University of California Curation Center

Merritt Storage Service

Rev. 1.00 – 2013-02-04

1 Introduction

Information technology and resources have become integral and indispensable to the pedagogic mission of the University of California. Members of the UC community routinely produce and utilize a wide variety of digital assets in the course of teaching, learning, and research. These assets represent the intellectual capital of the University; they have inherent enduring value and need to be managed carefully to ensure that they will remain available for use by future scholars. Within the UC system the California Digital Library (CDL) UC Curation Center (UC3) has a broad mandate to ensure the long-term usability of the digital assets of the University. UC3 views its mission in terms of *digital curation*, the set of policies and practices aimed at maintaining and adding value to authentic digital assets for use by scholars now and into the indefinite future [Abbott, 2008].

In order to meet these obligations UC3 is developing Merritt, an emergent approach to digital curation infrastructure [Merritt]. Merritt devolves infrastructure function into a growing set of granular, orthogonal, but interoperable micro-services embodying curation values and strategies. Since each of the services is small and self-contained, they are collectively easier to develop, deploy, maintain and enhance [Denning]; equally as important, since the level of investment in and commitment to any given service is small, they are more easily replaced when they have outlived their usefulness. Yet at the same time, complex curation functionality emerges from the strategic combination of individual, atomistic services [Fisher].

One of the foundational Merritt services is the Ingest service, which provides a robust, flexible, and easily deployed environment in which to manage the secure and persistent storage of encoded files that represent digital content.

NOTE The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” are to be interpreted as described in RFC 2119 [RFC2119]. Augmented Backus-Naur Form (ABNF) [RFC5234] is used to define the syntax of specific files required or recommended by this specification. Syntax rules names left undefined in this specification (for example, *`ALPHA`*) SHALL be interpreted as core ABNF names.

2 Storage service

The Merritt Storage service is based on the following conceptual entities, each defined in terms of its specific state properties.

- Service
- Node
- Object
- Version
- File
- Local-to-primary identifier map

The Storage service provides methods that can be used to manipulate and access these entities and their state in useful ways.

2.1 Service

The initial conceptual entity is the Storage service itself, providing secure, persistent storage of versioned digital objects. The global service state properties MUST minimally include:

• Service name.	[sto:name]
• Service identifier, assigned to be globally unique among all UC3-controlled instantiations.	[sto:identifier]
• Service description.	[sto:description]
• Service implementation version.	[sto:serviceVersion]
• Enumeration of actionable references to all storage node states.	[sto:nodeStates]
o Actionable reference to a storage node.	[sto:nodeState]
• Total number of objects.	[sto:numObjects]
• Total number of versions.	[sto:numVersions]
• Total number of files, when fully instantiated.	[sto:numFiles]
• Total size (in octets), when fully instantiated.	[sto:totalSize]
• Total number of actual files.	[sto:numActualFiles]
• Total size (in octets) of actual files.	[sto:totalActualSize]
• Creation date/timestamp.	[sto:created]
• Modification date/timestamp.	[sto:lastModified]
• Last add version date/timestamp	[sto:lastAddVersion]
• Service specification and version.	[sto:serviceScheme]
• Base URI for the service method invocations.	[sto:baseURI]
• Customer support URI.	[sto:supportURI]

Additional global service state properties MAY be defined and managed by the service.

An instance of the Storage service encapsulates an arbitrary number of storage *nodes*.

2.2 Node

A storage *node* is a digital object store, a system for managing some subset of the objects known to the service. Multiple nodes may be defined within the Storage service to represent different underlying storage technologies (e.g. disk vs. tape; FC vs. SATA), policy regimes (e.g. dark vs. bright), or other significant administrative categories. The node state properties MUST minimally include:

• Node name.	[nod:name
• Node identifier, assigned to be locally unique among all nodes known by a single instance of the storage service.	[nod:identifier]
• Node description.	[nod:description]
• Node implementation version.	[nod:nodeVersion]
• Total number of objects.	[nod:numObjects]
• Total number of versions.	[nod:numVersions]
• Total number of files, when fully instantiated.	[nod:numFiles]
• Total size (in octets), when fully instantiated.	[nod:totalSize]
• Total number of actual files.	[nod:numActualFiles]
• Total size (in octets) of actual files.	[nod:totalActualSize]
• Creation date/timestamp.	[nod:created]
• Modification date/timestamp.	[nod:lastModified]

• Last add version date/timestamp.	[nod:lastAddVersion]
• Underlying storage technology: `magnetic-disk`, `magnetic-tape`, `optical-disk`, `solid-state`.	[nod:mediaType]
• Underlying storage access modality: `on-line`, `near-line`, `off-line`.	[nod:accessMode]
• Pre-read verification status: `true` or `false`.	[nod:verifyOnRead]
• Post-write verification status: `true` or `face`.	[nod:verifyOnWrite]
• Node specification and version.	[nod:nodeScheme]
• Base URI for the node method invocations.	[nod:baseURI]
• Customer support URI.	[nod:supportURI]

Additional node state properties MAY be defined and managed.

If the Storage service encompasses multiple stores, each store MAY be implemented in terms of different technological systems and storage hardware.

An instance of a storage node encapsulates an arbitrary number of *objects*.

2.3 Object

An *object* is a set of digitally encoded files whose change history is aggregated into discrete versions. The object state properties MUST minimally include:

• Object primary identifier, locally unique among all objects managed in a given storage node.	[obj:identifier]
• Object local identifier context. (*optional*)	[obj:localContext]
• Object local identifier(s), meaningful in some external curatorial context. (*optional*)	[obj:localIdentifier]
• Actionable reference to parent storage node state.	[obj:nodeState]
• An enumeration of actionable references to all version states.	[obj:versionStates]
o Actionable reference to a version state.	[obj:versionState]
• Actionable reference to the current version.	[obj:currentVersionState]
• Number of versions.	[obj:numVersions]
• Total number of files, when fully instantiated.	[obj:numFiles]
• Total size (in octets), when fully instantiated.	[obj:totalSize]
• Total number of actual files.	[obj:numActualFiles]
• Total size (in octets) of actual files.	[obj:totalActualSize]
• Creation date/timestamp.	[obj:created]
• Modification date/timestamp.	[obj:lastModified]
• Last add version date/timestamp.	[obj:lastAddVersion]
• Actionable reference to the object.	[obj:object]

NOTE This reference corresponds to the *Get-object* method.

