYUo4OWJveG1ncG9rcGFRVWtQS1pFMjR2NGph
          ZE5pT21SSHJMeE11VzVrdG5kQk5Bb0dHbm9HRTNzYWZlc24wX2NPc3M2TTVXcEhR
          Z1N6VWxacF9aREJ5Q09ldFo1TmEwMjVfb2t1T2drUklKVjN4MmZPbWM2U0wwU3Fy
          R1lWSEhORmZKN2lLeVNXTk5Sdkl4UmhWTmJYVXFMRl9OYUMxZDdNQmpkTHY3WlFp
          Ym0yT3BmSWtmSlNNUzFKcFVwSVNRakR4djk2TFhIOHBhQXBTVDZtWnlEa0VrTW1O
          b2ZXSktwM0pQNUFTTXZ2S3dsbkgxQU9rSWJYZjdRLXlyYkthY1BtWEhkZ2hjRmZa
          Vi1ldU9ERzhfdTRWdGZwOEF0LVBRSkJyX1V0cVZJbHpReDBnUE5MVVdMbzdlRmlv
          dEtPTEx3Vjh2MGpxcDdUUlBIVzBHc3pVWmFZb0RjYUZMaXZDaFJwVDktQks3d29w
          RXhfN1FxUWUxSVhuMVNKQWRtZFpvZmdfSF9ZeWU0MVJjSnZGeWp6SmZBOGlranBs
          STMxVW9WSzVaTllXSVExZ2lvUFh3U3JUVW1GeE45X3Z0aHl5Z1pYTXQzVkZiUTNB
          eFBBZjlSSE5ENlpBOGFValk2R0pNb1BvX3FySkMyLW9hQzd0cTc0SURBekk3bmpM
          S09VMG9IZ3RtaGhGUVB6cloyWjZOYUlLaTJIWm9yLWc3LWctN1N3ZGRxd0RDSnlO
          ZXdEb3pJN3NNRFM3XzM1RkoydnRnaGZLUHFTLWhFd3JlZGtaNE1ORUg1dnRXU3F3
          NHRmZDFrQmNROExtRlZmM2hfV1FWZjVqQ0lGcEZNOV9MYnV6eWVmeU9IMENSLXlJ
          YlE0QXFleS1ZclA0YWJTdXF2VXBlMEFyVEdZS3pMWk5FZHhFVzliemxsYUVJaVNU
          MTRBNXAzam9CUVNOQmtLRUdPMVh2RTdSVnJ3T1kyemItWWFIRkxjU0gzdDRtMFlZ
          aVducFZ1RnBRZjExWVlqbTgzLXFEbHZuYUN5T19GQXBTMy1SRnhrVTZFVzRYcDZ3
          eVlxNDF1amxGVHp4Q1RnMExxeklqTGlyRm1VSGpKZUlhR3lVSzZ5TGpkYkhtR0Ns
          LW15T3FZR3hwNERybHVITHBkaWZHenhmQnd6OUM4MXNNbkFxYmNRWGMwcXdIMVdO
          al9uX0tMblJFdUhRWFBleDV2VmZ6eWRlZUZfNE93WExSdXZ1M3Zl3Z3VldWamRP
          ZWN2a05nbkVFUDdocVhkbXpRbVlKZndxSGRwZnNSRExESDloOWpRcEI0LTNMRjg4
          U2NSa1RteFlNU291bFVqcHdia25hTWZQRVMtZkgtTjVwQXNqVE0wOVFTV0Y5NENE
          bHdZR1ZhZzY2UkdLcGdwU1ptMGF1dGNsSG1KOGVRMjRuRzNKdEF6bGNkVC0zNXNC
          TlFsZHRkYlhkRzFxb2ZlOUpUNE5iSll1VXdwemlhNnJYNGQ5NkRqdm1VdWpqanNl
          NjhCYkx2bzJQM1VyTkxUWDBjNmUtb3lHbzcwdURTN1lxb2ZIUUxhU05ORHRnZ3d1
          V1IyZFRVUDdLRkk5N29Qb0ZiX1JjS3BQVVU4cDZsTjJRYkVOYy00TWNleWJYVS1U
          ODR1ZnpqZkJkdHdMdkYtQ3d6Nk9lS1V1NnBPeVN2MGVnbFNJUWlmb3JPU3FGVl9y
          QWdvX3NuNDVHVVhKU0dNc0FFT0NXN0lQX0E4Y1hvaFpGbi1NNW1UTU83b01fSWN1
          Q0dVclNuREV2X3NNdXFLdFk0cExbHRMT2tCTXVHU292Mm1DVHBmQjVnVnBvaGZf
          UzBaUW9CWlV2UDkwU3ZLUVRuTG12QnZnVTd3OVVmY1dRTzBrNUZZSjNfREZxNHVL
          VWJpNXB5cFVCT0txd0Zla3ljLXZXVGRPTGdLOW40ZkdUcjlzOG9ZejczdC1tVXZ5
          a3pfdkRzWWFpWHZQbkd3My00SDhVWllXZy1GSHJfSHlLTlFQNklGSTVGdEFneW5s
          QjF6SEQ2a0xXbGtGaWdfb0FjdXJ6enE3WXJJaUlOcnZFUzFUZUgwRXBwczVjZHpF
          WWh0M3ZHNFMtQTJqcE5PN1pJUklzUmFUMHkxbFBOLVZEdWJxOHBFZmhyX1B0ZVM0
          T21zNGNvLWxiOUJNR1BmeEp0cVRDUWJGSnJLTFBDbDJKNk5VVGtnTy0zVlgzUkIw
          Rk9uUmdWRl96aVQ0amF1UFFxZDRsQ3BOME12S2RCb29BWGtYNzkxU2JjbmJoMUhS
          cFlhblNVZENiZmZ5MGg3NTJ1MldmV3JjSUVDa29GWkRqelUtbG04bnF0OTZnS1ZH
          OFpHejF1bGhIZzVfVVIxVnlZNElROVBjNHVCTk1yazRUVThNRjF6R1RsalhnT25t
          UVZld0tPWmtkcFB4eWhGdExtalUzeWtoejBKMm9oMzJpaVhpbHFFeVQyQjM2Nzhz
          NHF3NHNkalNTdXdCWXZZZmtRNl80SnIzNnNrUm5CQ1ZLa21mdlFtbld4S3NheWQ2
          WnRPcllwZ1hjTlRfOUpYXzE5RFZiekdYbW1zTExONzRPQWg0QjZUVXciLCJ0YWci
          OiJnb1QzT0V0XzZ5eVJXdEtPdFJpUTdBIn0",
   "signature":
      "a9idqlHPlsfvg7IEKECf_XNeKKpU5jioicQTxvpQywpT8DM5M7sy0PfrOSEL4iKJ
       hGd32xQ-JwaIgT5RR0QwAM3cy15y0IMh-KkHh1H3Amsy2RIsi2jkZVpIN24OF0T4
       Wf6fXfuuXjBydT-PjIjQako2Lsic-nKAJukm0f3uwHxGXN3JGlIzi4_QrYeRjI4V
       b9ymaZW6soF1VaDqXulfaL7RtN_B46-Yg_N8XK5XWuIW5PZ1ZEkYFaqGzPaoMi31
       ipA1vx-VlQdVxCATOx2uQOT-um6NJAlvAigyWn45CRM5ETXja_V8WkpDPlHH4A5G
       SEzKUGz3tYC85Agd0-Aoig"
},
"dac_request_dest_certificate": {
   "kty": "RSA",
   "kid": "cdmi.hos-a.fr_encrypt_public",
   "key_ops": [
     "wrapKey",
     "unwrapKey",
     "encrypt",
     "decrypt"
   ],
   "n":
      "uL7ANgD80H5sNqo3nHzovPRxgncQLhz0oQvGMVvULCkrYXMaXZ5sNv7fT6UdMSZi
       T-e0sthapmEqrpeV9RKHsiF3COGl2YndUHixpEkHp8y1ggcH6iTzoBsgXMZ7OLW-
       mJ2RW3rodT7k-tcozYYsTSM5egMPQSAKgt0nMnPmdNRnEyA2_NJ8Y7lNkEXyja0Q
       JLstzkP8-cKS0BkEquLQEMbZVRM6U5uG69cj1i9OWvuRzPoaATKyt6Cc4f6PUu9L
       OyCBUAs9dXsRrt3B8H1qe7io7FAAcOpcUDKDnLFXS1THc37DK_zEyKZcMttjCvEl
       Ovt-cIaokdnxJeggv9AFGQ",
```


(continues on next page)

(continued from previous page)


```
      "e": "AQAB",
      "x5c": [
        "MIIDMDCCAhigAwIBAgIBBDANBgkqhkiG9w0BAQsFADBCMQswCQYDVQQGEwJubDER
         MA8GA1UEChMIbGllc2RvbmsxDTALBgNVBAsTBGNkbWkxETAPBgNVBAMTCHJzYS1y
         b290MCAXDTE2MTAyNzEyNDUwMFoYDzk5OTkxMjMxMjM1OTU5WjA1MQswCQYDVQQG
         EwJmcjEOMAwGA1UEChMFaG9zLWExFjAUBgNVBAMTDWNkbWkuaG9zLWEuZnIwggEi
         MA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQC4vsA2APzQfmw2qjecfOi89HGC
         dxAuHPShC8YxW9QsKSthcxpdnmw2/t9PpR0xJmJP57Sy2FqmYSqul5X1EoeyIXcI
         4aXZid1QeLGkSQenzLWCBwfqJPOgGyBcxns4tb6YnZFbeuh1PuT61yjNhixNIzl6
         Aw9BIAqC3Scyc+Z01GcTIDb80nxjuU2QRfKNrRAkuy3OQ/z5wpLQGQSq4tAQxtlV
         EzpTm4br1yPWL05a+5HM+hoBMrK3oJzh/o9S70s7IIFQCz11exGu3cHwfWp7uKjs
         UABw6lxQMoOcsVdLVMdzfsMr/MTIplwy22MK8SU6+35whqiR2fEl6CC/0AUZAgMB
         AAGjPDA6MAwGA1UdEwEB/wQCMAAwHQYDVR0OBBYEFBAIGICMR5H6KLKM1ZAEhCCc
         KwE9MAsGA1UdDwQEAwIEMDANBgkqhkiG9w0BAQsFAAOCAQEAANYSSryUU6112pYM
         r83M3GWnjzu16B+4KgimZ8kbey94zNPdwmwQdSe0Xmg+1Otc6VUB40ouNnwK8efB
         aWBtXwCA7Nb715nTqo2+rn+X+A0mGrYaKkToPEe8ZYwDcOlOpNC9JFE+QgP9/CJa
         AaWrf95W+4kra2WnA4Bhqu2WWXnQkL47/nKcGVZgQAH+mVnxPaI0gELYdonXU/S2
         8HqxoyjpGL/vmyc46zUbxYsgx/jiE7J0fJVP6Yk/3dlNYCCpLtV8VmzFAQAeCcn8
         AWowFcd09a4SY09rn1MUv/rrvXpzflfn9j7PtRRFj2e/KhitmOH1zKDuYzREpUOu
         TDlPIQ==",
        "MIIDQDCCAiigAwIBAgIBATANBgkqhkiG9w0BAQsFADBCMQswCQYDVQQGEwJubDER
         MA8GA1UEChMIbGllc2RvbmsxDTALBgNVBAsTBGNkbWkxETAPBgNVBAMTCHJzYS1y
         b290MCAXDTE2MTAyNzEyNDQwMFoYDzk5OTkxMjMxMjM1OTU5WjBCMQswCQYDVQQG
         EwJubDERMA8GA1UEChMIbGllc2RvbmsxDTALBgNVBAsTBGNkbWkxETAPBgNVBAMT
         CHJzYS1yb290MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAsrUj46dx
         5ojlaZk7YtOL6e+Q6JoG7gVMaXkJn1Szlx9ND/8w4PeO1SQ2skukdOHAlQRmxdft
         zhccNTM5hmbcn8TAfWSYqQF1R7s78bVjtmat6AQP1vSgiyZ8Ak+iYZEq3c2zVyYQ
         HKKxWxmFZt1HT8/H/B3bXveXQcERKE+Tq66h8pqVcocQUtzRFsEYmv0bR1rghtoq
         H8nhB5xnebgVlXjApW+et2SE7r6Fjv1aAbGI89ouJ1gsMPeX56P8AUjacFtNKc44
         Obu6HRXY/jm6f2m1EUm84EUsJ+9b5+S2x4qPttJDfSCasWYYz4mFJ8MwmFiBGUwf
         geT2bUm6t7qqbQIDAQABoz8wPTAPBgNVHRMBAf8EBTADAQH/MB0GA1UdDgQWBBR+
         tEB2udkEXxX0k15GztF/4ol03jALBgNVHQ8EBAMCAQYwDQYJKoZIhvcNAQELBQAD
         ggEBAIjx1f9rJ2B+mDSA3L2GRhjrPRjfI6Un3Z51CeW9gO9PMQ5ws5pDJyB79dE/
         Q8Uf1e8pZyjchTsRa8GRdnKyndN2imayOVUvPoTd3/ZSmfkurcbj3I4VW8sjHP7C
         E8fmUS8Xprdpo2SxV7oneJC0vt5eyh8mgfJ/qSbwVaiXuH1Wxi6duAvdxddMXAxQ
         KPG1KKVM7CYfCdpX/HagCOHzcto+374zFqqnQ1Kx5rbgvxNSgm/PDDOMwP03+bbT
         R63KSK1VbdtLBuS4jgaPabwyxQz/FciwTu/HLOQn8TNqDWyoIbs+eQX2Mds2Apul
         8XH2+CakjBLMLL3Tlj2x+6tKR9o="
        ]
      },
   "dac_request_dest_uri": "https://cdmi.hos-a.fr:9001/dac/"
}
```


The DAC provider at hospital A will retrieve the signing key from the JOSE protected header, validate it using the included X.509 certificates, and then verify/decrypt. It creates the following (plain) DAC response. Note that it included the object decryption key.


