SNIA

Introduction to the SNIA Cloud Data Management Interface (CDMI™) 3.0

SNIA Cloud Storage Technical Work Group

2025-08-20

Table of Contents

List of Figures

Introduction to CDMI 3.0

The SNIA Cloud Data Management Interface (CDMI), also known as ISO/IEC 17826, provides a vendor-neutral standard interface for multi-protocol discovery, configuration, management and portable data movement. Implementers should consider using CDMI when applications require protocol-independent data management.

Common use cases include:

1. Clients need to discover which namespaces exist
2. Clients need to discover which data access protocols can be used to access a namespace
3. Clients need to discover which data access protocols are supported for a namespace
4. Clients need to specify which data access protocols are active for a namespace
5. Clients need to specify how data should be managed
6. Clients need to move data between systems while preserving cross-protocol structure, contents and metadata

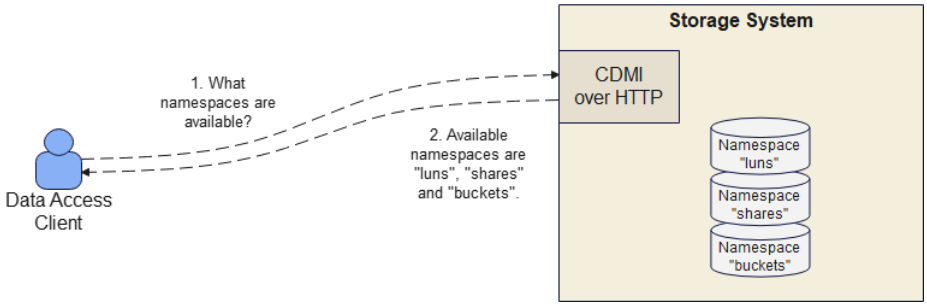
CDMI allows clients to use a lightweight and consistent management interface independent of data access protocols. By enabling management operations outside each specific data access protocol, CDMI works across multiple data access protocols and multiple deployment models, including on-premises, hybrid, and cloud deployments.

CDMI 3.0 is the next major revision of the standard. The SNIA Cloud Storage Technical Work Group is adding support for Model Context Protocol (MCP) as a peer protocol alongside HTTP, support for new AI-driven use cases, and support for additional storage protocols and data types, including object, table, graph and streaming data. This whitepaper highlights additions to core use case under the heading, and concludes with a summary of improvements and refinements originating from implementer feedback received over the last three years.

Discovering Namespaces

CDMI allows data management clients to discover namespaces that can be accessed by data access protocols.

Discovering namespaces



Namespaces are the way clients *identify* data that can be stored and retrieved. Namespaces can contain file names, volume LUNs, bucket objects, and table rows. Protocols provide a mechanism to *access* data within a given namespace. NFS can be used to access files, iSCSI can be used to access LUNs, S3 can be used to access objects, and SQL can be used to access tables rows.

How to Discover Namespaces using CDMI over HTTP

To discover namespaces, CDMI clients perform standard HTTP GET operations to request the CDMI representation of a resource specified by a URI:

```
GET / HTTP/1.1
Accept: application/cdmi-container
```

The response to this GET request is a JSON document specified in the CDMI standard (a CDMI container), which includes a list of "child" namespaces associated with the URI:

```
{
    "children": [
        "luns/",
        "shares/",
        "buckets/"
    ]
}
```

In the above example, three groupings of namespaces are indicated, LUNs, shares, and buckets. Namespaces can be listed recursively by performing additional requests to list children of each discovered namespace.

What's Coming in CDMI 3.0

In CDMI 2.0, clients must know the "root URI" to begin discovering namespaces ("/" in the above example). CDMI 3.0 adds support for RFC 8615 Well-Known Uniform Resource Identifiers to enable standardized discovery of the root URI for a given server.

How to Discover Namespaces using CDMI over MCP

At the lowest level, CDMI is a set of JSON-formatted media types defined in RFC 6208, coupled with a specification that defines how these documents are generated and interpreted by a CDMI server. The CDMI 2.0 specification defines how a CDMI server responds to CDMI JSON documents when received over HTTP.