• Object specification and version.	[obj:objectScheme]

An arbitrary number of local identifiers MAY be specified in a semicolon-separated list. Any semicolon embedded in an identifier MUST be represented in the standard ERC escape notation "`%sc`".

Additional object state properties MAY be defined and managed by the service.

NOTE The Merritt Storage service has a weak definition of digital objects, especially in relation to those used by other common repository services. This is by explicit design. The Storage service manages objects without *any* understanding of what information those objects represent. Higher order function implying or depending upon a conceptualization of objects as expressions of intellectual or aesthetic works *must* be provided by other systems.

An instance of a digital object encapsulates an arbitrary number of *versions*.

2.4 Version

A *version* is the particular configuration of an object at a point in time. The version state properties MUST minimally include:

• Version identifier, assigned in incremental numeric order.	[ver:identifier]
• Actionable reference to parent object state.	[ver:objectState]
• Enumeration of actionable references to all file states.	[ver:fileStates]
o Actionable reference to file state.	[ver:fileState]
• Current status: *true* or *false*.	[ver:isCurrent]
• Total number of files, when fully instantiated.	[ver:numFiles]
• Total size (in octets), when fully instantiated.	[ver:totalSize]
• Total number of actual files.	[ver:numActualFiles]
• Total size (in octets) of actual files.	[ver:totalActualSize]
• Creation date/timestamp.	[ver:created]
• Modification date/timestamp.	[ver:lastModified]
• Actionable reference to the version.	[ver:version]

NOTE This reference corresponds to the *Get-version* method.

Additional version state properties MAY be defined and managed by the service.

An instance of a version encapsulates an arbitrary number of *files*.

2.5 File

A *file* is a formatted digital manifestation of a unit of abstract content. The file state properties MUST minimally include:

• File identifier.	[fil:identifier]
• Actionable reference to parent version state.	[fil:versionState]
• File size (in octets).	[fil:size]
• Message digest type, value, and date/timestamp of last verification. Supported digest algorithm types SHOULD minimally include:	[fil:messageDigest]

- Adler-32
- CRC-32
- MD2
- MD5
- SHA-1
- SHA-256

    - SHA-384
    - SHA-512

• Creation date/timestamp.	[fil:created]
• Actionable reference to the file.	[fil:file]

NOTE This reference corresponds to the *Get-file* method.

Additional file state properties MAY be defined and managed by the service.

2.5 Local-to-Primary Identifier Map

A *local-to-primary identifier map* defines the association that may exist between an object's primary identifier and a local identifier and context. The map state properties MUST minimally include:

• Object local identifier context.	[map:localContext]
• Object local identifier.	[map:localIdentifier]
• Map existence: *true* or *false*.	[map:exists]
• Object primary identifier. (*optional*)	[map:identifier]
• Map creation date/timestamp. (*optional*)	[map:created]

An arbitrary number of local identifiers within a given content MAY be associated with the same primary identifier.

Additional version state properties MAY be defined and managed by the service.

3 Service Interface

All Merritt services are defined in terms of abstract interfaces that can be implemented in various interactive modalities, including a procedural API with various language bindings, a command line API supported in various operating system command shells, and a RESTful API [Fielding, 2002].

Containerized objects and versions MAY be requested *by value* in the following forms:

Format	***Extension***	***MIME type***
Tar	`.tar`	`application/tar`
Compressed tar	`.tar.gz`	`application/x-gzip`
Zip	`.zip`	`application/zip`

Table 1 – Container formats

Containerized objects and versions, and uncontainerized files MAY be requested *by reference* in the following manifest format:

Format	***Extension***	***MIME type***
Checkm	`.txt`	`text/checkm`

Table 2 – Container manifest format

NOTE Until such time as a formal MIME type for the Checkm format [Checkm] is established at the IANA registry, the experimental MIME type "`text/x-checkm`" SHOULD be used.

Uncontainerized files are delivered in their original format with their original file name and extension. However, since the storage service does not manage format metadata of individual files, the *reported* format of all delivered files is “`application-octet-stream`”.

State information about the various entities managed by the service MAY be requested in the following formats:

Format	***Extension***	***MIME type***
ANVL	`.txt`	`text/anvl`
JSON	`.json`	`application/json`
RDF/Turtle	`.ttl`	`text/turtle`
XHTML	`.html`	`application/xhtml+xml`
XML	`.xml`	`application/xml`

Table 3 – State formats

NOTE Until such time as formal MIME types for the ANVL [ANVL] and Turtle [Turtle] formats are established at the IANA registry, the experimental MIME types “`text/x-anvl`” and “`text/x-turtle`” SHOULD be used, respectively.

The default format for state information in command line interfaces is ANVL; the default for web interfaces is XHTML.

3.1 RESTful API

The general organization of the RESTful API is to use the HTTP method to indicate the general operation, the request URI to indicate the entity that is being operated on, query string arguments to modify the method, and HTTP content negotiation headers to indicate the desired response form. The general form of the Storage service HTTP request URI is:

```
/entity[/node[/object[/version[/file]]]][?arg[&arg]...]
```

where the leading slash represents the storage service itself; *`entity`* indicates the entity that is the object of the request, and is one of “`help`”, “`state`”, or “`content`”; *`node`* is a storage node identifier; *`object`* is an object identifier; *`version`* is a version number; and *`file`* is a file name. The special version number “`0`” always refers to the current version.

3.2 Command Line API

The commend line API is designed to closely mimic the RESTful API.