```
{
   "dac_response_version": "1",
   "dac_response_id": "73da04e1-2182-447e-8342-f4b9f06ec936",
   "dac_applied_mask": "ALL_PERMS",
   "dac_object_key": {
      "kty": "oct",
      "kid": "encryption_key_1",
      "use": "enc",
      "alg": "A256KW",
      "k": "1mk_8n9GZJTLDUEuxBYT-9GO8bC_fR2qqt03rVSRFak"
   },
   "dac_response_headers": {
      "CDMI-DAC-AuthInfo": "Key successfully obtained from KMS."
   },
   "dac_key_cache_expiry": "2017-04-06T14:42:47.393Z",
   "dac_response_cache_expiry": "2017-04-06T14:42:47.393Z"
}
```


As before, DAC response will be JWE encrypted using the key in server_identity. The result will be:


```
{
```


(continues on next page)

(continued from previous page)


```
  "dac_response": {
     "protected":
        "eyJraWQiOiJjZG1pLmhvcy1hLmZyX3NpZ25fcHJpdmF0ZSIsImp3ayI6IntcImt0
         eVwiOlwiUlNBXCIsXCJraWRcIjpcImNkbWkuaG9zLWEuZnJfc2lnbl9wcml2YXRl
         XCIsXCJrZXlfb3BzXCI6W1widmVyaWZ5XCIsXCJzaWduXCJdLFwiblwiOlwiblQ2
         Rjc1aVZGZkdQLXdramdneVhacFdvRWhxNmZTWkNfRENTYXNLLUdVdXdWaUhETUJG
         cXVreUM4ZmsxajNBY0JyTTlIZERIbnFrVzRfM2YzOVdZMlMtRFY5YjhkWWpRRHJl
         TDFfcHBNRVg2enJnN1hBWEJJa1ViT2h1dXJOTVNBNENlQXlCY2xqWUJpQ3dvTWxv
         aUNGb2RsbFYzUDZwekVLNjduTHNfYWVfTHVaUmRzaFhXakotYm9qQzZiNGhJcWVx
         UFh0dXBPekl5MDBKLVZydHNGUTBZaDN2dWRmbVVJTkZSTFl6c1lzY1Y3TVR4TUd5
         S1hIMXE0ekdHSHZmSlpTS0MxTEtYYk9Dc3JhaHVrUFVhY0tBSFBqbTNKaHpONTR6
         amVYTGxnaDZPT2x4X2Eza1V4YlBFTl8zYjhHTDhOVUhuWFljcUNjVDVXUUV2ZUZw
         OFlRXCIsXCJlXCI6XCJBUUFCXCIsXCJ4NWNcIjpbXCJNSUlETURDQ0FoaWdBd01C
         QWdJQkFqQU5CZ2txaGtpRzl3MEJBUXNGQURCQ01Rc3dDUVlEVlFRR0V3SnViREVS
         TUE4R0ExVUVDaE1JYkdsbGMyUnZibXN4RFRBTEJnTlZCQXNUQkdOa2JXa3hFVEFQ
         QmdOVkJBTVRDSEp6WVMxeWIyOTBNQ0FYRFRFMk1UQXlOekV5TkRVd01Gb1lEems1
         T1RreE1qTXhNak0xT1RVNVdqQTFNUXN3Q1FZRFZRUUdFd0ptY2pFT01Bd0dBMVVF
         Q2hNRmFHOXpMV0V4RmpBVUJnTlZCQU1URFdOa2JXa3VhRzl6TFdFdVpuSXdnZ0Vp
         TUEwR0NTcUdTSWIzRFFFQkFRVUFBNElCRHdBd2dnRUtBb0lCQVFDZFBvWHZtSlVW
         OFkvN0NTT0NESmRtbGFnU0dycDlKa0w4TUpKcXdyNFpTN0JXSWNNd0VXcTZUSUx4
         K1RXUGNCd0dzejBkME11ZXFSYmovZC9mMVpqWkw0Tlgxdngxa U5BT3Q0dlgrbWt3
         UmZyT3VEdGNCY0VpUlJzNkc2NnMweElEZ0o0RElGeVdOZ0dJTENneVdpSUlXaDJX
         VlhjL3FuTVFycnVjdXo5cDc4dTVsRjJ5RmRhTW41dWlNTHB2aUVpcDZvOWUyNms3
         TWpMVFFuNVd1MndWRFJpSGUrNTErWlFnMFZFdGpPeGl4eFhzeFBFd2JJcGNmV3Jq
         TVlZZTk4bGxJb0xVc3BkczRLeXRxRzZROVJwd29BYytPYmNtSE0zbmpPTjVjdVdD
         SG80NlhIOXJlUlRGczhRMy9kdndZdncxUWVkZGh5b0p4UGxaQVM5NFdueGhBZ01C
         QUFHalBEQTZNQXdHQTFVZEV3RUIvd1FDTUFBd0hRWURWUjBPQkJZRUZNWThmOW9h
         aXhYTkFWSW1Nby9heE9kN205clFNQXNHQTFVZER3UUVBd0lIZ0RBTkJna3Foa2lH
         OXcwQkFRc0ZBQU9DQVFFQWJPejR5akdLektpM0pVQTJ0WVVOeFhoQ1RKbnBVUGZB
         WDhRdGs5ZEJXTzZhVXNsSzJlTkh1TzZjL2tKMG0vKzR4LzJiczJlU0dISmRldU9w
         dzd3QmQwTDB6L1hUeStFVHRQM0k1TEovYzhNdmN6NTQyZCtMQUpXdlZtU1hIQU82
         QjZRZDJtSkJ5aGFMU3k2YWUxK3BwcmE5Yk1aYjFTVWVxc3ZtZ1R5Z2h2Y3NkVzky
         TUhBUHlxU1hxTmthaTNhNW5kbE14YjdqcWFhMzNCZkxTOHhVcE1FQ0wvaGJ4L0dH
         WXcxZXIzSUFCVDBCeG9JQXp0c0tHQTFUOWh2cnA4NCtFNjJhcnZrU1lHUk8zQmpu
         SGRHcGdnRUhRMmhGNkR1UFIvRGVyV2d2R0MyREN0cFNxVzJTTklaUFprb2tsZ3pE
         QnVXdVlHYjl4d1RWa0psWVRpSzFOdz09XCIsXCJNSUlEUURDQ0FpaWdBd0lCQWdJ
         QkFUQU5CZ2txaGtpRzl3MEJBUXNGQURCQ01Rc3dDUVlEVlFRR0V3SnViREVSTUE4
         R0ExVUVDaE1JYkdsbGMyUnZibXN4RFRBTEJnTlZCQXNUQkdOa2JXa3hFVEFQQmdO
         VkJBTVRDSEp6WVMxeWIyOTBNQ0FYRFRFMk1UQXlOekV5TkRRd01Gb1lEems1T1Rr
         eE1qTXhNak0xT1RVNVdqQkNNUXN3Q1FZRFZRUUdFd0p1YkRFUk1BOEdBMVVFQ2hN
         SWJHbGxjMlJ2Ym1zeERUQUxCZ05WQkFzVEJHTmtiV2t4RVRBUEJnTlZCQU1UQ0hK
         ellTMXliMjkwTUlJQklqQU5CZ2txaGtpRzl3MEJBUUVGQUFPQ0FROEFNSUlCQ2dL
         Q0FRRUFzclVqNDZkeDVvamxhWms3WXRPTDZlK1E2Sm9HN2dWTWFYa0puMVN6bHg5
         TkQvOHc0UGVPMVNRMnNrdWtkT0hBbFFSbXhkZnR6aGNjTlRNNWhtYmNuOFRBZldT
         WXFRRjFSN3M3OGJWanRtYXQ2QVFQMXZTZ2l5WjhBaytpWVpFcTNjMnpWeVlRSEtL
         eFd4bUZadDFIVDgvSC9CM2JYdmVYUWNFUktFK1RxNjZoOHBxVmNvY1FVdHpSRnNF
         WW12MGJSMXJnaHRvcUg4bmhCNXhuZWJnVmxYakFwVytldDJTRTdyNkZqdjFhQWJH
         STg5b3VKMWdzTVBlWDU2UDhBVWphY0Z0TktjNDRPYnU2SFJYWS9qbTZmMm0xRVVt
         ODRFVXNKKzliNStTMng0cVB0dEpEZlNDYXNXWVl6NG1GSjhNd21GaUJHVXdmZ2VU
         MmJVbTZ0N3FxYlFJREFRQUJvejh3UFRBUEJnTlZIUk1CQWY4RUJUQURBUUgvTUIw
         R0ExVWREZ1FXQkJSK3RFQjJ1ZGtFWHhYMGsxNUd6dEYvNG9sMDNqQUxCZ05WSFE4
         RUJBTUNBUVl3RFFZSktvWklodmNOQVFFTEJRQURnZ0VCQUlqeDFmOXJKMkIrbURT
         QTNMMkdSaGpyUFJqZkk2VW4zWjUxQ2VXOWdPOVBNUTV3czVwREp5Qjc5ZEUvUThV
         ZjFlOHBaeWpjaFRzUmE4R1Jkbkt5bmROMmltYXlPVlV2UG9UZDMvWlNtZmt1cmNi
         ajNJNFZXOHNqSFA3Q0U4Zm1VUzhYcHJkcG8yU3hWN29uZUpDMHZ0NWV5aDhtZ2ZK
         L3FTYndWYWlYdUgxV3hpNmR1QXZkeGRkTVhBeFFLUEcxS0tWTTdDWWZDZHBYL0hh
         Z0NPSHpjdG8rMzc0ekZxcW5RMUt4NXJiZ3Z4TlNnbS9QRERPTXdQMDMrYmJUUjYz
         S1NLMVZiZHRMQnVTNGpnYVBhYnd5eFF6L0ZjaXdUdS9ITE9RbjhUTnFEV3lvSWJz
         K2VRWDJNZHMyQXB1bDhYSDIrQ2FrakJMTUxMM1RsajJ4KzZ0S1I5bz1cIl19Iiwi
         YWxnIjoiUlMyNTYifQ",
     "payload":
        "eyJwcm90ZWN0ZWQiOiJleUpyYVdRaU9pSmpaRzFwTG1odmN5MWlMblZ6WDJWdVkz
         SjVjSFJmY0hWaWJHbGpJaXdpWVd4bklqb2lVbE5CTFU5QlJWQWlMQ0psYm1NaU9p
         SkJNalUyUjBOTkluMCIsImVuY3J5cHRlZF9rZXkiOiJLZ1lXVDVDOVBiN0FEOVZH
         bTczSDRQSkFJUHNnbTN2ampNbmZJcXZBOEFLOEtNREowMTlNQkVHeC1tTUpIQTBo
         dGhyb1F0UXA3NklDR09GLXJaNnBxajNTYm1iNFNUc3FhUjQ5aTN2Y2x1dUhRSlZM
         WVNOVklWOHBJYzZLbllDWFNGMzNldHlIZ0p0bFZjXzE2RTIyUGxEVE9IWjhPRzZz
         dklWa1o1cktnLVNGS1ZZbVlWeC05UC04NU1ueThRMkYtR3VBcDNiOHVZZzhFNXdx
```


(continues on next page)

(continued from previous page)