However, CDMI is not limited to HTTP. For example, CDMI can be used with MCP as a transport. To discover namespaces using MCP, an MCP client first discovers what CDMI services a CDMI MCP server supports, then constructs a CDMI JSON document to be sent to the MCP server:

```
{
    "jsonrpc": "2.0",
    "id": 4,
    "method": "tools/call",
    "params": {
        "function": "get",
        "representation": "cdmi_container",
        "uri": "/?children",
        "arguments": {
        }
    }
}
```

The response to this request is an MCP response that includes the listing of namespaces as specified in the CDMI standard:

```
{
    "jsonrpc": "2.0",
    "id": 4,
    "result": {
        "function": "get",
        "representation": "cdmi_container",
        "uri": "/?children",
        "results": {
            "children": [
                "luns/",
                "shares/",
                "buckets/"
            ]
        }
    }
}
```

What's Coming in CDMI 3.0

In CDMI 2.0, HTTP is the only transport specified for CDMI JSON requests and responses. CDMI 3.0 will explore how these existing JSON documents can be used with the MCP to enable standardized agent-driven storage discovery and management.

How to Recursively Discover Namespaces

If the CDMI 2.0 server implements the "Extended Child Listing" CDMI extension, all namespaces can be discovered in one operation:

```
GET /?children=! HTTP/1.1
Accept: application/cdmi-container
```

The returned JSON includes the full namespace tree:

```
{
    "children": [
        "luns/", [
            "lun_1",
            "lun_2" ],
        "shares/", [
            "docs/",
            "home/", ],
        "buckets/", [
            "public/" ]
    ]
}
```

In the above example, we see the three groupings of namespaces, along with all of the recursively listed namespaces within each of these namespaces

What's Coming in CDMI 3.0

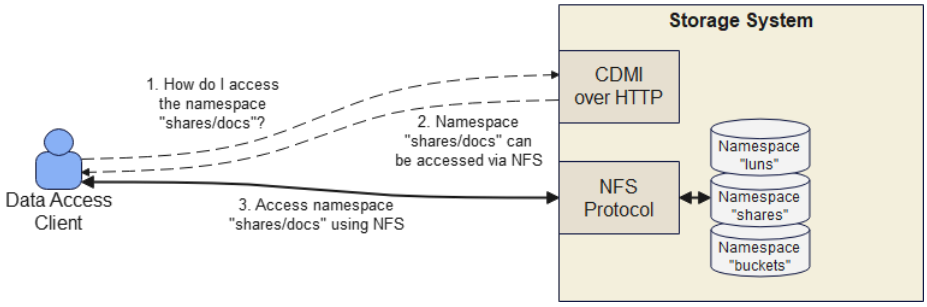
In CDMI 2.0, the introduction to the standard lacked a clear description of how CDMI relates to data namespaces. CDMI 3.0 includes a completely re-written introduction that clearly explains how CDMI "containers" relate to namespaces, and how CDMI can add management value to every data storage product.

Additional feedback on the extended child listing extension is also being requested, and if two interoperable implementations can be tested, the extension can be merged into CDMI 3.0.

Discovering How to Access a Namespace

CDMI 3.0 adds the discovery of which data access protocols are active for a given namespace. This is accomplished by working alongside data access protocols such as NFS, iSCSI and S3 to provide sufficient information for these protocols to be used to access namespaces.

Discovering how to access a namespace



How to Discover Active Data Access Protocols

To discover how to access a specific namespace, CDMI clients perform an HTTP GET for a discovered namespace:

```
GET /shares/docs/?exports-provided HTTP/1.1
Accept: application/cdmi-container
```

The response to this GET request is a JSON document containing the different protocols that can be used to access the namespace associated with the URI:

```
{
    "exports-provided": {
        "nfsv4": {
            "client_uri": "nfs://example/docs/",
            "client_protocol": "NFSv4.1",
            "client_protocol_transport": "TCP/IP",
            "client_flags": [ "hard", "port=2049", "sec=krb5p" ],
            "export_definition_uri": "/shares/docs/"
        },
        "smb": {
            "client_uri": "smb://example/docs/",
            "client_protocol": "SMB 3.1",
            "client_protocol_transport": "TCP/IP",
            "client_flags": [ "vers=3.1.1", "multiuser", "sec=krb5", "mfsymlinks" ],
            "export_definition_uri": "/shares/docs/"
        },
        "s3": {
            "client_uri": "https://docs.example.com/",
            "client_protocol": "S3",
            "client_protocol_transport": "TCP/IP",
            "client_flags": [ ],
            "export_definition_uri": "/shares/docs/"
```

```
            }
        }
}
```

This returns a list of active data access protocols for that namespace. The returned list also includes configuration information required to use each data access protocol, such as network addresses, protocol-specific options, and security information.

In the above example, the namespace can be accessed using multiple protocols, including NFS, SMB, and S3, and the mount parameters required to configure the client are provided for each protocol.