RESTful options	***Command line options***		***Function***
	`-C` *`context`*	`--context` *`context`*	Object local identifier context
`D[=`*`level`*`]`	`-D [`*`level`*`]`	`--debug [`*`level`*`]`	Debug level
`f`	`-f`	`--force`	Force a conditional action
`h[=`*`topic`*`]`	`-h [`*`topic`*`]`	`--help [`*`topic`*`]`	Help
	`-I` *`identifier`*	`--identifier` *`identifier`*	Object local identifier
`manifest`	`-M` *`manifest`*	`--manifest` *`manifest`*	Manifest
	`-o` *`file`*	`--output` *`file`*	Output to file (rather than standard output)

`r=`*`mode`*	`-r` *`mode`*	`--response-mode` *`mode`*	Response mode: `by-reference` or `by-value`
`R=`*`mode`*	`-R` *`mode`*	`--request-mode` *`mode`*	Request mode: `by-reference` or `by-value`
`Accept:` *`form`*	`-t` *`form`*	`--response-form` *`form`*	Response format
`Content-type:`	`-T` *`form`*	`--request-form` *`form`*	Request format
`url`	`-U` *`url`*	`--url` *`url`*	URL
`v`	`-v`	`--verbose`	Verbose response
`V`	`-V`	`--version`	Version information
`X`	`-X`	`--expand`	Expand delta-compressed versions

Table 4 – Common API options

4 Service Methods

The Storage service SHOULD support the following methods. Each method is first defined abstractly and then in terms of RESTful and command shell APIs.

NOTE The RESTful API is defined in terms of HTTP request and response messages. The notations "`UA`" and "`OS`" are used to distinguish the User Agent request from the Origin Server response. Names in *italics* indicate arbitrary, rather than fixed values. Brackets "`[`" and "`]`" enclose optional elements and a vertical bar "`|`" separates alternatives.

The methods implicating a storage node are all parameterized with a node identifier. The storage node API implemented by CAN [CAN] is thus identical to the service API without the node parameter.

4.1 Help

METHOD Help			[*idempotent*, *safe*]
Method	Enum	Optional	Specific method about which help is requested.
ResponseForm	Enum	Optional	Response form. The supported forms SHOULD include ANVL (default for command line interfaces), XML, XHTML (default for web interfaces), RDF/Turtle, and JSON.
RETURN	ResponseForm	Mandatory	Help information about the specific method or the service as a whole.
SIDE EFFECTS	—		
ERRORS	400	Badly-formed request.	
	401	Unauthorized user agent.	
	415	Unsupported response form.	
	503	Service unavailable.	
	500	Service error.	

- RESTful API

```
UA: GET /help[?t=form] HTTP/1.x
UA: Host: store.cdlib.org
UA: Accept: response/form
UA:
```

```
OS: HTTP/1.x 200 OK
OS: Content-type: response/form
OS:
OS: help
```

- Command line API

```
% store -h [method] [-t form] [-o file]
% store help [-t form] [-o file]
% store getServiceState -h [-t form] [-o file]
% store getNodeState -h [-t form] [-o file]
% store getobjectState -h [-t form] [-o file]
% store getVersionState -h [-t form] [-o file]
% store getFileState -h [-t form] [-o file]
% store getObject -h [-t form] [-o file]
% store getVersion -h [-t form] [-o file]
% store getFile -h [-t form] [-o file]
% store addVersion -h [-t form] [-o file]
% store deleteObject -h [-t form] [-o file]
% store deleteVersion -h [-t form] [-o file]
```

4.2 Get Service State

METHOD *Get-service-state*			[*idempotent*, *safe*]
—			No argument.
ResponseForm	Enum	Optional	Response form. The supported forms SHOULD include ANVL (default for command line interfaces), JSON, RDF/Turtle, XHTML (default for web interfaces), and XML.
RETURN	Response form	Mandatory	Global service state.
SIDE EFFECTS	—		
ERRORS	400 Badly-formed request.		
	401 Unauthorized user agent.		
	415 Unsupported response form.		
	503 Service unavailable.		
	500 Service error.		

- RESTful API

```
UA: GET /state[?t=form] HTTP/1.x
UA: Host: store.cdlib.org
UA: Accept: response/form
UA:

OS: HTTP/1.x 200 OK
OS: Content-type: response/form
OS:
OS: state
```

- Command line API

```
% store getServiceState [-t form] [-o file]
```

4.3 Get Node State

METHOD *Get-node-state*			[*idempotent*, *safe*]
Node	Identifier	Mandatory	Node identifier
ResponseForm	Enum	Optional	Response form. The supported forms SHOULD include ANVL (default for command line interfaces), JSON, RDF/Turtle, XHTML (default for web interfaces), and XML.
RETURN	Response form	Mandatory	Node state.
SIDE EFFECTS	—		
ERRORS	400 Badly-formed request.		
	401 Unauthorized user agent.		
	404 Node not found.		
	415 Unsupported response form.		
	503 Service unavailable.		
	500 Service error.		

- RESTful API

```
UA: GET /state/node[?t=form] HTTP/1.x
UA: Host: store.cdlib.org
UA: Accept: response/form
UA:

OS: HTTP/1.x 200 OK
OS: Content-type: response/form
OS:
OS: state
```

- Command line API

```
% store getNodeState node [-t form] [-o file]
```

4.4 Get Object State

METHOD *Get-object-state*			[*idempotent*, *safe*]
Node	Identifier	Mandatory	Node identifier
Object	Identifier	Mandatory	Object primary identifier.
ResponseForm	Enum	Optional	Response form. The supported forms SHOULD include ANVL (default for command line interfaces), JSON, RDF/Turtle, XHTML (default for web interfaces), and XML.
RETURN	Response form	Mandatory	Object state.
SIDE EFFECTS	—		
ERRORS	400 Badly-formed request.		
	401 Unauthorized user agent.		
	404 Node not found.		
	404 Object not found.		
	415 Unsupported response form.		
	503 Service unavailable.		
	500 Service error.		

- RESTful API

```
UA: GET /state/node/object[?t=form] HTTP/1.x
UA: Host: store.cdlib.org
UA: Accept: response/form
UA:

OS: HTTP/1.x 200 OK
OS: Content-type: response/form
OS:
OS: state
```

- Command line API

```
% store getObjectState node object [-t form] [-o file]
```

4.5 Get Version State

METHOD *Get-version-state*			[*idempotent*, *safe*]
Node	Identifier	Mandatory	Node identifier.
Object	Identifier	Mandatory	Object primary identifier.
Version	Number	Optional	Version number, defaults to current version. The number "0" indicates current version.
ResponseForm	Enum	Deprecated	Response form. The supported forms SHOULD include ANVL (default for command line interfaces), JSON, RDF/Turtle, XHTML (default for web interfaces), and XML.
RETURN	Response form	Mandatory	Version state.
SIDE EFFECTS	—		
ERRORS	400 Badly-formed request.		
	401 Unauthorized user agent.		
	404 Node not found.		
	404 Object not found.		
	404 Version not found.		
	415 Unsupported response form.		
	503 Service unavailable.		
	500 Service error.		