```
          Z1AwY2NHYVNjZF82cFUzdk83aUZGWlVLLV8ycHlzQWdRMlo3Zkl0a0lSWnVPdFFf
          RkpNaThLcFdKc3lMNFNDanZSNkxFSVBIekdTMUtkTmNEVTVrU28yc3pkanQ0WENH
          Sk9WSkc4dFhfWmZRTjh6RzdpTktJcXhiTVYwRUZVOWRjUzBkeWciLCJpdiI6Im9n
          Z3hKM3g4YjBhb2xlSDQiLCJjaXBoZXJ0ZXh0IjoiLWhkM1A3UGM5emtpQ0hhYnda
          ZVhWU29rYmNXcmdqQ1QyU0FZaFFGYzNMNFY4b2ZLYVNYR3UtazF3T014OHhZY1FH
          Slo4Mk12b2tEUXNGRnZaQjBNUG9hdEljQWtmTms0VURqREcxNi1zVGc3QURKeV9p
          TkdmZ1dkUy1XU3QxWW1PZW5HaXBYNEZWM3ZyNU9URlE4X3hCRHdYSHRiU05KX3Jy
          RlVWX2R3TnJQdUdiSlYzSXhOYi01RFJ1Rkl1Z0ptY0JXelJYRW1LeUEyZEdZTG1V
          eEF3WFpWT18wYm1KVkxzZzlKa1puczU3YXNqR1Fna0c4ZzhCNDlnTEl0QWRFQXpJ
          SFlrNTVpUkRvQk9WRnhGR284Vlc1ZFNrM05ldi1mZ1JxYzFDZEdodEZuTnlzUjUy
          akhtMFprMzRoeTJRLWZVNnNTVXBwMU5vVmw3YTd1NWMxdEVZSGRNVU9fWWtHV0g0
          dFIxeWdmbG94dFY3ODRMYXdObGNCYlV6NFpvWXFKeXg2amEzdGxHaUpxaWdLZ043
          WENTT0R6RXF0QVRQeXBITnVfYXJHRm9LNmk3cGdKZXcwYUxMQWJQOFA0SFJHTVZ6
          Q1V6dktDbDdKYS1XTE81U1UwS0ViLVhJZGdHX195a005WTJqUVQzdi1MUjNNRG1q
          WlE0ZDM4cG9BS1lfSHJCWDg1Z1dMQXFzY3MyRHhwRTlIZm9xaXk0T2tscVlaa3Js
          MUpNT1dTQVZkWWNqckhjc3Y3VjZGSUJGV2Vjc1Nwb185ZUsxOCIsInRhZyI6IkJ6
          NFZZeHdtNFkyU1Y4bDVUckMwUEEifQ",
      "signature":
         "RjUCI3Q_zfBJyeHjYfldsd6MppSDNUAIzC77lsbM1MiKfLDi8oN0999gpByS7Sx6
          kCXqsNkV4T0z_qaqy4UY9JrdjMRTNFPXJMwhqbBem-s6dJT6VquF3GBQTU8wb4OK
          5E8rGvTcWw-Hd0SfpjGoJtgv5RmpzfVgdvANZcJfST-rOra3EnPitOf8dJ95Db3t
          78mEbMfqdoobk1Dnc39DvpnzD61TIoxWoZj3UGcBcvNPsl2XijS6yZlgAsrQbGnX
          xWx-PCwEACZoVekzt-YV5QUFH2Jqb1pGeUUCwoFv1ON90iXVusIdWWnJO51gSKwg
          i80ZxOSSBwF6b9WIeXHe5g"
   },
   "dac_response_dest_certificate": {
      "kty": "RSA",
      "kid": "cdmi.hos-b.us_encrypt_public",
      "key_ops": [
         "wrapKey",
         "unwrapKey",
         "encrypt",
         "decrypt"
      ],
      "n":
         "oQMQkY85UzwO7K6H0QQNfAiRMN3ZfhK0aXEKx7YwvrCU9IKOquZ1OYZ9Cv8556_8
          E8yZm02JDWOBOaSsGHU835jvXf12f4MywKGWj5FtIGL-j9kXF6SWq3zuLVYlXpMI
          KsJngHMVFca_-ZhZ2vLsrnDR1aCNEC48gR26ewp6WX1ptnSc1W4x3Mj-ONMVzxVE
          7XNlwYysTgDtonmTQD-YG6_KhhAPx0YowMbUWv_cMQvXsi7MMDyZn6fxfq4zQmQ2
          V5RtUy5msd6K3beDzS4LmZhsJmjU7YnhOj0pZby4Zckm43npjXPAuwPhzK2OW7qb
          fkv0qm4rsFWUcuNh8lBsDw",
      "e": "AQAB",
      "x5c": [
         "MIIDMDCCAhigAwIBAgIBBTANBgkqhkiG9w0BAQsFADBCMQswCQYDVQQGEwJubDER
          MA8GA1UEChMIbGllc2RvbmsxDTALBgNVBAsTBGNkbWkxETAPBgNVBAMTCHJzYS1y
          b290MCAXDTE2MTAyNzEyNDUwMFoYDzk5OTkxMjMxMjM1OTU5WjA1MQswCQYDVQQG
          EwJ1czEOMAwGA1UEChMFaG9zLWIxFjAUBgNVBAMTDWNkbWkuaG9zLWIudXMwggEi
          MA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQChAxCRjzlTPA7srofRBA18CJEw
          3dl+ErRpcQrHtjC+sJT0go6q5nU5hn0K\/znnr\/wTzJmbTYkNY4E5pKwYdTzfmO
          9d\/XZ\/gzLAoZaPkW0gYv6P2RcXpJarfO4tViVekwgqwmeAcxUVxr\/5mFna8uy
          ucNHVoI0QLjyBHbp7CnpZfWm2dJzVbjHcyP440xXPFUTtc2XBjKxOAO2ieZNAP5g
          br8qGEA\/HRijAxtRa\/9wxC9eyLswwPJmfp\/F+rjNCZDZXlG1TLmax3ordt4PN
          LguZmGwmaNTtieE6PSllvLhlySbjeemNc8C7A+HMrY5bupt+S\/SqbiuwVZRy42H
          yUGwPAgMBAAGjPDA6MAwGA1UdEwEB\/wQCMAAwHQYDVR0OBBYEFH7NJvMIfTQtZn
          nyiIdLNkjCgwSIMAsGA1UdDwQEAwIEMDANBgkqhkiG9w0BAQsFAAOCAQEAdiADIv
          0v09SUDcPL+BKysvchn\/Sgx5KBu7n9KFwE31Dhx2zvT6ruL8kXdekPH9cfrDafW
          6I\/vnbzAVj02i5pM2cHayj13fTOWSVwpcQuvkoIF9eVIWONkemMMf7M7jpTw07z
          7S2T5usaDmMNpqj8y5pRpQo3PnBVxpEZJ0XaSdfuiHtVLDq8gDZCq6Hc2tt7JM3W
          njnQgs+1lSGRuqWOcpmVONIoqvhiolNDZV35Z7puRwqck1N2f1qyHHGBWXfCN9U4
          ci6q1BnWBIFV+hURge8NSbpqawolaNueUbTcKjN3JsMC4ZxhMF9rN3uuPn+UAYkA
          yQkcSmGSMM07wcAkMg==",
         "MIIDQDCCAiigAwIBAgIBATANBgkqhkiG9w0BAQsFADBCMQswCQYDVQQGEwJubDER
          MA8GA1UEChMIbGllc2RvbmsxDTALBgNVBAsTBGNkbWkxETAPBgNVBAMTCHJzYS1y
          b290MCAXDTE2MTAyNzEyNDQwMFoYDzk5OTkxMjMxMjM1OTU5WjBCMQswCQYDVQQG
          EwJubDERMA8GA1UEChMIbGllc2RvbmsxDTALBgNVBAsTBGNkbWkxETAPBgNVBAMT
          CHJzYS1yb290MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAsrUj46dx
          5ojlaZk7YtOL6e+Q6JoG7gVMaXkJn1Szlx9ND\/8w4PeO1SQ2skukdOHAlQRmxdf
          tzhccNTM5hmbcn8TAfWSYqQF1R7s78bVjtmat6AQP1vSgiyZ8Ak+iYZEq3c2zVyY
```


(continues on next page)

(continued from previous page)

```
            QHKKxWxmFZt1HT8\/H\/B3bXveXQcERKE+Tq66h8pqVcocQUtzRFsEYmv0bR1rgh
            toqH8nhB5xnebgVlXjApW+et2SE7r6Fjv1aAbGI89ouJ1gsMPeX56P8AUjacFtNK
            c44Obu6HRXY\/jm6f2m1EUm84EUsJ+9b5+S2x4qPttJDfSCasWYYz4mFJ8MwmFiB
            GUwfgeT2bUm6t7qqbQIDAQABoz8wPTAPBgNVHRMBAf8EBTADAQH\/MB0GA1UdDgQ
            WBBR+tEB2udkEXxX0k15GztF\/4ol03jALBgNVHQ8EBAMCAQYwDQYJKoZIhvcNAQ
            ELBQADggEBAIjx1f9rJ2B+mDSA3L2GRhjrPRjfI6Un3Z51CeW9gO9PMQ5ws5pDJy
            B79dE\/Q8Uf1e8pZyjchTsRa8GRdnKyndN2imayOVUvPoTd3\/ZSmfkurcbj3I4V
            W8sjHP7CE8fmUS8Xprdpo2SxV7oneJC0vt5eyh8mgfJ\/qSbwVaiXuH1Wxi6duAv
            dxddMXAxQKPG1KKVM7CYfCdpX\/HagCOHzcto+374zFqqnQ1Kx5rbgvxNSgm\/PD
            DOMwP03+bbTR63KSK1VbdtLBuS4jgaPabwyxQz\/FciwTu\/HLOQn8TNqDWyoIbs
            +eQX2Mds2Apul8XH2+CakjBLMLL3Tlj2x+6tKR9o="
        ]
    }
}
```

The CDMI server at Hospital B can now decrypt this message, process the access control decision, and use the object key to decrypt the encrypted object:

```
<-- HTTP/1.1 200 OK
<-- Content-Type: text/plain
<-- Content-Length: 33
<--
<-- This is an unencrypted text file.
```

Clause 25

Data object versions

25.1 Overview

Data object versioning supports multiple client use cases:

- Clients can preserve all data written to a data object over time by using versions to retain all updates made to a data object.
- Clients can control how long and much many historical versions are retained by specifying constraints in data system metadata.
- Clients can restore the contents of a historical version by copying it into the version-enabled data object.
- Clients can consistently retrieve data object values using multiple parallel or sequential transactions without worring about corruption due to concurrent updates by using the current version data object.
- Clients can detect where concurrent updates have occurred and can access any overwritten data by iterating through historical versions.
- Distributed CDMI implementations can merge concurrent changes made on different, eventually consistent nodes without resulting in data loss.

Version-enabled data objects allow the previous state of a data object to be retained when an update is performed. In a non-version-enabled data object, each update changes the state of the object and the previous state is lost. This mode of operation is shown in Fig. 20.

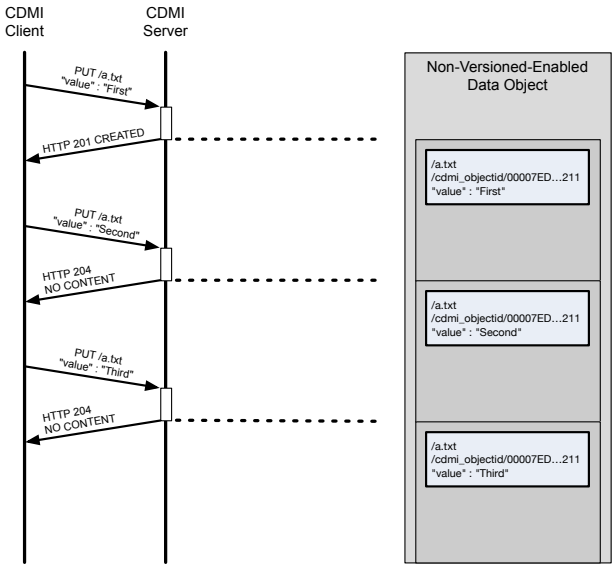


Fig. 20: Updates to a non-version-enabled data object

When a data object has versioning enabled, each update creates a new “current version” with the same contents of the version-enabled data object, with the previous current version becoming a historical version. The current version and all historical versions can be accessed by ID, and are immutable. The version-enabled data object continues to be mutable and has the same behaviors to clients as a non-version-enabled data object. This behavior is shown in Fig. 21 from the perspective of a client.

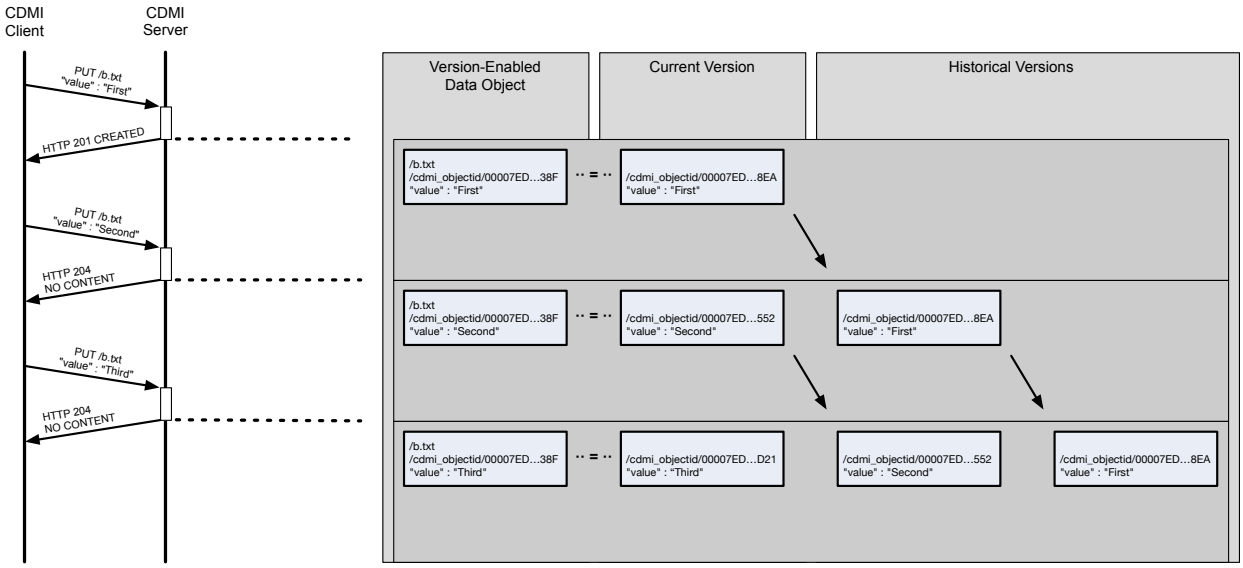


Fig. 21: Updates to a version-enabled data object

Using this approach, CDMI clients that are not aware of versioning can continue to access version-enabled data objects the same way as non-version-enabled data objects, while CDMI clients that are aware of versioning can access and manage the immutable versions associated with the version-enabled data object.

Versioning is enabled for a data object by adding a data system metadata item that indicates that versioning is desired.

Version-enabled data objects and all associated versions contain additional storage system metadata items. These metadata items allow a client to discover the versions that are associated with a version-enabled data object and to iterate through these versions.

The maximum number of versions to be retained, maximum age of versions to be retained, and the maximum space that can be consumed by versions is controlled by data system metadata.

When a data object is version enabled, it always contains at least one version, the “current version”. The current version has the same contents as the version-enabled data object but has a different identifier (URI and Object Identifier) and is immutable. When a version-enabled data object is changed, a new current version is created, and the previous current version becomes a historical version.

25.2 Traversing version-enabled data objects

Version-enabled data objects have additional metadata items that allow a client to discover and traverse historical versions.

Version-enabled data objects shall contain the following metadata items, as shown in Table 159.

Table 159: Version-enabled data object metadata items

Metadata Item Name	Type	Description	Requirement
cdmi_version_current	JSON String	The URI of the current version of the version-enabled data object. This metadata item shall be present in the version-enabled data object and all historical versions.	Conditional
cdmi_version_oldest	JSON Array of JSON Strings	One or more URIs of the oldest version(s). This metadata item shall be present in the version-enabled data object, the current version and all historical versions except the oldest historical versions.	Conditional
cdmi_version_object	JSON String	The URI of the version-enabled data object. This metadata item shall be present in the current version and all historical versions.	Conditional
cdmi_version_parent	JSON String	The URI of the previous historical version. This metadata item shall be present in the current version and all historical versions except the oldest historical versions.	Conditional
cdmi_version_children	JSON Array of JSON Strings	One or more URIs of historical versions (or the current version) created by updating a given historical version. This metadata item shall be present in all historical versions.	Conditional

Situations where a version-enabled data object or a historical data object may have multiple oldest versions or multiple children is explained in 25.3.

To visualize how these metadata items allow a client to traverse data object versions, the linkages between the version-enabled data object and data object versions in the final state of Fig. 21 is shown in Fig. 22.

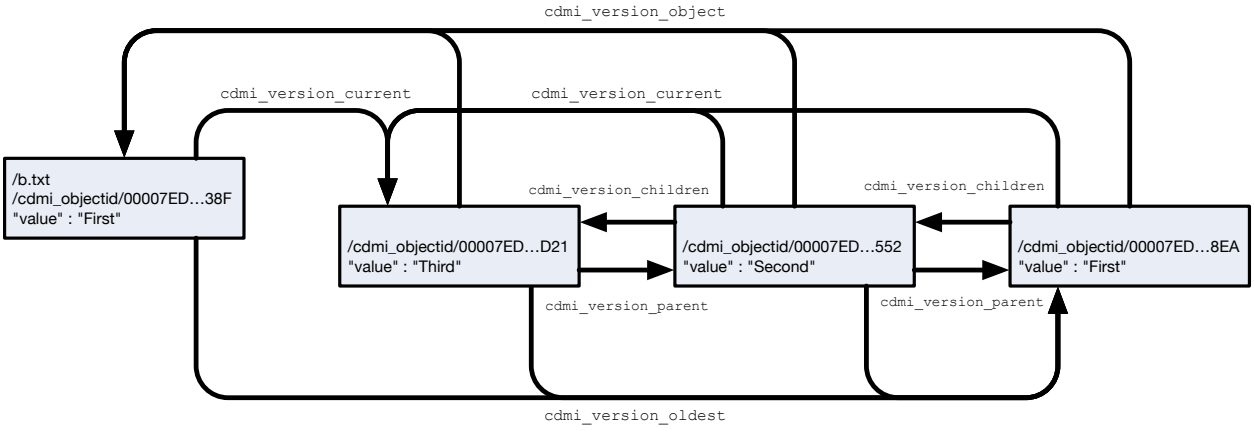


Fig. 22: Linkages between a version-enabled data object and data object versions

A client accessing the version-enabled data object (`/b.txt`) can traverse to the current version and to the oldest version.

A client accessing a data object version can traverse to the version-enabled data object, to the current version, to the parent version, to child versions, and to the oldest version.

25.3 Concurrent updates and version-enabled data objects

When multiple concurrent updates are performed against a version-enabled data object, each update is performed against the state of the object at the time the update starts. The change to the state resulting from the update to the object becomes visible to clients at the time the update completes.

Two different types of concurrent updates can occur: overlapping updates and nested updates.

Fig. 23 and Fig. 24 show the update sequence and resulting version linkages for overlapping updates:

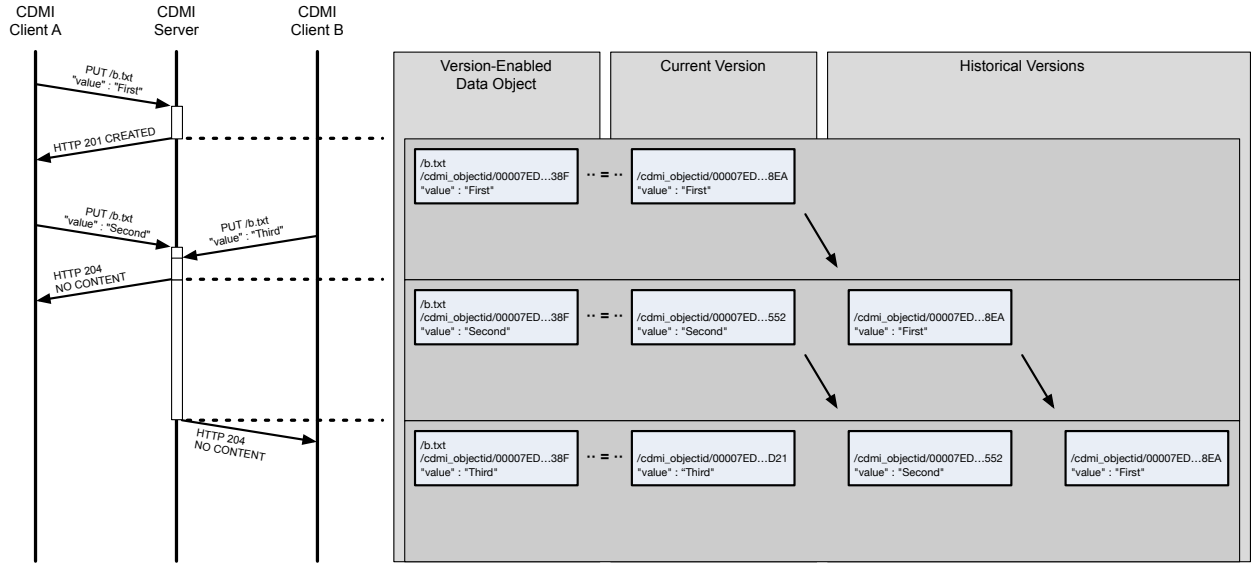


Fig. 23: Overlapping concurrent updates

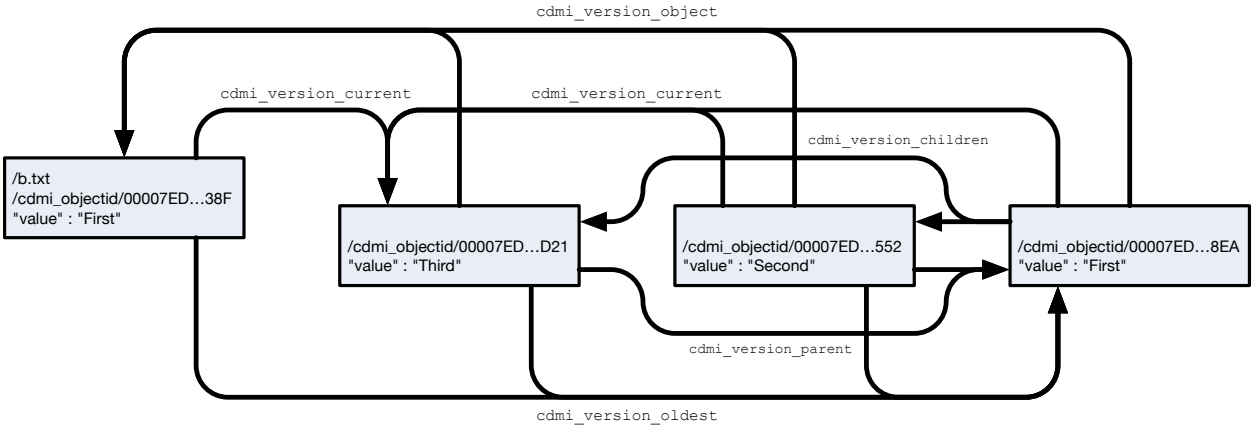


Fig. 24: Linkages for overlapping updates

In the sequence shown in Fig. 23, both the “Second” and “Third” updates are performed against the “First” state. As the “Third” update completes last, it becomes the current version. In this example, historical version `00007ED...8EA` would have two children, versions `00007ED...552` and `00007ED...D21`. Both versions `00007ED...552` and `00007ED...D21` would have the same parent `00007ED...8EA`.