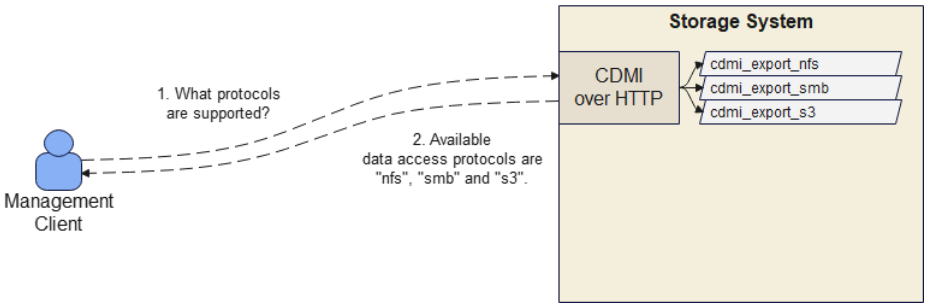
What's Coming in CDMI 3.0

In CDMI 2.0, determining the exports provided required looking at the specified exports for a namespace and all parent namespaces. CDMI 3.0 adds the above descripted "exports_provided" to simplify export discovery, and to provide a method for storage servers to indicate what additional configuration information is required by clients to successfully access the namespace.

Discovering Supported Data Access Protocols

The CDMI standard allows a management client to discover which data access protocols are supported for a given namespace.

Discovering Supported Data Access Protocols



How to Discover Supported Data Access Protocols

To discover which data access protocols are supported for a given namespace, CDMI's "capabilities" facility is accessed to return the list of supported data access protocols. CDMI clients first perform an HTTP GET for a discovered namespace to get the corresponding "capabilities URI":

```
GET /shares/docs/?capabilitiesURI HTTP/1.1
Accept: application/cdmi-container
```

The response to this GET request is a JSON document containing the URI that can be used to look up the capabilities associated with the namespace:

```
{
    "capabilitiesURI": "/.well-known/cdmi/capabilities/share-type-232/"
}
```

CDMI clients can now use this capability to look up which data access protocols are supported by performing an HTTP GET for returned capabilities URI:

```
GET /.well-known/cdmi/capabilities/share-type-232/?capabilities=cdmi_export HTTP/1.1
Accept: application/cdmi-capability
```

The response to this GET request is a JSON document containing the capabilities associated with the namespace:

```
{
    "capabilities": {
        "cdmi_export_nfs": "true",
        "cdmi_export_nfs_versions": ["3.0", "4.0", "4.1", "4.2"],
        "cdmi_export_smb": "true",
        "cdmi_export_smb_versions": ["3.0", "3.1"],
        "cdmi_export_s3": "true",
        "cdmi_export_s3_profiles": ["snia_base"],
}
```

This returned list indicates which data access protocols can be specified for a given namespace.

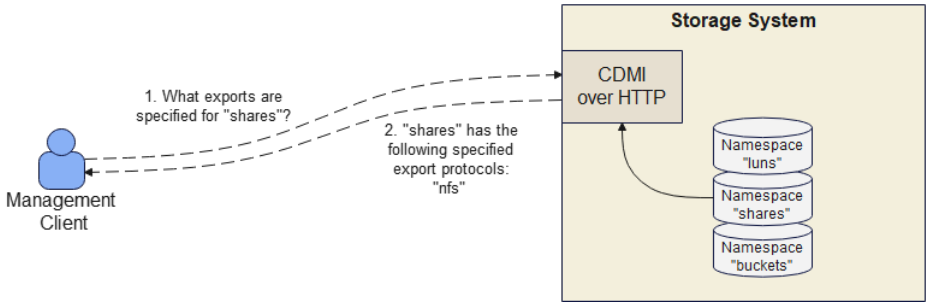
What's Coming in CDMI 3.0

In CDMI 2.0, support for specific data export protocols is defined as part of the standard, such as "cdmi_export_nfs" and "cdmi_export_smb". CDMI 3.0 adds companion capabilities describing which protocol versions are supported. CDMI 3.0 also expands the number of supported data export protocols, such as "s3", "http", and "mcp", adds the ability to discover supported interoperability profiles, and adds the ability to include management protocols such as SNIA Swordfish™, in addition to data access protocols.

Discovering Specified Namespace Data Access Protocols

The CDMI standard enables the discovery of which data access protocols have been specified for a namespace. These protocols are known as "exports".

Discover how a namespace can be accessed



Exports are declarative instructions to a storage server that a given namespace should be accessible using the specified access protocol. For example, an NFS export indicates that the storage system should attempt to make that namespace available using the NFS protocol.