- RESTful API

```
UA: GET /state/node/object/version[?t=form] HTTP/1.x
UA: Host: store.cdlib.org
UA: Accept: response/form
UA:

OS: HTTP/1.x 200 OK
OS: Content-type: response/form
OS:
OS: state
```

- Command line API

```
% store getVersionState node object [version] [-t form] [-o file]
```

4.6 Get File State

METHOD *Get-file-state*			[*idempotent*, *safe*]
Node	Identifier	Mandatory	Node identifier.
Object	Identifier	Mandatory	Object primary identifier.
Version	Number	Mandatory	Version number, defaults to current version. "0" indicates current version.
File	Identifier	Mandatory	File name.
ResponseForm	Enum	Deprecated	Response form. The supported forms SHOULD include ANVL (default for command line interfaces), JSON, RDF/Turtle, XHTML (default for web interfaces), and XML.
RETURN	Response form	Mandatory	File state.
SIDE EFFECTS	—		
ERRORS	400 Badly-formed request		
	405 Method not allowed		
	404 Node not found.		
	404 Object not found.		
	404 Version not found.		
	404 File not found.		
	415 Unsupported response form.		
	503 Service unavailable.		
	500 Service error.		

- RESTful API

```
UA: GET /state/node/object/version/file[?t=form] HTTP/1.x
UA: Host: store.cdlib.org
UA: Accept: response/form
UA:

OS: HTTP/1.x 200 OK
OS: Content-type: response/form
OS:
OS: state
```

- Command line API

```
% store getFileState node object version file [-t form] [-o file]
```

4.7 Get Object

METHOD *Get-object*			[*idempotent*, *unsafe*]
Node	Identifier	Mandatory	Node identifier.
Object	Identifier	Mandatory	Object primary identifier.
ResponseForm	Enum	Optional	Response form for by-value mode. Supported forms SHOULD include Tar and Zip, with an appropriate default value.
ResponseMode	Enum	Optional	Response mode: *by-reference* (default) or *by-value*.
Expand	Boolean	Optional	If true expand any delta-compressed versions into their fully-instantiated form; otherwise (default) return all versions in the form in which they are managed within the service.
RETURN	Response form	Mandatory	Object files aggregated into a single container.
	Checkm		Manifest containing network-accessible references to object files.
SIDE EFFECTS	Update last access date/timestamp.		
ERRORS	400	Badly-formed request.	
	401	Unauthorized user agent.	
	404	Node not found.	
	404	Object not found.	
	415	Unsupported response form.	
	501	Unsupported response mode.	
	503	Service unavailable.	
	500	Service error.	
NOTE	The structural layout of the object files in the container or manifest reflect the internal structure of the Merritt archival information package (AIP).		

- RESTful API

```
UA: GET /content/node/object[?[r=mode][[&]X]] HTTP/1.x
UA: Host: store.cdlib.org
UA: Accept: response/form
UA:

OS: HTTP/1.x 200 OK
OS: Content-type: response/form | text/checkm
OS:
OS: container | manifest
```

- Command line API

```
% store getObject node object [-t form] [-r mode] [-X] [-o file]
```

4.8 Get Object (DataONE)

METHOD *Get-object-DataONE*			[*idempotent*, *unsafe*]
Node	Identifier	Mandatory	Node identifier.
Object	Identifier	Mandatory	Object primary identifier.
ResponseForm	Enum	Optional	Response form for by-value mode. Supported forms SHOULD include Tar and Zip, with an appropriate default value.
ResponseMode	Enum	Optional	Response mode: *by-reference* (default) or *by-value*.
Expand	Boolean	Optional	If true expand any delta-compressed versions into their fully-instantiated form; otherwise (default) return all versions in the form in which they are managed within the service.
RETURN	Response form	Mandatory	Object files aggregated into a single container.
SIDE EFFECTS	Update last access date/timestamp.		
ERRORS	400 Badly-formed request.		
	401 Unauthorized user agent.		
	404 Node not found.		
	404 Object not found.		
	415 Unsupported response form.		
	501 Unsupported response mode.		
	503 Service unavailable.		
	500 Service error.		
NOTE	The structural layout of the object files in the container conform to the DataONE packaging standard [DataONE], which is based on the BagIt specification [BagIt], regardless of the internal structure of the Merritt archival information package (AIP).		

- RESTful API

```
UA: GET /dataone/node/object[?[r=mode][[&]X]] HTTP/1.x
UA: Host: store.cdlib.org
UA: Accept: response/form
UA:

OS: HTTP/1.x 200 OK
OS: Content-type: response/form
OS:
OS: container
```

- Command line API

```
% store getObjectDataONE node object [-t form] [-r mode] [-X]
                         [-o file]
```

4.9 Get Object (BagIt)

METHOD *Get-object-BagIt*			[*idempotent*, *unsafe*]
Node	Identifier	Mandatory	Node identifier.
Object	Identifier	Mandatory	Object primary identifier.
ResponseForm	Enum	Optional	Response form for by-value mode. Supported forms SHOULD include Tar and Zip, with an appropriate default value.
ResponseMode	Enum	Optional	Response mode: *by-reference* (default) or *by-value*.
Expand	Boolean	Optional	If true expand any delta-compressed versions into their fully-instantiated form; otherwise (default) return all versions in the form in which they are managed within the service.
RETURN	Response form	Mandatory	Object files aggregated into a single container.
SIDE EFFECTS	Update last access date/timestamp.		
ERRORS	400	Badly-formed request.	
	401	Unauthorized user agent.	
	404	Node not found.	
	404	Object not found.	
	415	Unsupported response form.	
	501	Unsupported response mode.	
	503	Service unavailable.	
	500	Service error.	
NOTE	The structural layout of the object files in the container conform to the BagIt specification [BagIt], regardless of the internal structure of the Merritt archival information package (AIP).		