4052 Fig. 25 and Fig. 26 show the update sequence and resulting version linkages for nested updates:

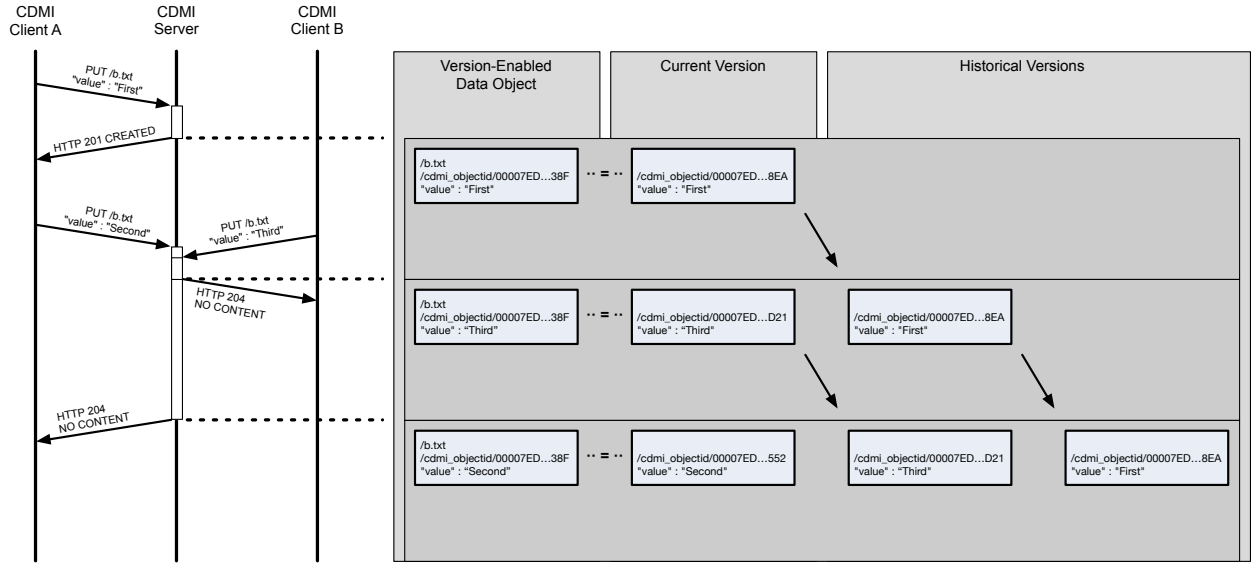


Fig. 25: Nested concurrent updates

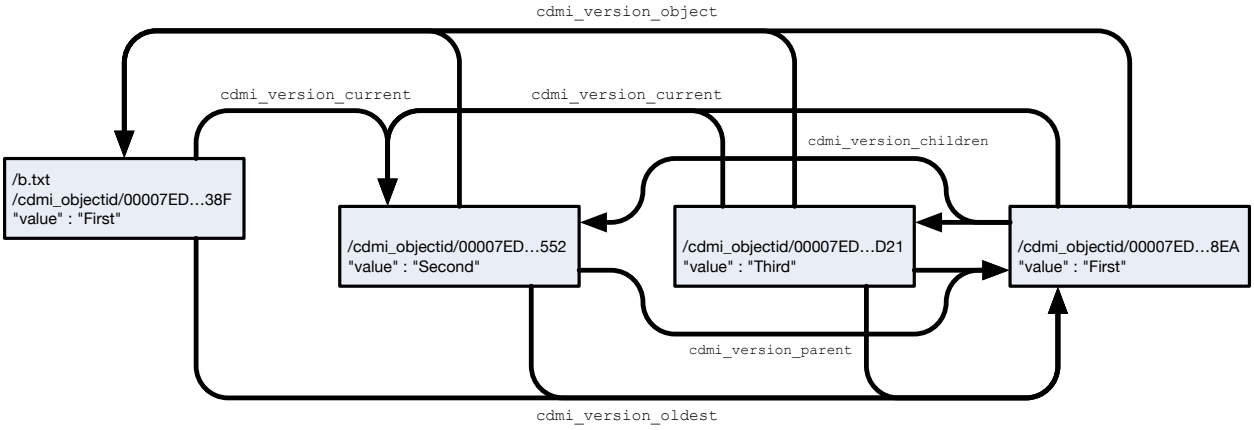


Fig. 26: Linkages for nested updates

4053 In the sequence shown in these figures, both the “Second” and “Third” updates are performed against the “First” state.
4054 As the “Second” update completes last, it becomes the current version. In this example, historical version `00007ED...8EA` would have two children, versions `00007ED...552` and `00007ED...D21`. Both versions `00007ED...552` and
4056 `00007ED...D21` would have the same parent `00007ED...8EA`.

4057 Both of these data structures are equivalent, with the only difference being which update completed last.

25.4 Capabilities for version-enabled data objects

The relationship between version-enabled data objects, data object versions, and capabilities is shown in Fig. 27.

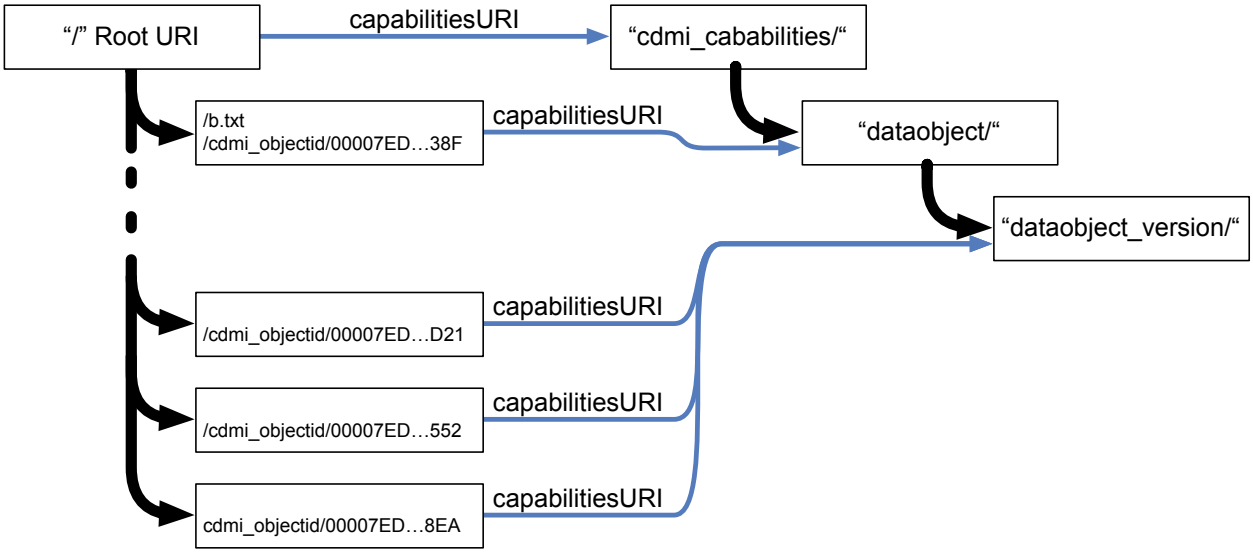


Fig. 27: Version to `capabilityURI` relationships

Data object versions are immutable but may be deleted by a client or by the system, depending on the data system metadata specified.

25.5 Updates triggering version creation

If versioning is enabled by setting the value of the `cdmi_versions` metadata item in the version-enabled data object to "`value`", the following updates will trigger the creation of a new version:

- changing the `mimetype`,
- changing the `value`, or
- changing the `valuetransferencoding`.

If versioning is enabled by setting the value of the `cdmi_versions` metadata item in the version-enabled data object to "`user`", the following updates will trigger the creation of a new version:

- changing the `mimetype`,
- changing the `value`,
- changing the `valuetransferencoding`, or
- adding, modifying, or removing user metadata.

If versioning is enabled by setting the value of the `cdmi_versions` metadata item in the version-enabled data object to "`all`", then all updates to the data object will trigger the creation of a new version.

While ACLs for historical versions are left unchanged, the effective ACL, owner, and domain of historical versions shall be the ACL, owner, and domain of the current version-enabled data object. This means that changing the ACL of a versioned data object also overrides the historical version ACL for all previous versions.

Modifications performed with the `X-CDMI-Partial` header shall not trigger the creation of a new version until the `completionStatus` is changed from "`Processing`" to "`Complete`".

25.6 Operations on version-enabled data objects

Moving a version-enabled data object within a system is considered to be an update to the `name` and/or `parentURI` fields.

Moving a version-enabled data object between systems moves all data object versions associated with the version-enabled data object and preserves all identifiers. If the destination name and/or URI are different, the move is considered to be an update to the `name` and/or `parentURI` fields.

Copying a version-enabled data object shall only copy the current version of the version-enabled data object. Versions of the version-enabled data object are not copied.

Deleting a version-enabled data object shall also delete all versions associated with that version-enabled data object.

Disabling versioning for a version-enabled data object shall preserve all versions. Previously existing versioning metadata shall remain present while versioning is disabled. Re-enabling versioning for a data object that previously was version-enabled shall result in the creation of a new current version.

If a version-enabled data object is placed under retention or hold, the retention behaviors of the version-enabled data object shall be applied to the data object versions.

No additional notifications are defined for version-enabled data objects. When a version-enabled data object is updated, an additional creation notification message shall be generated for the created data object version. Likewise, when a data object version is accessed or deleted, a notification message is generated. If a limited number, size, or age for versions is requested and a change to a version-enabled data object results in a version being automatically deleted, then the system shall generate a corresponding deletion notification message for the deleted data object version.

25.7 Operations on data object versions

A data object version is presented to the client as a standard CDMI data object.

Moving, copying over, deserializing over, and updating a data object version shall not be permitted and shall result in an HTTP status code of `403 Forbidden`.

Copying a data object version is permitted. For example, to promote a version to become the current version of a version-enabled data object, the URI of the data object version is used in the copy field when performing an update to the URI of the version-enabled data object. Updates may also be performed as part of the copy operation.

Deleting the current version or historical versions shall maintain the relationships in Table 159.

Deleting the current version shall revert the current version to the parent. If there is no parent version, deleting the current version shall result in an HTTP status code of `403 Forbidden`.

Deleting a historical version shall only be permitted when the client has ACL permissions to delete the historical version and has ACL permissions to delete the version-enabled data object.

Deleting a historical version shall use the domainURI metadata of the version-enabled data object.

Reading a historical version shall update the `cdmi_acount` and `cdmi_atime` of the historical version, when present.

Reading a historical version shall only be permitted when the client has ACL permissions to read the historical version and has ACL permissions to read the version-enabled data object.

Reading a historical version shall use the domainURI metadata of the version-enabled data object.

Standard notification messages are sent when data object versions are read or deleted.

25.8 Query of data object versions

Data object versions are regular CDMI objects, consequently they will be included in query results unless explicitly excluded.

Querying for data object versions is performed by including the scope:

```
"metadata" : {
    "cdmi_version_children" : "*"
}
```

Querying for version-enabled data objects (but not their versions) is performed by including the scope:

```
"metadata" : {
    "cdmi_versioning" : "*"
}
```

Querying for non-versioned data objects with no versions is performed by including the scope:

```
"metadata" : {
    "cdmi_version_current" : "!*"
}
```

Querying for non-versioned data objects with versions is performed by including the scope:

```
"metadata" : {
    "cdmi_versioning" : "!*",
    "cdmi_version_current" : "*"
}
```

25.9 Version-enabled data object serialization

Version-enabled data objects are serialized by performing the following steps:

- Serialize the current version and all historical versions as described in 15.2.
- Place the serialized current version and historical versions into a JSON Array.
- Serialize the version-enabled data object as described in 15.2.
- Replace the value field in the serialized version-enabled data object with the JSON Array containing the serialized current version and historical versions.

Serializing a non-version-enabled data object that has versions shall follow the same process.

EXAMPLE 1: A version-enabled data object with two historical versions is serialized.

```
{
    "objectType" : "application/cdmi-object",
    "objectID" : "00007ED900100DA32EC94351F8970400",
    "objectName" : "MyVersionedDataObject.txt",
    "parentURI" : "/MyContainer/",
    "parentID" : "00007E7F00102E230ED82694DAA975D2",
    "domainURI" : "/cdmi_domains/MyDomain/",
    "capabilitiesURI" : "/cdmi_capabilities/dataobject/",
    "completionStatus" : "Complete",
    "mimetype" : "text/plain",
    "metadata" : {
        "cdmi_size" : "33",
        "cdmi_versioning" : "user",
        "cdmi_version_object" : "/cdmi_objectid/00007ED900100DA32EC94351F8970400",
        "cdmi_version_current" : "/cdmi_objectid/00007ED90010F077F4EB1C99C87524CC",
        "cdmi_version_oldest" : [
            "/cdmi_objectid/00007ED90010512EB55A9304EAC5D4AA"
        ],

        ...
    },
    "value" : [
        {
            "objectType" : "application/cdmi-object",
            "objectID" : "00007ED90010F077F4EB1C99C87524CC",
            "objectName" : "MyVersionedDataObject.txt",
            "parentURI" : "/MyContainer/",
            "parentID" : "00007E7F00102E230ED82694DAA975D2",
            "domainURI" : "/cdmi_domains/MyDomain/",
            "capabilitiesURI" : "/cdmi_capabilities/dataobject/dataobject_version/",
            "completionStatus" : "Complete",
            "mimetype" : "text/plain",
            "metadata" : {
                "cdmi_size" : "33",
                "cdmi_version_object" : "/cdmi_objectid/
↪00007ED900100DA32EC94351F8970400",
                "cdmi_version_current" : "/cdmi_objectid/
↪00007ED90010F077F4EB1C99C87524CC",
                "cdmi_version_oldest" : [
                    "/cdmi_objectid/00007ED90010512EB55A9304EAC5D4AA"
                ],
                "cdmi_version_parent" : "/cdmi_objectid/
↪00007ED9001005192891EEBE599D94BB",
                "cdmi_version_children" : [ ],

                ...
            },
            "valuerange" : "0-32",
            "valuetransferencoding" : "utf-8",
            "value" : "Third version of this Data Object"
        },
        {
            "objectType" : "application/cdmi-object",
```

(continues on next page)

(continued from previous page)

```
            "objectID" : "00007ED9001005192891EEBE599D94BB",
            "objectName" : "MyVersionedDataObject.txt",
            "parentURI" : "/MyContainer/",
            "parentID" : "00007E7F00102E230ED82694DAA975D2",
            "domainURI" : "/cdmi_domains/MyDomain/",
            "capabilitiesURI" : "/cdmi_capabilities/dataobject/dataobject_version/",
            "completionStatus" : "Complete",
            "mimetype" : "text/plain",
            "metadata" : {
                "cdmi_size" : "34",
                "cdmi_version_object" : "/cdmi_objectid/
↪00007ED900100DA32EC94351F8970400",
                "cdmi_version_current" : "/cdmi_objectid/
↪00007ED90010F077F4EB1C99C87524CC",
                "cdmi_version_oldest" : [
                    "/cdmi_objectid/00007ED90010512EB55A9304EAC5D4AA"
                ],
                "cdmi_version_parent" : "/cdmi_objectid/
↪00007ED90010512EB55A9304EAC5D4AA",
                "cdmi_version_children" : [
                    "/cdmi_objectid/00007ED90010F077F4EB1C99C87524CC"
                ],

                ...
            },
            "valuerange" : "0-33",
            "valuetransferencoding" : "utf-8",
            "value" : "Second version of this Data Object"
        },
        {
            "objectType" : "application/cdmi-object",
            "objectID" : "00007ED90010512EB55A9304EAC5D4AA",
            "objectName" : "MyVersionedDataObject.txt",
            "parentURI" : "/MyContainer/",
            "parentID" : "00007E7F00102E230ED82694DAA975D2",
            "domainURI" : "/cdmi_domains/MyDomain/",
            "capabilitiesURI" : "/cdmi_capabilities/dataobject/dataobject_version/",
            "completionStatus" : "Complete",
            "mimetype" : "text/plain",
            "metadata" : {
                "cdmi_size" : "33",
                "cdmi_version_object" : "/cdmi_objectid/
↪00007ED900100DA32EC94351F8970400",
                "cdmi_version_current" : "/cdmi_objectid/
↪00007ED90010F077F4EB1C99C87524CC",
                "cdmi_version_oldest" : [
                    "/cdmi_objectid/00007ED90010512EB55A9304EAC5D4AA"
                ],
                "cdmi_version_children" : [
                    "/cdmi_objectid/00007ED9001005192891EEBE599D94BB"
                ],

                ...
            },
            "valuerange" : "0-32",
            "valuetransferencoding" : "utf-8",
            "value" : "First version of this Data Object"
        }
    ]
}
```

Deserializing a version-enabled data object or a non-version-enabled data object with versions shall restore the data object and all serialized versions.

Individually serializing and deserializing current versions or historical versions shall not be permitted.

Deserializing a serialized any data object with versions onto a system that does not support versions shall result in an HTTP status code of `400 Bad Request`.

Part V

CDMI Annexes

Clause 26

Extensions

26.1 Overview

CDMI extensions describe non-normative additional functionality for extending the CDMI International Standard. Each extension is first written as a standalone document that describes the changes that are required to implement the functionality being added into this International Standard.

When one or more vendors have implemented a CDMI extension, it is eligible to be added to this annex. When multiple vendors have implemented a CDMI extension and demonstrated interoperability, the extension is eligible to be merged into the CDMI International Standard itself, at which point it becomes normative.

CDMI extensions shall not break or modify existing functionality, and thus do not result in compatibility problems with existing clients. Compatibility is typically accomplished by relaxing restrictions imposed in the current CDMI International Standard, adding new fields, or using reserved names for metadata. The clients that are using CDMI capabilities can identify the functionality that is associated with these CDMI extensions.

26.2 Summary metadata for bandwidth

26.2.1 Overview

Domain summaries provide summary measurement information about domain usage and billing. Some systems may track additional usage and billing information related to network bandwidth. This extension proposes a set of additional, optional contents for domain summary objects.

26.2.2 Changes to specification

Add new terms to clause 3:

---

private network segment a single IP address or range of IP addresses that are considered internal (e.g., LAN)

public network segment a single IP address or range of IP addresses that are considered external (e.g., WAN)

Add table entries to end of Table 79 in 10.3:

Metadata name	Type	Description	Requirement
`cdmi_summary_network_bytes`	JSON string	Total number of bytes read/written to/from public/private network segments	Optional
`cdmi_summary_reads_private`	JSON string	Total number of bytes read from private network segment	Optional
`cdmi_summary_reads_private_min`	JSON string	Minimum number of bytes read from private network segment for the given interval	Optional
`cdmi_summary_reads_private_max`	JSON string	Maximum number of bytes read from private network segment for the given interval	Optional
`cdmi_summary_reads_private_avg`	JSON string	Average number of bytes read from private network segment for the given interval	Optional
`cdmi_summary_writes_private`	JSON string	Total number of bytes written to private network segment	Optional
`cdmi_summary_writes_private_min`	JSON string	Minimum number of bytes written to private network segment for the given interval	Optional
`cdmi_summary_writes_private_max`	JSON string	Maximum number of bytes written to private network segment for the given interval	Optional
`cdmi_summary_writes_private_avg`	JSON string	Average number of bytes written to private network segment for the given interval	Optional
`cdmi_summary_reads_public`	JSON string	Total number of bytes read from public network segment	Optional
`cdmi_summary_reads_public_min`	JSON string	Minimum number of bytes read from public network segment for the given interval	Optional
`cdmi_summary_reads_public_max`	JSON string	Maximum number of bytes read from public network segment for the given interval	Optional
`cdmi_summary_reads_public_avg`	JSON string	Average number of bytes read from public network segment for the given interval	Optional
`cdmi_summary_writes_public`	JSON string	Total number of bytes written to public network segment	Optional

Continued on next page

Table 160 – continued from previous page

Metadata name	Type	Description	Requirement
cdmi_summary_writes_public_min	JSON string	Minimum number of bytes written to public network segment for the given interval	Optional
cdmi_summary_writes_public_max	JSON string	Maximum number of bytes written to public network segment for the given interval	Optional
cdmi_summary_writes_public_avg	JSON string	Average number of bytes written to public network segment for the given interval	Optional
cdmi_summary_reads_total	JSON string	Total number of bytes read from both public and private network segments	Optional
cdmi_summary_writes_total	JSON string	Total number of bytes written to both public and private network segments	Optional

26.3 Expiring access control entries (ACEs)

26.3.1 Overview

A common trait of cloud storage services is the ability to share an object with other clients for a limited time. This extension adds an attribute of ACEs used in ACLs that imposes a time limit (expiration) on the ACE. Once the ACE expires, the ACE is no longer valid or included in the authorization calculation for the object.

26.3.2 Changes to specification

Insert into 17.2.7:

After the bullet item:

- ACEs that do not refer to the principal P requesting the operation are ignored.

Insert bullet:

- ACEs that have an expiration value less than the current time are ignored.

Change 17.2.7:

Original text:

```
ACE = { acetype , identifier , aceflags , acemask , acetime }
```

Revised text:

```
ACE = { acetype , identifier , aceflags , acemask , acetime, expiration }
```

Insert into 17.2.7 after "`acemask = uint_t | acemaskstring`":

```
expiration = uint_t
```

Insert into 17.2.7 after "When ACE masks…":

When ACE expiration is presented in string format, it shall be specified in ISO-8601 point-in-time format as described in 5.6.

Insert a new sub-clause after 17.2.10: "ACE expiration"

An ACE may have an optional expiration associated with it. The expiration is a point-in-time value, in ISO-8601 point-in-time format, as described in 5.6, which specifies that the ACE is no longer valid and shall be ignored after the time specified.

26.4 Group storage system metadata

26.4.1 Overview

ACLs in CDMI can refer to the owner of an object by specifying an ACE Who of `"OWNER@"`. This reference corresponds to the contents of the `cdmi_owner` storage system metadata. However, no `cdmi_group` storage system metadata corresponds to an ACE Who of `"GROUP@"`.

This extension defines a new storage system metadata item, `cdmi_group`, that allows an object to be associated with a group for ACL evaluation purposes.

26.4.2 Changes to specification

Add a new row at end of table Table 126 in 12.2.9:

Capability name	Type	Definition
`cdmi_group`	JSON string	If present and `"true"`, this capability indicates that the cloud storage system supports group storage system metadata to indicate a group associated with the object.

Add a new row below `"cdmi_owner"` in Table 141 of 16.2:

Metadata name	Type	Description	Requirement
`cdmi_group`	JSON string	The name of the group that is associated with the object.	Optional

26.5 Header-based metadata

26.5.1 Overview

The CDMI protocol enables CDMI-aware clients to store and retrieve structured metadata using JSON bodies, but does not permit HTTP-based clients to access this metadata. This extension extends CDMI metadata to permit HTTP header metadata to be stored and retrieved as a subset of CDMI metadata.

Due to limitations associated with HTTP headers, certain restrictions must be placed on metadata that is accessible via headers.

26.5.2 Changes to specification

Add a new row at end of table Table 9 in 6.2:

Header	Type	Description	Requirement
x-*-meta-*	Header string	If the “cdmi_header_metadata” capability is present, for each request header matching the pattern `"x-*-meta-*"`, a new user metadata item shall be created, with the metadata name set to the header field-name, and the metadata value set to the header field-value. If the number of headers, the length of any of the headers, or the total size of the headers exceeds the limits specified in RFC 2616, or specified by the `cdmi_header_metadata_maxitems`, `cdmi_header_metadata_maxsize`, or the `cdmi_header_metadata_maxtotalsize` capabilities, a `400 Bad Request` shall be returned to the client.	Conditional

Add new example at end of 6.2:

EXAMPLE 1: PUT to the container URI the data object name, contents, and metadata:

```
--> PUT /cdmi/2.0.0/MyContainer/MyDataObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: text/plain;charset=utf-8
--> X-CDMI-Meta-Colour: Yellow
--> X-Object-Meta-Shape: Square
--> Content-Length: 37
-->
--> This is the Value of this Data Object

<-- HTTP/1.1 201 Created
```

After 6.2, add a new clause “Inspect a Data Object using HTTP”:

26.5.3 Synopsis

To check for the presence of a data object, the following request shall be performed:

- `HEAD <root URI>/<ContainerName>/<DataObjectName>`

Where:

- `<root URI>` is the path to the CDMI cloud.
- `<ContainerName>` is zero or more intermediate containers that already exist, with one slash (i.e., `"/"`) between each pair of container names.
- `<DataObjectName>` is the name specified for the data object to be checked.

The object shall also also be able to be checked at `<root URI>/cdmi_objectid/<objectID>`.

26.5.4 Capabilities

The following capabilities describe the supported operations that may be performed when reading an existing data object:

- Support for the ability to read the metadata of an existing data object is indicated by the presence of the `cdmi_read_metadata` capability in the specified object.

26.5.5 Request headers

Request headers may be provided as per RFC 2616.

26.5.6 Request message body

A request message body shall not be provided.

26.5.7 Response headers

The HTTP response headers for checking for the presence of a data object using HTTP are shown in Table 161.

Table 161: Response headers - Inspect a data object using HTTP

Header	Type	Description	Requirement
`Content-Type`	Header string	The content type returned shall be the mimetype field in the data object.	Mandatory
`Location`	Header string	The server shall respond with the URI that the reference redirects to if the object is a reference.	Conditional
`x-\*-meta-\*`	Header string	If the `"cdmi_header_metadata"` capability is present, for each user metadata item in the “metadata” field with a metadata name that is a case-insensitive match to the pattern `"x-*-meta-*"`, a corresponding response header shall be returned to the client where the header field-name shall be the metadata item name, and the header field-value shall be the metadata item value. If a header value to be return is not conformant with RFC 2616, the server may omit the field from the response headers.	Conditional

26.5.8 Response message body

No response body shall be provded, as per RFC 2616.

26.5.9 Response status

The HTTP status codes that occur when checking the presence of a data object using HTTP are described in Table 162.

Table 162: HTTP status codes - Inspect a data object using HTTP

HTTP status	Description
`200 OK`	The queue object content was returned in the response.
`302 Found`	The resource is a reference to another resource.
`400 Bad Request`	The request contains invalid parameters or field names.
`401 Unauthorized`	The authentication credentials are missing or invalid.
`403 Forbidden`	The client lacks the proper authorization to perform this request.
`404 Not Found`	The resource was not found at the specified URI.

26.5.10 Example

EXAMPLE 1: HEAD to the data object URI to check for the presence of a data object with header metadata:

```
--> HEAD /cdmi/2.0.0/MyContainer/MyDataObject.txt HTTP/1.1
--> Host: cloud.example.com

<-- HTTP/1.1 200 OK
<-- Content-Type: text/plain
<-- Content-Length: 37
<-- X-CDMI-Meta-Colour: Yellow
<-- X-Object-Meta-Shape: Square
```

Add a new row at end of table Table 13 in 6.3:

Header	Type	Description	Requirement
`x-\*-meta-\*`	Header string	If the `"cdmi_header_metadata"` capability is present, for each user metadata item in the “metadata” field with a metadata name that is a case-insensitive match to the pattern `"x-*-meta-*"`, a corresponding response header shall be returned to the client where the header field-name shall be the metadata item name, and the header field-value shall be the metadata item value. If a header value to be return is not conformant with RFC 2616, the server may omit the field from the response headers.	Conditional

---

Add new example at end of 6.3:

EXAMPLE 6: GET to the data object URI to read the value of the data object with header metadata:

```
--> GET /cdmi/2.0.0/MyContainer/MyDataObject.txt HTTP/1.1
--> Host: cloud.example.com

<-- HTTP/1.1 200 OK
<-- Content-Type: text/plain
<-- Content-Length: 37
```

(continues on next page)

(continued from previous page)

```
<-- X-CDMI-Meta-Colour: Yellow
<-- X-Object-Meta-Shape: Square
<--
<-- This is the Value of this Data Object
```

Add a new row at end of table Table 16 in 6.4:

Header	Type	Description	Requirement
`x-\*-meta-\*`	Header string	If the "`cdmi_header_metadata`" capability is present, for each request header matching the pattern "`x-*-meta-*`", a new user metadata item shall be created, or an existing metadata item shall be updated, with the metadata name set to the header field-name, and the metadata value set to the header field- value. If a metadata item already exists where the metadata name and the header field-name differ only in case, the existing metadata item value shall be updated. If an empty header field-value is specified, the corresponding metadata item shall be removed from the object. If the number of headers, the length of any of the headers, or the total size of the headers exceeds the limits specified in RFC 2616, or specified by the `cdmi_header_metadata_maxitems`, `cdmi_header_metadata_maxsize`, or the `cdmi_header_metadata_maxtotalsize` capabilities, a `400 Bad Request` shall be returned to the client.	Conditional

Add new example at end of 6.4:

EXAMPLE 3: PUT to the data object URI to update the value and metadata of the data object:

```
--> PUT /cdmi/2.0.0/MyContainer/MyDataObject.txt HTTP/1.1
--> Host: cloud.example.com
--> Content-Type: text/plain;charset=utf-8
--> X-CDMI-Meta-Colour: Green
--> Content-Length: 41
-->
--> This is the new Value of this Data Object

<-- HTTP/1.1 204 No Content
```

Add a new table to Request Headers in 7.2:

Table 163: Request headers - Create a container object using HTTP

Header	Type	Description	Requirement
x-*-meta-*	Header string	If the “cdmi_header_metadata” capability is present, for each request header matching the pattern "x-*-meta-*", a new user metadata item shall be created, with the metadata name set to the header field-name, and the metadata value set to the header field-value. If the number of headers, the length of any of the headers, or the total size of the headers exceeds the limits specified in RFC 2616, or specified by the cdmi_header_metadata_maxitems, cdmi_header_metadata_maxsize, or the cdmi_header_metadata_maxtotalsize capabilities, a 400 Bad Request shall be returned to the client.	Conditional

---

Add new example at end of 7.2:

EXAMPLE 2: PUT to the URI the container object name and metadata:

```
--> PUT /cdmi/2.0.0/MyContainer/ HTTP/1.1
--> Host: cloud.example.com
--> X-CDMI-Meta-Colour: Yellow

<-- HTTP/1.1 201 Created
```

After 7.2, add a new sub-clause “Inspect a container object using HTTP”:

26.5.11 Synopsis

To check for the presence of a container object, the following request shall be performed:

- HEAD <root URI>/<ContainerName>/<TheContainerName>/

Where:

- <root URI> is the path to the CDMI cloud.
- <ContainerName> is zero or more intermediate containers that already exist, with one slash (i.e., "/") between each pair of container names.
- <TheContainerName> is the name specified for the container object to be checked.

The container object shall also also be able to be checked at <root URI>/cdmi_objectid/<objectID>.

26.5.12 Capabilities

The following capabilities describe the supported operations that may be performed when reading an existing container object:

- Support for the ability to read the metadata of an existing conatiner object is indicated by the presence of the cdmi_read_metadata capability in the specified container object.

26.5.13 Request headers

Request headers may be provided as per RFC 2616.

26.5.14 Request message body

A request message body shall not be provided.

26.5.15 Response headers

The HTTP response headers for checking for the presence of a CDMI container object using HTTP are shown in Table 164.

Table 164: Response Headers - Inspect a container object using HTTP

Header	Type	Description	Requirement
`Content-Type`	Header string	`"application/cdmi-container"`	Mandatory
`Location`	Header string	The server shall respond with the URI that the reference redirects to if the object is a reference.	Conditional
`x-\*-meta-\*`	Header string	If the `"cdmi_header_metadata"` capability is present, for each user metadata item in the “metadata” field with a metadata name that is a case-insensitive match to the pattern `"x-*-meta-*"`, a corresponding response header shall be returned to the client where the header field-name shall be the metadata item name, and the header field-value shall be the metadata item value. If a header value to be return is not conformant with RFC 2616, the server may omit the field from the response headers.	Conditional

26.5.16 Response message body

No response body shall be provded, as per RFC 2616.

26.5.17 Response status

The HTTP status codes that occur when checking the presence of a container object using HTTP are described in Table 165.

Table 165: HTTP status codes - Inspect a container object using HTTP

HTTP Status	Description
`200 OK`	The queue object content was returned in the response.
`302 Found`	The resource is a reference to another resource.
`400 Bad Request`	The request contains invalid parameters or field names.
`401 Unauthorized`	The authentication credentials are missing or invalid.
`403 Forbidden`	The client lacks the proper authorization to perform this request.
`404 Not Found`	The resource was not found at the specified URI.

26.5.18 Example

EXAMPLE 1: HEAD to the container object URI to check for the presence of a container object with header metadata:

```
--> HEAD /cdmi/2.0.0/MyContainer/ HTTP/1.1
--> Host: cloud.example.com

<-- HTTP/1.1 200 OK
<-- Content-Type: application/cdmi-container
<-- X-CDMI-Meta-Colour: Yellow
```

Replace contents of 7.3 with:

26.5.19 Synopsis

Reading a container object using HTTP is not defined by this version of this international standard. 9.4 describes how to read a container object using CDMI.

A server implementation is free to respond to HTTP GETs for Container Objects in any way that conforms with RFC 2616.

If container object metadata items matching the pattern `"x-*-meta-*"` are present, these metadata items shall be returned as response headers as per 9.4.

Replace contents of 7.4 with:

26.5.20 Synopsis

Updating a container object using HTTP is not defined by this version of this international standard. clause 9.5 describes how to update a container object using CDMI.

A server implementation is free to respond to HTTP PUTs for existing Container Objects in any way that conforms with RFC 2616.

If container object metadata items matching the pattern `"x-*-meta-*"` are present, these metadata items shall be returned as response headers as per 9.5.

Add new rows to end of Table 124 in 12.2.7:

Capability name	Type	Definition
cdmi_header_metadata	JSON string	If present and `"true"`, this capability indicates that the cloud storage system supports header-visible metadata.
`cdmi_header_metadata_maxitems`	JSON string	If present, this capability indicates the maximum number of user-defined header metadata items supported per object. If absent, there is no additional limit placed on the number of user-defined metadata items.
`cdmi_header_metadata_maxsize`	JSON string	If present, this capability indicates the maximum size, in bytes, of each user-defined header metadata item. If absent, there is no additional limit placed on the size of user-defined metadata items.

Continued on next page

Table 166 – continued from previous page

Capability name	Type	Definition
cdmi_header_metadata_maxtotalsize	JSON string	If present, this capability indicates the maximum size, in bytes, of all user-defined header metadata per object. If absent, there is no additional limit placed on the size of user-defined metadata.

Add to end of 16.5:

If metadata items with a name is a case-insensitive match to the pattern "x-*-meta-*" are created or updated through a CDMI request, the following conditions shall be true, or else a 400 Bad Request result code shall be returned to the client:

- The metadata name shall be a valid HTTP header field-name
- The metadata value that is a valid HTTP header field-value
- The number of matching headers shall not exceed the limits specified by RFC 2616, and shall not exceed the number specified in the cdmi_header_metadata_maxitems capability.
- The size of each matching header shall not exceed the limits specified by RFC 2616, and shall not exceed the number specified in the cdmi_header_metadata_maxsize capability.
- The total size of all of the matching headers shall not exceed the limits specified by RFC 2616, and shall not exceed the number specified in the cdmi_header_metadata_maxtotalsize capability.

26.6 Immediate query

26.6.1 Overview

CDMI provides a query mechanism based around the concept of persistance. A query queue is created, metadata is specified that defines the query operation, the query is performed asynchronously, and results are populated in the queue and then read by the client as separate operations.

This architecture, while providing significant value, is complex for clients that do not need to persist the results of a query. Specifically, a client must: a) asynchronously poll the query queue to determine when results are present and when the query has completed, and b) delete the queue when results are no longer needed.

To provide a simpler interface for simple queries where a small number of results are expected and persistence is not required, the TWG has proposed the following approach to allow query queues to optionally not be persistent, with the results being returned immediately as the response to the initial query queue creation.

In addition, functionality that permits results to be returned immediately has been added to creating asynchronous query queues.

26.6.2 Changes to specification

Modify existing `cdmi_query` entry in Table 124 in 12.2.7:

Capability name	Type	Definition
`cdmi_query`	JSON string	If present and "`true`", the CDMI server supports persistent query queues.

Add a new row at end of table Table 124 in 12.2.7:

Capability name	Type	Definition
`cdmi_query_immediate`	JSON string	If present and "`true`", the CDMI server supports immediate query queues.

Replace the first paragraph of Overview in clause 22 with:

A cloud storage system may optionally implement metadata and/or full-text query functionality. The implementation of query is indicated by the presence of the cloud storage system-wide capabilities for query and requires support for CDMI queues when persisting query results.

Replace the third paragraph of Overview in clause 22 with:

When a client wishes to perform queries, it shall first determine if the system is capable of providing query functionality by checking to see if the `cdmi_query` or `cdmi_query_immediate` capabilities are present in the root container capabilities. If these capabilities are not present and queues are supported, creating a query queue shall be successful, but no query results shall be enqueued into the query queue.

Modify existing cdmi_queue_type entry in Table 152 in 22:

Table 167: Required metadata for a query queue

Metadata name	Type	Description	Requirement
cdmi_queue_type	JSON string	Queue type indicates how the cloud storage system shall manage the queue object. The defined values are: • "cdmi_query_queue" – Perform an asynchronous query, which may return none, some, or all results in the request response body. A new queue object shall be created. • "cdmi_query_immediate" – Perform a synchronous query, returning all matching results in the request response body. The query queue object may not be accessible and shall be automatically deleted when the query completes.	Mandatory

Add new clause “Immediate Queries” to end of 22:

If "cdmi_query_immediate" is specified in cdmi_queue_type, all query results shall be immediately returned in the response body as shown in the following example.

EXAMPLE 3: Perform an Immediate Query:

```
--> PUT /cdmi/2.0.0/MyContainer/myQuery HTTP/1.1
--> Host: cloud.example.com
--> Accept: application/cdmi-queue
--> Content-Type: application/cdmi-queue
-->
--> {
-->   "metadata" : {
-->     "cdmi_queue_type" : "cdmi_query_immediate",
-->     "cdmi_scope_specification" : [
-->       {
-->         "domainURI" : "== /cdmi_domains/MyDomain/",
-->         "parentURI" : "starts /sandbox",
-->         "metadata" : {
-->           "cdmi_size" : "#> 100000"
-->         }
-->       }
-->     ],
-->     "cdmi_results_specification" : {
-->       "objectID" : "",
-->       "metadata" : {
-->         "cdmi_size" : ""
-->       }
-->     }
-->   }
--> }

<-- HTTP/1.1 201 Created
<-- Content-Type: application/cdmi-queue
<-- Location: https://cloud.example.com/cdmi/2.0.0/MyContainer/myQuery
<--
```

(continues on next page)

(continued from previous page)

```
<-- {
<--   "objectType" : "application/cdmi-queue",
<--   "objectID" : "00007E7F00104BE66AB53A9572F9F51E",
<--   "objectName" : "myQuery",
<--   "parentURI " : "/MyContainer/",
<--   "parentID" : "0000706D0010B84FAD185C425D8B537E",
<--   "domainURI" : "/cdmi_domains/MyDomain/",
<--   "capabilitiesURI" : "/cdmi_capabilities/queue/",
<--   "completionStatus" : "Complete",
<--   "metadata" : {
<--     "cdmi_queue_type" : "cdmi_query_immediate",
<--     "cdmi_scope_specification" : [
<--       {
<--         "domainURI" : "== /cdmi_domains/MyDomain/",
<--         "parentURI" : "starts /sandbox",
<--         "metadata" : {
<--           "cdmi_size" : "#> 100000"
<--         }
<--       }
<--     ],
<--     "cdmi_results_specification" : {
<--       "objectID" : "",
<--       "metadata" : {
<--         "cdmi_size" : ""
<--       }
<--     }
<--   },
<--   "queueValues" : "0-0",
<--   "mimetype": [ "application/json" ],
<--   "valuerange": [ "0-111" ],
<--   "valuetransferencoding": [ "base64" ],
<--   "value": "ew0KCQkJIm9iamVjdElEIiA6ICIwMDAwN0U3RjAwMTBFQjkwOTJ
<--     CMjlGNkNENkFENjgyNCIsDQoJCQkibWV0YWRhdGEiIDogew0KCQ
<--     kJCSJjZG1pX3NpemUiIDogIjEwODI2MyINCgkJCX0NCgkJfQ0K"
<-- }
```

Where the value of the above base64 encoded value is:

EXAMPLE 4: An example of the metadata associated with a query queue is as follows:

```
{
    "objectID" : "00007E7F0010EB9092B29F6CD6AD6824",
    "metadata" : {
        "cdmi_size" : "108263"
    }
}
```

Part VI

References

Bibliography

[1] Carl Beame, Robert Thurlow, Brent Callaghan, David Robinson, David Noveck, Mike Eisler, and Spencer Shepler. Network File System (NFS) version 4 Protocol. RFC 3530, April 2003. URL: https://rfc-editor.org/rfc/rfc3530.txt, doi:10.17487/RFC3530.

[2] Tim Berners-Lee, Roy T. Fielding, and Larry M Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986, January 2005. URL: https://rfc-editor.org/rfc/rfc3986.txt, doi:10.17487/RFC3986.

[3] Scott O. Bradner. Key words for use in RFCs to Indicate Requirement Levels. RFC 2119, March 1997. URL: https://rfc-editor.org/rfc/rfc2119.txt, doi:10.17487/RFC2119.

[4] Mallikarjun Chadalapaka, Julian Satran, Kalman Meth, and David L. Black. Internet Small Computer System Interface (iSCSI) Protocol (Consolidated). RFC 7143, April 2014. URL: https://rfc-editor.org/rfc/rfc7143.txt, doi:10.17487/RFC7143.

[5] Douglas Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627, July 2006. URL: https://rfc-editor.org/rfc/rfc4627.txt, doi:10.17487/RFC4627.

[6] Lisa M. Dusseault. HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV). RFC 4918, June 2007. URL: https://rfc-editor.org/rfc/rfc4918.txt, doi:10.17487/RFC4918.

[7] Roy Thomas Fielding. *REST: Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000. URL: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[8] Professor John Franks, Phillip Hallam-Baker, Lawrence C. Stewart, Jeffery L. Hostetler, Scott Lawrence, Paul J. Leach, and Ari Luotonen. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617, June 1999. URL: https://rfc-editor.org/rfc/rfc2617.txt, doi:10.17487/RFC2617.

[9] Ned Freed and Dr. Nathaniel S. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. RFC 2045, November 1996. URL: https://rfc-editor.org/rfc/rfc2045.txt, doi:10.17487/RFC2045.

[10] Ned Freed and Dr. Nathaniel S. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. RFC 2046, November 1996. URL: https://rfc-editor.org/rfc/rfc2046.txt, doi:10.17487/RFC2046.

[11] Tony Hansen and Alexey Melnikov. Additional Media Type Structured Syntax Suffixes. RFC 6839, January 2013. URL: https://rfc-editor.org/rfc/rfc6839.txt, doi:10.17487/RFC6839.

[12] Russ Housley. Cryptographic Message Syntax (CMS). RFC 5652, September 2009. URL: https://rfc-editor.org/rfc/rfc5652.txt, doi:10.17487/RFC5652.

[13] Russ Housley, Tim Polk, Dr. Warwick S. Ford, and David Solo. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 3280, May 2002. URL: https://rfc-editor.org/rfc/rfc3280.txt, doi:10.17487/RFC3280.

[14] Karthik Jaganathan, Larry Zhu, and John Brezak. SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows. RFC 4559, June 2006. URL: https://rfc-editor.org/rfc/rfc4559.txt, doi:10.17487/RFC4559.

[15] Michael Jones. JSON Web Algorithms (JWA). RFC 7518, May 2015. URL: https://rfc-editor.org/rfc/rfc7518.txt, doi:10.17487/RFC7518.

[16] Michael Jones. JSON Web Key (JWK). RFC 7517, May 2015. URL: https://rfc-editor.org/rfc/rfc7517.txt, doi:10.17487/RFC7517.

[17] Michael Jones, John Bradley, and Nat Sakimura. JSON Web Signature (JWS). RFC 7515, May 2015. URL: https://rfc-editor.org/rfc/rfc7515.txt, doi:10.17487/RFC7515.

[18] Michael Jones and Joe Hildebrand. JSON Web Encryption (JWE). RFC 7516, May 2015. URL: https://rfc-editor.org/rfc/rfc7516.txt, doi:10.17487/RFC7516.

[19] Simon Josefsson. The Base16, Base32, and Base64 Data Encodings. RFC 4648, October 2006. URL: https://rfc-editor.org/rfc/rfc4648.txt, doi:10.17487/RFC4648.

[20] Larry M Masinter and Ernesto Nebel. Form-based File Upload in HTML. RFC 1867, November 1995. URL: https://rfc-editor.org/rfc/rfc1867.txt, doi:10.17487/RFC1867.

[21] Keith McCloghrie, Jürgen Schönwälder, David T. Perkins, and Keith McCloghrie. Structure of Management Information Version 2 (SMIv2). RFC 2578, April 1999. URL: https://rfc-editor.org/rfc/rfc2578.txt, doi:10.17487/RFC2578.

[22] Keith Moore. MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text. RFC 2047, November 1996. URL: https://rfc-editor.org/rfc/rfc2047.txt, doi:10.17487/RFC2047.

[23] Henrik Frystyk Nielsen, Jeffrey Mogul, Larry M Masinter, Roy T. Fielding, Jim Gettys, Paul J. Leach, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, June 1999. URL: https://rfc-editor.org/rfc/rfc2616.txt, doi:10.17487/RFC2616.

[24] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018. URL: https://rfc-editor.org/rfc/rfc8446.txt, doi:10.17487/RFC8446.

[25] Eric Rescorla and Tim Dierks. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, August 2008. URL: https://rfc-editor.org/rfc/rfc5246.txt, doi:10.17487/RFC5246.

[26] Krishna Sankar and Arnold Jones. Cloud Data Management Interface (CDMI) Media Types. RFC 6208, April 2011. URL: https://rfc-editor.org/rfc/rfc6208.txt, doi:10.17487/RFC6208.

[27] Jamie Zawinski, Larry M Masinter, and Martin J. Dürst. The 'mailto' URI Scheme. RFC 6068, October 2010. URL: https://rfc-editor.org/rfc/rfc6068.txt, doi:10.17487/RFC6068.

[28] ISO/IEC Joint Directives Maintenance Team. ISO/IEC directives, part 2 – principles and rules for the structure and drafting of ISO and IEC documents. ISO/IEC Directives, Part 2, 2018, 2018. URL: https://www.iso.org/directives-and-policies.html.

[29] ISO/IEC JTC 1/SC 25 Interconnection of information technology equipment. Information technology – small computer system interface (SCSI) – part 414: SCSI architecture model-4 (sam-4). ISO/IEC 14776-414:2009, June 2009. URL: https://www.iso.org/standard/53961.html.

[30] ISO/IEC JTC 1/SC 27 Information security, cybersecurity and privacy protection. Information technology – security techniques – storage security. ISO/IEC 27040:2015, January 2015. URL: https://www.iso.org/standard/44404.html.

[31] ISO/IEC JTC 1/SC 38 Cloud Computing and Distributed Platforms. Information technology – cloud computing – overview and vocabulary. ISO/IEC 17788:2014, October 2014. URL: https://www.iso.org/standard/60544.html.

[32] ISO/TC 154 Processes, data elements and documents in commerce, industry and administration. Date and time – representations for information interchange – part 1: basic rules. ISO 8601-1:2019, February 2019. URL: https://www.iso.org/standard/70907.html.

[33] ISO/TC 154 Processes, data elements and documents in commerce, industry and administration. Date and time – representations for information interchange – part 2: extensions. ISO 8601-2:2019, February 2019. URL: https://www.iso.org/standard/70907.html.

[34] ISO/TC 20/SC 13 Space data and information transfer systems. Space data and information transfer systems – open archival information system (OAIS) – reference model. ISO 14721:2012, August 2012. URL: https://www.iso.org/standard/57284.html.

[35] ISO/TC 46 Information and documentation. Codes for the representation of names of countries and their subdivisions – part 1: country codes. ISO 3166-1:2013, November 2013. URL: https://www.iso.org/standard/63545.html.

[36] ISO/TC 46 Information and documentation. Codes for the representation of names of countries and their subdivisions – part 2: country subdivision code. ISO 3166-2:2013, November 2013. URL: https://www.iso.org/standard/63546.html.

[37] ISO/TC 46 Information and documentation. Codes for the representation of names of countries and their subdivisions – part 3: code for formerly used names of countries. ISO 3166-3:2013, November 2013. URL: https://www.iso.org/standard/63547.html.

[38] ISO/TC 68/SC 8 Reference data for financial services. Codes for the representation of currencies. ISO 4217:2015, August 2015. URL: https://www.iso.org/standard/64758.html.

[39] ISO/TC JTC 1/SC 27 Information security, cybersecurity and privacy protection. TLS specification for storage systems. ISO 20648:2016, August 2016. URL: https://www.iso.org/obp/ui/#iso:std:iso-iec:20648:ed-1:v1:en.

[40] Open Grid Forum. Open cloud computing interface v1.1. June 2011. URL: http://occi-wg.org/about/specification/.

[41] POSIX - Austin Joint Working Group. IEEE standard for information technology–portable operating system interface (POSIX(r)) base specifications, issue 7. IEEE 1003.1-2017, December 2017. URL: https://standards.ieee.org/standard/1003_1-2017.html.

[42] Storage Networking Industry Association. TLS specification for storage systems v1.1.0. November 2020. URL: https://www.snia.org/tech_activities/standards/curr_standards/tls.

[43] Information technology – open systems interconnection – the directory – part 8: public-key and attribute certificate frameworks. ISO/IEC 9594-8:2017, May 2017. URL: https://www.iso.org/standard/72557.html.