Discovering Specified Exports

To discover how a specific namespace should be accessible by clients, CDMI clients perform an HTTP GET for a given namespace's exports:

```
GET /shares/docs/?exports HTTP/1.1
Accept: application/cdmi-container
```

This returns a list of declarative exports, each of which specifies how the server should allow access to the namespace using the specified data access protocol:

```
{
    "exports": {
        "nfsv4": {
            "type": "NFS",
            "protocol": "NFSv4.1",
            "path": "example:/docs"
        },
        "smb": {
            "type": "SMB",
            "sharename": "docs",
            "comment": "Documents"
        }
    }
}
```

In the above examples, three exports are requested for the "shares/docs" namespace: NFS and SMB. Not all storage servers can support all possible export types, so the "exports_provided" described earlier in this whitepaper provides a mechanism by which to determine which of the specified exports are active.

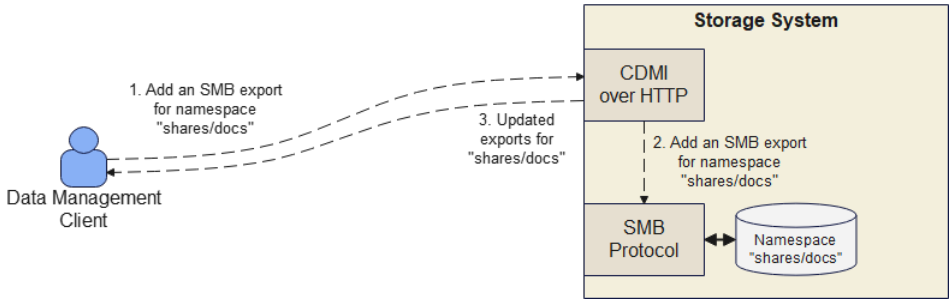
What's Coming in CDMI 3.0

CDMI 3.0 expands the number of supported data export protocols, such as "s3", "http", and "mcp" and adds the ability to discover management protocols such as SNIA Swordfish™.

Specifying Namespace Data Access Protocols

The CDMI standard enables the specification of which data access protocols can be used to access data namespaces.

Specify how a namespace can be accessed



Exports are declarative instructions to a storage server that a given namespace should be accessible using the specified access protocol. For example, an NFS export indicates that the storage system should attempt to make that namespace available using the NFS protocol.

Adding Exports

Clients with write permissions can add, modify, and remove exports associated with a namespace by replacing the exports associated with a namespace. This is accomplished by performing an HTTP PATCH operation. For example, to add an S3 export, the following PATCH request would be performed:

```
PATCH /shares/docs/?exports HTTP/1.1
Accept: application/cdmi-container
Content-Type: application/cdmi-container

{
    "exports": {
        "nfsv4": {
            "type": "NFS",
            "protocol": "NFSv4.1",
            "path": "example:/docs"
        },
        "smb": {
```


```
            "type": "SMB",
            "sharename": "docs",
            "comment": "Documents"
        },
        "s3": {
            "type": "S3",
            "bucket_name": "docs",
            "virtual_server": "docs.example.com
        }
    }
}
```

Here, the "s3" export is added, and the previously existing NFS and SMB exports are preserved.

What's Coming in CDMI 3.0

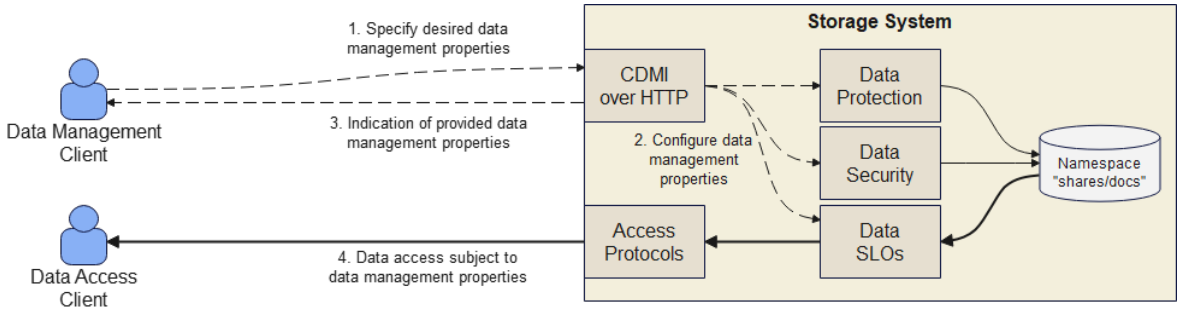
CDMI 3.0 adds support for new types of data protocol exports, including NFS-over-RDMA, S3, S3-over-RDMA, HTTP and MCP, plus new table, graph and stream protocols. These newly added protocols are critical for emerging AI workflows, allowing CDMI to help connect AI clients and AI data servers together.