- RESTful API

```
UA: GET /bagit/node/object[?[r=mode][[&]X]] HTTP/1.x
UA: Host: store.cdlib.org
UA: Accept: response/form
UA:

OS: HTTP/1.x 200 OK
OS: Content-type: response/form
OS:
OS: container
```

- Command line API

```
% store getObjectBagIt node object [-t form] [-r mode] [-X] [-o file]
```

4.10 Get Version

METHOD *Get-version*			[*idempotent*, *unsafe*]
Node	Identifier	Mandatory	Node identifier.
Object	Identifier	Mandatory	Object primary identifier.
Version	Number	Mandatory	Version number. The number “0” indicates the current version.
ResponseMode	Enum	Optional	Response mode: *by-reference* (default) or *by-value*
ResponseForm	Enum	Optional	Response form for *by-value* mode. Supported forms SHOULD include Tar and Zip, with an appropriate default value.
RETURN	*Response form*	Mandatory	Version files aggregated into a single container.
	Checkm		Manifest containing network-accessible references to version files.
SIDE EFFECTS	Update last access date/timestamp.		
ERRORS	400 Badly-formed request.		
	401 Unauthorized user agent.		
	404 Node not found.		
	404 Object not found.		
	404 Version not found.		
	415 Unsupported response form.		
	501 Unsupported response mode.		
	503 Service unavailable.		
	500 Service error.		
NOTE	The structural layout of the version files in the container or manifest reflect the internal structure of the Merritt archival information package (AIP).		

- RESTful API

```
UA: GET /content/node/object/version[?r=mode] HTTP/1.x
UA: Host: store.cdlib.org
UA: Accept: response/form
UA:
OS: HTTP/1.x 200 OK
OS: Content-type: response/form | text/ checkm
OS:
OS: container | manifest
```

- Command line API

```
% store getVersion node object [version] [-t form] [-r mode] [-o file]
```

4.11 Get Version (BagIt)

METHOD *Get-version-BagIt*			[*idempotent*, *unsafe*]
Node	Identifier	Mandatory	Node identifier.
Object	Identifier	Mandatory	Object primary identifier.
Version	Number	Mandatory	Version number. The number "0" indicates the current version.
ResponseMode	Enum	Optional	Response mode: *by-reference* (default) or *by-value*
ResponseForm	Enum	Optional	Response form for *by-value* mode. Supported forms SHOULD include Tar and Zip, with an appropriate default value.
RETURN	*Response form*	Mandatory	Version files aggregated into a single container.
SIDE EFFECTS	Update last access date/timestamp.		
ERRORS	400 Badly-formed request.		
	401 Unauthorized user agent.		
	404 Node not found.		
	404 Object not found.		
	404 Version not found.		
	415 Unsupported response form.		
	501 Unsupported response mode.		
	503 Service unavailable.		
	500 Service error.		
NOTE	The structural layout of the version files in the container conform to the BagIt specification [BagIt], regardless of the internal structure of the Merritt archival information package (AIP).		

- RESTful API

```
UA: GET /bagit/node/object/version[?r=mode] HTTP/1.x
UA: Host: store.cdlib.org
UA: Accept: response/form
UA:
OS: HTTP/1.x 200 OK
OS: Content-type: response/form
OS:
OS: container
```

- Command line API

```
% store getVersionBagIt node object [version] [-t form] [-r mode]
                        [-o file]
```

4.12 Get Version (DataONE)

METHOD *Get-version-DataONE*			[*idempotent*, *unsafe*]
Node	Identifier	Mandatory	Node identifier.
Object	Identifier	Mandatory	Object primary identifier.
Version	Number	Mandatory	Version number. The number "0" indicates the current version.
ResponseMode	Enum	Optional	Response mode: *by-reference* (default) or *by-value*
ResponseForm	Enum	Optional	Response form for *by-value* mode. Supported forms SHOULD include Tar and Zip, with an appropriate default value.
RETURN	*Response form*	Mandatory	Version files aggregated into a single container.
SIDE EFFECTS	Update last access date/timestamp.		
ERRORS	400 Badly-formed request.		
	401 Unauthorized user agent.		
	404 Node not found.		
	404 Object not found.		
	404 Version not found.		
	415 Unsupported response form.		
	501 Unsupported response mode.		
	503 Service unavailable.		
	500 Service error.		
NOTE	The structural layout of the version files in the container conform to DataONE packaging standard [DataONE], which is based on the BagIt specification [BagIt], regardless of the internal structure of the Merritt archival information package (AIP).		

- RESTful API

```
UA: GET /dataeone/node/object/version[?r=mode] HTTP/1.x
UA: Host: store.cdlib.org
UA: Accept: response/form
UA:
OS: HTTP/1.x 200 OK
OS: Content-type: response/form
OS:
OS: container
```

- Command line API

```
% store getVersionDataONE node object [version] [-t form] [-r mode]
                          [-o file]
```

4.13 Get File

METHOD *Get-file*			[*idempotent*, *unsafe*]
Node	Identifier	Mandatory	Node identifier.
Object	Identifier	Mandatory	Object primary identifier.
Version	Number	Mandatory	Version number. The number "`0`" indicates current version.
File	Identifier	Mandatory	File name.
ResponseMode	Enum	Optional	Response mode: *by-reference* or *by-value* (default)
Force	Boolean	Optional	If true and the global "`Verify-on-read`" property is set and the verification fails, nevertheless deliver the file.
RETURN	Octet-stream	Mandatory	File content.
	Checkm		Manifest containing network-accessible reference to file content.
SIDE EFFECTS	Update last access date/timestamp.		
ERRORS	400 Badly-formed request.		
	401 Unauthorized user agent.		
	404 Node not found.		
	404 Object not found.		
	404 Version not found.		
	404 File not found.		
	415 Unsupported response mode.		
	503 Service unavailable.		
	500 Service error.		

- RESTful API

```
UA: GET /content/node/object[/version]/file[?[r=mode][[&]f]] HTTP/1.x
UA: Host: store.cdlib.org
UA:
OS: HTTP/1.x 200 OK
OS: Content-type: mime/type | text/checkm
OS:
OS: content | manifest
```

- Command line API

```
% store getFile node object version file [-t form] [-r mode] [-f]
  [-o file]
```

4.14 Add Version

METHOD *Add-version*			[*non-idempotent*, *unsafe*]
Node	Identifier	Mandatory	Node identifier.
Object	Identifier	Mandatory	Object primary identifier.
RequestMode	Enum	Optional	Request mode: *by-reference* (default) or *by-value*.
URL	URL	Mandatory	If request mode is *by-reference*, URL of manifest containing network accessible references to version files.
Manifest	Checkm	Mandatory	If request mode is *by-reference*, manifest containing network-accessible references to version files.
Size	Number	Optional	Manifest size, in octets.
DigestType	Enum	Optional	Manifest message digest type. The supported types SHOULD include Adler-32, CRC-32, MD2, MD5, SHA-1, SHA-256, SHA-384, and SHA-512.
DigestValue	String	Optional	Hexadecimal representation of the manifest message digest value.
LocalContext	Identifier	Optional	Object local identifier context.
LocalIdentifier	Identifier	Optional	Object local identifier list. **
ResponseForm	Enum	Optional	Response form. The supported forms SHOULD include ANVL (default for command line interfaces), JSON, RDF/Turtle, XHTML (default for web interfaces), and XML.
RETURN	Response form	Mandatory	Newly created version state.
SIDE EFFECTS	If the object does not exist, create an empty object; increment current version number; populate the version; establish appropriate version state; and update object, node, and service state. A persistent mapping is maintained between the local identifier and context, if supplied, and the object primary identifier, for use by the *Get-primary-identifier* method.		
ERRORS	400 Badly-formed request.		
	401 Unauthorized user agent.		
	404 Node not found.		
	404 Object not found.		
	400 Unsupported request mode.		
	415 Unsupported request form.		
	415 Unsupported response form.		
	413 Version too large.		
	400 Empty version		
	400 Duplicate version		
	503 Service unavailable.		
	500 Service error.		
NOTE	*Any* change introduced to an object SHALL result in a new version. In particular, the concept of a *replace* or *update-in-place* method is *not* supported. When adding a subsequent version to an existing object, only the file components that differ between the versions need to be supplied as method parameters. Changed or added files are enumerated directly in the Checkm manifest. Files to be deleted are enumerated in a special file with the reserved name, "mrt-store-delete.txt", which is itself included in the Checkm manifest. Files that are not included in the manifest or in the deletion file are assumed to be unchanged between versions.		

** Multiple local identifiers MAY be supplied as a semicolon-separated list. Any semicolon embedded in an identifier MUST be represented in the standard ERC escape notation “%sc”.

- RESTful API

```
UA: POST /content/node/object[?t=form] HTTP/1.x
UA: Host: store.cdlib.org
UA: Accept: response/form
UA: Content-type: multipart/form-data; boundary=boundary
UA:
UA: --boundary
UA: ( Content-disposition: form-data; name=”url”
UA: Content-type: application/x-www-form-urlencoded
UA:
UA: url ) |
UA: ( Content-disposition: form-data; name=”manifest”
UA: Content-type: text/checkm
UA:
UA: manifest )
UA: --boundary
UA: [ Content-disposition: form-data; name=”size”
UA: Content-type: text/plain
UA:
UA: size
UA: --boundary ]
UA: [ Content-disposition: form-data; name=”digest-type”
UA: Content-type: text/plain
UA:
UA: type
UA: --boundary
UA: Content-disposition: form-data; name=”digest-value”
UA: Content-type: text/plain
UA:
UA: value
UA: --boundary ]
UA: [ Content-disposition: form-data; name=”local-context”
UA: Content-type: text/plain
UA:
UA: context
UA: --boundary
UA: Content-disposition: form-data; name=”local-identifier”
UA: Content-type: text/plain
UA:
UA: identifier[; identifier[; ...]]
UA: --boundary ]
UA: [ Content-disposition: form-data; name=”response-form”
UA: Content-type: text/plain
UA:
UA: form
UA: --boundary ]

OS: HTTP/1.x 201 CREATED
OS: Content-type: response/form
OS: Location: http://store.cdlib.org/state/node/object/version
OS:
OS: state
```

NOTE The RequestMode argument is not explicitly represented in a RESTful request; its value is implied by the presence of either the "`url`" (*by-reference*) or "`manifest`" (*by-value*) named parts.

- Command line API

```
% store addVersion node object -U url | -M manifest [-T mode]
        [-t form] [-o file]
```

An *Add-version* manifest is a Checkm manifest listing the locations of version files. The Checkm profile for the manifest (`http://uc3.cdlib.org/registry/store/manifest-mrt-add-manifest`) is defined as follows:

- The URL, SHA-256 digest type and value, file size, and target filename fields MUST be specified.
- The modification time field SHOULD NOT be specified, and SHALL be ignored if provided.
- The first two entries in the manifest MUST be structured comments specifying the Checkm conformance level and profile identifier.
- The next three entries SHOULD be structured comments specifying namespace prefixes and field definitions.
- The last entry in the manifest SHOULD be a structured comment explicitly representing the end of the file.

```
  #%checkm_0.7
  #%profile | http://uc3.cdlib.org/registry/store/manifest/mrt-add-manifest
[ #%prefix  | mrt: | http://uc3.cdlib.org/ontology/mom# ]
[ #%prefix  | nfo: | http://www.semanticdesktop.org/ontologies/2007/03/22/nfo# ]
[ #%fields  | nfo:fileUrl  | nfo:hashAlgorithm    | nfo:hashValue |
              nfo:fileSize | nfo:fileLastModified | nfo:fileName
  @url | sha256 | value | size | | filename
  ...
[ #%eof ]
```

4.15 Delete Object

METHOD *Delete-object*			[*idempotent*, *unsafe*]
Node	Identifier	Mandatory	Node identifier.
Object	Identifier	Mandatory	Object primary identifier.
ResponseForm	Enum	Optional	Response form. The supported forms SHOULD include ANVL (default for command line interfaces), JSON, RDF/Turtle, XHTML (default for web interfaces), and XML.
RETURN	Response form	Mandatory	Deleted object state.
SIDE EFFECTS	Delete the object and update node and service state.		
ERRORS	400 Badly-formed request.		
	401 Unauthorized user agent.		
	404 Node not found.		
	404 Object not found.		
	415 Unsupported response form.		
	503 Service unavailable.		
	500 Service error.		