Storage developers interested in enhancing their products with multi-protocol access to data are welcomed to join the SNIA Cloud Storage Technical Work Group to help specify how data access and query protocols can be managed using CDMI. Contact cloudtwgchair@snia.org to join.

Specify how Data is Managed

CDMI provides a data-independent mechanism to specify desired management behaviors, such as desired client service level objectives (SLOs), geographic access restrictions, retention and hold, data protection, and more. This is accomplished by adding declarative metadata, known as "Data System Metadata" to namespaces, where each data system metadata item has a well-defined meaning shared by the declaring client and the interpreting server.

Specifying data management properties



Examples of data system metadata defined in CDMI includes:

- The degree of redundancy required for stored data, including the required immediate redundancy before a positive indication that data is stored is returned to the data access protocol client,
- The degree of data geographic dispersion required for stored data,
- Restrictions on the placement of stored data within country and region boundaries,
- Retention and hold restrictions on the deletion or data,
- Requirements for encryption, hash-verification of data, and sanitization method for data,
- Requirements on maximum latency and minimum throughput for data retrieval, and
- Requirements for RTO and RPO.

Each data system metadata field specifying a desired management behavior has a companion "data system metadata provided" field that indicates the actual level of service provided for the namespace. For example, the maximum latency for a namespace is indicated by the value of the "cdmi_latency_provided" metadata, which can be used to determine if data is stored on a high-latency storage system such as a tape library.

Specifying Desired Data Management Properties

For example, if a data management client desires to specify that the contents of a namespace shall only be stored within the EU, the following "Data System Metadata" can be added:

```
PATCH /shares/docs/?metadata/cdmi_geographic_placement HTTP/1.1
Accept: application/cdmi-container
Content-Type: application/cdmi-container

{
    "metadata": {
        "cdmi_geographic_placement": [ "EU" ]
    }
}
```

Here, the "cdmi_geographic_placement" data system metadata item is added, with a value that specifies that only placement within the EU is permitted.

Obtaining Provided Data Management Properties

Specifying desired data management properties is declarative. It provides the storage system with an indication of what the data management client desires for the namespace. However, what is desired may not be achievable, for example when the storage system is not capable of achieving what is requested, the level of payment is not sufficient to provide that level of service, or when the storage system is not configured to provide that level of service.

CDMI allows data management clients to obtain the actual level of service being provided through the "_provided" read-only data system metadata items. To obtain the current latency for stored data

in a namespace, the data management client would perform a GET operation to read this metadata item:

```
GET /shares/docs/?metadata/cdmi_latency_provided HTTP/1.1
Accept: application/cdmi-container
```

This returns the value of the requested data system metadata item:

```
{
    "metadata": {
        "cdmi_latency_provided": "0.01"
    }
}
```

Here, the "cdmi_latency_provided" data system metadata value indicates that the storage system guarantees that the data will be provided within 10 ms.

What's Coming in CDMI 3.0

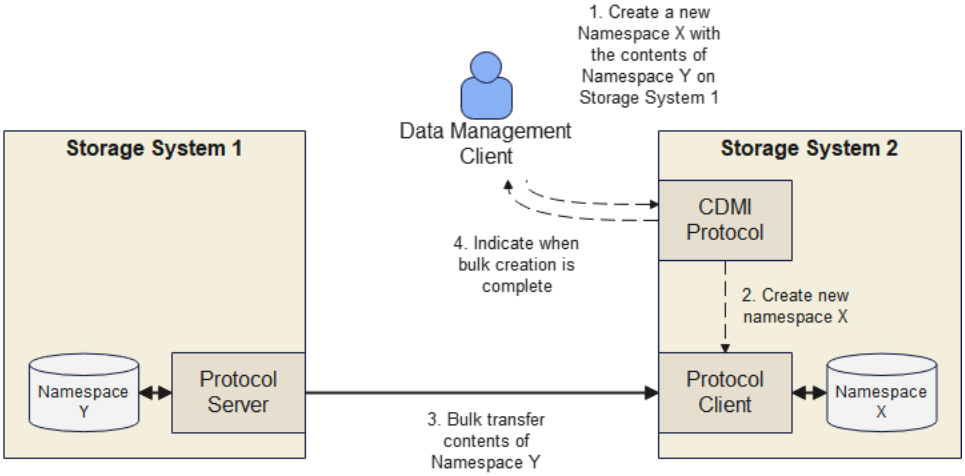
CDMI 3.0 adds new data system metadata for specifying the preferred representation and for discovering available representations.

Move Data Between Systems

CDMI provides a system-independent mechanism to request the bulk movement of data from one storage system to another. This is accomplished by using server-side copy, where the destination can be directed to create a new namespace based on the contents of a second system that contains the data used to populate the new namespace.

CDMI also provides a serialization format that can be used to create a new file containing the contents of a namespace. This serialization format is JSON-based and contains a superset of all metadata across all access protocols to enable lossless transport of data.

Performing a Server-Side Data Copy



CDMI also supports the ability to have long-running background operations that indicate their progress and error state.

Third-Party Copy

To move the contents of one namespace from one system to a second system, the CDMI data management client creates a new namespace on the destination system, while specifying a source location on the source system:

```
PUT /shares/new-namespace-x HTTP/1.1
Accept: application/cdmi-container
Content-Type: application/cdmi-container

{
    "copy": "nfs://server/namespace/namespace-y/"
}
```

Any URI scheme can be supported, as long as the CDMI server supports reading data from that location. Server-side copy also requires the server to have authorization credentials for the source, either delegated by the data management system, or provided by a key management server.

Serialization and Deserialization

Any CDMI namespace can be serialized into a file by specifying the serialization source:

```
PUT /shares/docs/serialized-namespace.json HTTP/1.1
Accept: application/cdmi-object
Content-Type: application/cdmi-object

{
    "serialize": "/shares/new-namespace-x"
}
```

This creates a new file named "serialized-namespace.json" that contains the contents of the "new-namespace-x".

Deserializing a serialized representation reverses the operation:

```
PUT /shares/docs/third-namespace HTTP/1.1
Accept: application/cdmi-object
Content-Type: application/cdmi-object

{
    "deserialize": "serialized-namespace.json"
}
```

This creates a new namespace "third-namespace" that deserializes the contents of the "serialized-namespace.json" file.

What's Coming in CDMI 3.0

CDMI 3.0 adds new capabilities to discover which URI schemes are supported for server-side copy, move, and deserialization.

Additional CDMI 3.0 Refinements

Based on feedback from implementers, the following additional refinements are proposed for CDMI 3.0:

Guidance for S3 Protocol Exports

CDMI 3.0 includes guidance for S3 protocol bucket exports. Amazon's AWS S3 service has been widely adopted as an object data protocol, which uses the concept of a "bucket" to contain a flat organization of key/value data available via an HTTP-based protocol (the "S3" protocol). Object names (keys) in S3 can use specially designated delimiters, such as "/" to create hierarchical organizations of data within the bucket.

CDMI S3 exports allow management clients to specify to storage systems that namespaces (or containers within a namespace) should provide access via the S3 protocol. For example, a CDMI client can specify that a directory in a namespace exported through NFS should also be exported as an S3 bucket, allowing cross-protocol access to the contents of that bucket/directory.

CDMI defines the configuration required to export a container as an S3 bucket, for example, the bucket name, access mapping, and name resolution. CDMI also defines recommendations for handling file names that are not expressible through the S3 interface, and S3 object names that are not expressible through the file interface. Finally, CDMI defines how S3 bucket and object metadata and attributes are presented through the CDMI representations and serializations.

Support for Graph Data Models

CDMI provides a means to support data and serialization management tasks associated with graph-based information. CDMI's graph capabilities provide a consistent, standardized API for implementers to bridge interchange gaps and enable graph model interoperability.

Today graph data modeling is a complex, evolving technological landscape without a clear single universal standard addressing all aspects of graph data models. One method to enable graph model interoperability between platforms uses serializations formats. Several specifications may be used to enable graph model interchange using serialization. The models also may be directly manipulated to modify models and model content.

Many types of information, including geospatial data, decision trees, knowledge maps, and statistical maps include explicitly-defined relationships between data items. These relationships form a graph, where the data items (the "nodes" or "vertices") are connected using additional data (the "edges"). These edges can be typed to indicate the relationship between the vertices.

The addition of graph data models to CDMI enables management clients to specify graph relationships for stored data and metadata in a standardized way. This allows CDMI to support serialization formats for graph data, and allows CDMI data to be exported using graph query protocols such as GQL. CDMI serialization of graph data provides a JSON array of all matching edges and nodes, using the same approach used for serializing CDMI objects by ID.