- RESTful API

```
UA: DELETE /content/node/object[?t=form] HTTP/1.x
UA: Host: store.cdlib.org
UA: Accept: response/form
UA:

OS: HTTP/1.x 202 Accepted
OS: Content-type: response/form
OS:
OS: state
```

- Command line API

```
% store deleteObject node object [-t form] [-o file]
```

4.16 Delete Version

METHOD *Delete-version*			[*idempotent*, *unsafe*]
Node	Identifier	Mandatory	Node identifier.
Object	Identifier	Mandatory	Object primary identifier.
Version	Number	Mandatory	Version number. The number "0" indicates the current version.
ResponseForm	Enum	Optional	Response form. The supported forms SHOULD include ANVL (default for command line interfaces), JSON, RDF/Turtle, XHTML (default for web interfaces), and XML.
RETURN	Response form	Mandatory	Deleted version state.
SIDE EFFECTS	Delete the object version and update object, node, and service state.		
ERRORS	400 Badly-formed request.		
	401 Unauthorized user agent.		
	404 Node not found.		
	404 Object not found.		
	404 Version not found.		
	415 Unsupported response form.		
	503 Service not available.		
	500 Service error.		

- RESTful API

```
UA: DELETE /content/node/object/version[?t=form] HTTP/1.x
UA: Host: store.cdlib.org
UA: Accept: response/form
UA:

OS: HTTP/1.x 202 Accepted
OS: Content-type: response/form
OS:
OS: state
```

- Command line API

```
% store deleteVersion node object version [-t form] [-o file]
```

4.17 Get Primary Identifier

METHOD *Get-primary-identifier*			[*idempotent*,*safe*]
Node	Identifier	Mandatory	Node identifier.
Context	Identifier	Mandatory	Object local identifier context.
Local	Identifier	Mandatory	Object local identifier.
ResponseForm	Enum	Optional	Response form. The supported forms SHOULD include ANVL (default for command line interfaces), JSON, RDF/Turtle, XHTML (default for web interfaces), and XML.
RETURN	*ResponseForm*	Mandatory	Object local-to-primary identifier map state.
SIDE EFFECTS	—		
ERRORS	400	Badly-formed request.	
	401	Unauthorized user agent.	
	404	Node not found	
	503	Service unavailable.	
	500	Service error	
NOTE	The responsibility for maintaining a persistent mapping from an object's local identifier and context to its primary identifier is the responsibility of the storage node in which the object is managed.		

- RESTful API

```
UA: GET /primary/node/context/identifier[?t=form] HTTP/1.x
UA: Host: store.cdlib.org
UA: Accept: response/form
UA:

OS: HTTP/1.x 200 OK
OS: Content-type: response/form
OS:
OS: state
```

- Command line API

```
% store getPrimaryIdentifier node context identifier
```

5 Implementation

The Storage service exposes its function via three interactive modalities:

- A Java J2SE 1.6 API
- A command line application based on the Java API
- A web interface based on Jetty, Jersey, and the Java API

Although the method names (e.g. "`getServiceState`", "`addVersion`") used in the Storage service command line application are documented above in camel-case, they SHALL be parsed and matched in a case-insensitive manner. Similarly, although the documented method syntax uses the terse, single letter form of command options (e.g. "`-t` *`form`*"), the analogous keyword variants ("`--response-format` *`form`*") MUST also be accepted. The command line application SHALL support the "`-v`" and "`--version`" (version) command options.

Although a creation date/timestamp is documented above as part of the service, node, object, version, and file state, it SHALL not be managed or reported.

NOTE File and directory creation times are not directly retrievable in J2SE 1.6.

The Delete-version method is only valid when invoked against the *current* version, thus the required syntax MUST be:

```
UA: DELETE /content/node/object/0 HTTP/1.x
```

The Storage service uses CAN (Content Access Node) as the implementation for its storage nodes [CAN]; Pairtree to structure the branches of the CAN hierarchical object stores [Pairtree]; Dflat to structure the leaves of the hierarchical object stores [Dflat]; Checkm to define Dflat manifests [Checkm]; and ReDD to delta-compress Dflat object versions [ReDD].

The Storage service imposes the following additional requirements beyond these specified by the CAN specification:

- The Namaste tag is REQUIRED.
- The global properties file "`can-info.txt`" is REQUIRED and SHALL express the "`name`", "`identifier`", "`description`", "`nodeScheme`", "`branchScheme`", "`leafScheme`", "`mediaType`", "`accessMode`", "`verifyOnRead`", "`verifyOnWrite`", and "`supportURI`" properties.
- The log directory "`log`" is REQUIRED and SHALL hold the file "`last-activity.txt`" and "`summary-stats`" files.

The Storage service imposes the following additional requirements beyond these specified by the Dflat specification:

- The Namaste tag is REQUIRED.
- The current version file "`current.txt`" is REQUIRED.
- The global properties file "`dflat-info.txt`" is REQUIRED and SHALL express the "`objectScheme`", "`manifestScheme`", "`fullScheme`", "`deltaScheme`", and "`currentScheme`" properties.
- The log directory "`log`" is REQUIRED and SHALL hold the files "`last-activity.txt`" and "`summary-stats.txt`".

- The current version sub-directory SHALL contain a version manifest file "`manifest.txt`".
- All non-current version sub-directories SHALL be delta-compressed representations and SHALL contain delta directory and version manifest files, "`d-manifest.txt`" and "`manifest.txt`", respectively.

The Storage service is instantiated in a file system with the following structure:

```
<store_home>/
              0=store_0.17
              admin/
            [ lock.txt ]
              log/
              nodes.txt
              store-info.txt
```

The REQUIRED file "`0=store_0.17`" is the Storage service's Namaste signature. .A Namaste signature plays the same role for a directory that a magic number plays for a file.

The REQUIRED directory "admin" holds administrative declarations about the service itself, as opposed to the objects managed in it.

The OPTIONALfile "`lock.txt`" indicates that a write operation is in-process and that the service may be in a temporarily unknown, inconsistent, or incomplete state. If present, the lock file MUST conform to the syntax, semantics, and normative obligations defined by the LockIt specification [LockIt].

The REQUIRED file "`nodes.txt`" defines the set of storage nodes known to the service.

The REQUIRED file "`store-info.txt`" defines the global properties of the service.

Within the file system hierarchy rooted at a Storage service home directory, all file names starting with "`store`", "`sto`", "`merritt`", or "`mrt`", on a case-insensitive basis, are reserved.