Support for Graph Relationships

CDMI represents graph data as relationships between objects. Objects can have zero or more relationships, with each relationship indicating the subject of the relationship, the type (predicate) of the relationship, and the object that the relationship refers to. Any CDMI object item can include relationships, and any CDMI object can be the subject, predicate and/or object of a relationship, as these are all expressible as URLs.

Graph relationships are stored in a "rel" field, as specified by the W3 RDF-JSON recommendation[1]. An example of a CDMI data object with a graph reference indicating that it was published by SNIA is shown below:

```
{
    "objectName": "CDMI Whitepaper",
    "objectID": "urn:uuid:22074f74-8a8b-4281-922c-02f2c5ffe236",
    "rel": {
        ".#value": {
            "http://purl.org/dc/elements/1.1/publisher": [ {
                "value": "https://www.snia.org/",
                "type": "uri"
            ] },
        },
    },
```

[1] https://www.w3.org/TR/rdf-json/

```
    "value": "<data>"
}
```

In this example, the relationship subject ".#value" refers to the value field of the CDMI object, the relationship predicate "http://purl.org/dc/elements/1.1/publisher" refers to the Dublin Core "publisher" term[2], and the object is a URL to SNIA's website "https://www.snia.org/".

Support for Graph Models

In addition to allow graph relationships expressed as metadata associated with any existing CDMI object, CDMI can also provide access to stored graph models using serialization formats. CDMI allows serialization formats to be specified when exporting the graph model data over different data access protocols. When data access protocols support the specification of the desired representation, such as HTTP and MCP, the client can use CDMI to discover which graph serialization formats are supported, and request the desired serialization format using available data access protocols.

How to Discover Graph Model Serialization Representations using CDMI over MCP

To discover available graph model serialization representations using MCP, two steps are performed. First, the MCP client constructs a CDMI JSON document to be sent to the MCP server to discover what representations are available for a graph model:

```
{
    "jsonrpc": "2.0",
    "id": 4,
    "method": "tools/call",
    "params": {
        "function": "get",
        "representation": "cdmi_object",
        "uri": "/mygraphmodel?metadata/cdmi_representations_provided",
        "arguments": {
        }
    }
}
```

The response to this request is an MCP response that includes the list of available representations, specifically, the available graph model serialization formats:

```
{
```

[2] https://www.dublincore.org/specifications/dublin-core/dcmi-terms/#http://purl.org/dc/elements/1.1/publisher

```
    "jsonrpc": "2.0",
    "id": 4,
    "result": {
        "function": "get",
        "representation": "cdmi_object",
        "uri": "/mygraphmodel?metadata/cdmi_representations_provided",
        "results": {
            "metadata": {
                "cdmi_representations_provided/": [
                    "application/graphml+xml",
                    "application/rdf+xml",
                    "application/vnd.xmi+xml",
                    "application/gml+xml"
                ]
            }
        }
    }
}
```

To select and read a specific available graph model serialization representation using MCP, the MCP client constructs a CDMI JSON document to be sent to the MCP server to access the desired representations of the graph model:

```
{
    "jsonrpc": "2.0",
    "id": 4,
    "method": "tools/call",
    "params": {
        "function": "get",
        "representation": "application/rdf+xml",
        "uri": "/mygraphmodel",
        "arguments": {
        }
    }
}
```

The response to this request is an MCP response that includes the list available representations, specifically, the available graph model serialization formats:

```
{
    "jsonrpc": "2.0",
    "id": 4,
    "result": {
        "function": "get",
        "representation": "application/rdf+xml",
        "uri": "/mygraphmodel ",
        "results": {
            "value": "<xml ...>"
            }
        }
    }
}
```

Native Support for Table Data

Many types of information are structured according to a schema that defines which specific sets of data fields (tables) and how different tables are related to each other. This includes traditional relational databases and newer "noSQL" database systems. Tables have two addressable dimensions: "columns," which define the data items in the table, and "rows," specific instances of data items in the table.

The addition of table query export protocols (e.g. SQL) for table-structured data enables CDMI to manage access to tables using different protocols (e.g. file, query, ETL and streaming protocols), as well as manage declarative metadata associated with the table in a standardized way. Additional work is planned to define which export protocols and data representations are desired by implementers.

Native Support for Tensor Data

Multi-dimensional arrays of numeric data (known as tensors) are increasingly being managed by storage systems.

The addition of support for tensor data enables CDMI to manage access to tensors using different protocols (pytorch, numpy, vector query), as well as manage declarative metadata associated with tensors in a standardized way. This allows CDMI to act as a standard interchange format for tensor data.

CDMI represents tensors as arrays of numeric values. CDMI also allows tensors to have searchable metadata, including keywords, which are attached to each tensor. Tensors can also be combined with graphs to allow tensors to have relationships with other data, including other vectors.

An example of a serialized CDMI representation of a tensor with associated metadata and graph links is shown below:

```
{
    "objectType" : "application/tensor+json",
    "metadata": {
        "tags": [ "keyword" ]
    },
    "rel": {
        ".#value": {
            "http://purl.org/dc/elements/1.1/publisher": [ {
                "value": "http://www.snia.org/",
                "type": "uri"
            ] },
        },
    },
    "value": [
        [
            "1",
            "2",
            "3"
        ], [
            "4",
            "5",
```

```
            "6"
        ]
    ]
}
```

As with all CDMI serializations, numbers are stored as strings to avoid silent data corruption of large numbers, and to enable support for non-conventional numeric systems.

CDMI allows tensors to be accessed using different representations. To obtain the list of available representations, the data management client would perform a GET operation to read representations provided data system metadata item:

```
GET /shares/docs/myTensor?metadata/cdmi_representations_provided HTTP/1.1
Accept: application/cdmi-object
```

This returns the value of the requested data system metadata item:

```
{
    "metadata": {
        "cdmi_representations_provided": [
            "application/tensor+json",
            "application/vnd.tf.tfRecord",
            "application/vnd.pyTorch.safeTensor"
        ]
    }
}
```

Here, the tensor can be accessed using three different representations, a JSON tensor representation, as a TensorFlow TFRecord, and as a pyTorch SafeTensor representation.

Generalized Stored Queries and Query Result Data

In the CDMI 2.0 specification, CDMI queries allows a client to specify a query (a specification of an intended scope of resources and which specific data from within those identified resources) as a CDMI-managed object; and allows a client to access query-result data using a CDMI queue object. This approach allows for the efficient identification and access to user-specified subsets of managed data.

In CDMI 3.0, this approach will be broadened to allow any server-provided query language to be incorporated into CDMI as a "query export protocol," which can then be used by non-CDMI clients for performing queries and accessing query-result data. This allows CDMI to manage client access to data using industry standard query languages and protocols (e.g., SQL queries for access to table data and GQL queries for access to graph data, LLM queries to LLM datasets, natural language query commonly used in generative AI, and other de-facto query-response protocols). Similar to other data access protocol exports, each query protocol is defined as an exported protocol and can be specified for objects and namespaces managed through CDMI.

These new capabilities allow management clients to specify which query protocols can be used to access which sets of data. For example, using SQL to access specific sets of table data, using SQL to access the contents of arrow or HDF5 data, or using GQL to access specific sets of graph data, or using an LLM query protocol to access an LLM dataset. This approach is consistent with how CDMI allows configuration of data-access protocols such as NFS, SMB, and iSCSI.

CDMI management clients can specify query protocol exports for CDMI data objects and CDMI containers representing various types of data. Once exports are specified, the CDMI server configures corresponding query providers allowing query protocol access to exported data.

When queries are attached to CDMI objects, CDMI specifies how these queries are passed to the associated query provider and how the query-result data is stored into the associated CDMI queue objects.

General Simplifications of CDMI

Based on feedback from implementers, CDMI 3.0 introduces several simplifications to the CDMI specification to reduce complexity.

These include:

- The CDMI 3.0 specification document has been re-organized to be protocol-based, allowing sections of the standard that are not applicable for an implementer to be easily skipped.
- CDMI 3.0 implementers are no longer required to obtain an IANA enterprise number.
- CDMI 3.0 relaxes constraints for optional object identifiers by allowing the use of any URN namespace to be used, such as UUID and hash-based identifiers.
- CDMI 3.0 exports are now allowed for all CDMI object types.
- CDMI 3.0 aligns with new and updated RFCs to clarify under-specified behaviors, including:
    - using RFC 8615 "well-known" URIs for capabilities,
    - using RFC 6902 JSON patch path specifiers for GET and PATCH URIs,
    - using RFC 7396 JSON patch merge for PATCH body handling, and,
    - using RFC 9457 Problem Details for error reporting.
- CDMI 3.0 relaxes constraints on objects duality, allowing objects to have simultaneous data, container and queue representations.

SNIA
5201 Great America Parkway, Suite 320, Santa Clara, CA, 95054
Phone:719-694-1380 • Fax: 719-694-1385 • www.snia.org