5.1 Namaste signature file (0=store_<*version*>)

The REQUIRED Namaste signature file "`0=store_0.17`" self-identifies a directory as a Storage service home directory. The "<*`version`*>" portion of the file name asserts the version of the service specification to which the directory conforms. If present, the contents of the file MUST contain the specification name and version, separated by a forward slash, followed by a CR, CRLF, or LF end-of-line (EOL) marker.

```
Store/0.17
```

Note that this value is duplicative of the "`serviceScheme`" property of the global properties file "`store-info.txt`". If both are present, the global properties file is considered authoritative.

5.2 Administrative directory (admin/)

The administrative directory "`admin`" holds administrative declarations associated with the Storage service itself, as opposed to the objects managed in it.

5.3 Log directory (log/)

The logging directory "`log`" holds log information about the use of the Storage service and the objects managed in it.

```
log/
    cmd-access.txt
  [ diagnostics.txt ]
    last-actvity.txt
  [ lock.txt ]
  [ log-<month><month><day> ]
        web-access.txt
```

The directory MUST contain a file named "`cmd-access.txt`" containing a log entry for each invocation of a Storage service method via the command line API. The directory SHALL contain a file named "`web-access.txt`" containing a log entry for each invocation of a Storage service method via the web API. The format of these log files is the Combined Log Format (CLF).

```
ip - - date-time "request-uri" status size "referer" "user-agent"
```

where "*`ip`*" is the IP address of the client; "*`date-time`*" is the date/timestamp of the request; "*`request-uri`*" is the request URI; "*`status`*" is the status of the response; "*`size`*" is the size of the response, in octets; "*`referer`*" is the referring URI; and "*`user-agent`*" is the requesting user agent. For the command line log, "*`ip`*" SHOULD be set to the Storage service name (the "`Name`" property in the "`store-info.txt`" file); "`request-uri`" SHOULD be set to the equivalent web API syntax; "*`referer`*" SHOULD be set to the current working directory (e.g. the value of Java `user.dir` or Unix `$cwd` property); and "*`user-agent`*" SHOULD be set to the command shell login name of the user (e.g. the value of.Java `user.name` or Unix `$user` property)

The directory MAY contain a file named "`diagnostics.txt`" containing diagnostic logging information.

The directory MUST contain a file named "`last-activity.txt`" containing the date/timestamp. in fully-qualified W3C form [DateTime], of the last instances of various activities performed on the objects in the Storage service and a unique identifiers (such as process numbers or thread identifiers) of those activities, for example:

```
lastSddVersion: 2008-11-23T08:17:53-08:00
lastObjectDelete: 2008-05-03T13:04:12-08:00
lastFixity: 2008-12-22T23:00:10-08:00
```

The sub-directory MAY contain a file named in the form "`log-<year><month><day>.txt`", that holds log information for the Storage service for the specified month. Log files for previous months MAY be compressed using GZIP [RFC1952] or ZIP [ZIP], resulting in file names of the form, "`log-<year><month><day>.txt.gz`" and "`log-<year><month><day>.txt.zip`", respectively.

5.4 Storage nodes file (nodes.txt)

The REQUIRED file "`nodes.txt`" defines the nodes known to the Storage service, for example:

```
http://can01.cdlib.org/
http://can02.cdlib.org/
file:///home/can03
/usr/canusr/mycan
...
```

Local storage nodes, i.e. those instantiated in file systems mountable by the server on which the Storage service is running, MUST be specified in terms of "`file`" scheme URIs or relative or absolute file pathnames; remote storage nodes MUST be specified in terms of "`http`" URIs.

5.5 Global properties file (store-info.txt)

The REQUIRED file "`store-info.txt`" declares the global Storage service properties, for example:

```
name: UC3
description: UC3 storage micro-service
serviceScheme: Store/0.8/0.1
baseURI: http://store.cdlib.org/
supportURI: mailto:merritt-support@ucop.edu
```

The "`name`" property indicates the name of the Storage service, which SHOULD be assigned to be unique within the administrative regime in which the service operates.

The "`description`" property provides a short textual description of the Storage service.

The "`serviceScheme`" property indicates the version of the Storage service specification to which the service conforms, and optionally, the version of the service implementation. Note that this specification version is duplicative to the information provided by the Namaste tag. If both are present but differ, the properties file is considered authoritative

The "`baseURI`" property indicates the base URI used for user access to the service.

The "`supportURI`" property indicates the URI for user support.

Within a properties file all property names starting with "`store`", "`sto`", "`merrit`", or "`mrt`", on a case-insensitive basis, are reserved.

Appendix A: Complete Storage Service Conventions

The overall file system structure of the Storage service is:

```
<store_home>/
             0=store_<version>
             admin/
```

```
              [ lock.txt ]
              [ log/
                  [ last-actvity.txt ]
                  [ lock.txt ]
                  [ log-<year><month><day>.txt
                  [ log-<year><month><day>.txt.gz ]
                  [ log-<year><month><day>.txt.zip ]]
                store-info.txt
```

The production rule for the Storage service Namaste signature file “`0=store_<version>`” is:

```
<storesig>     = “Store/” <version> EOL
<version>      = NONNEG 0*(“.” 1*DIGIT)
NONNEG         = “0” / (1*POSDIG 0*DIGIT)
POSDIG         = “1” / “2” / “3” / “4” / “5” / “6” / “7” / “8” / “9”
EOL            = CR / CRLF / LF
```

The generic production rules for all ANVL-based files are:

```
<file>         = 1*<line>
<line>         = <name> “:” 1*WSP <value> EOL
<name>         = 1*VCHAR
<value>        = 1*VCHAR
```

Matching of all ANVL property names MUST be performed on a case-insensitive basis. More specific rules MAY be defined by each such ANVL-based file. It is RECOMMENDED that only a single white space character (WSP) is used to separate the name/value pairs in these files.

The production rule for the last activity log file “`last-activity.txt`” is:

```
<activity>     = 1*(<activity> 1*WSP <date-time> EOL)
<activity>     = <add> / <object> / <version> / <fixity>
<add>          = “lastAddVersion:”
<object>       = “lastDeleteObject:”
<version>      = “lastDeleteVersion:”
<fixity>       = “lastFixity:”
<date-time>    = <date> “T” <time>
<date>         =  <year> “-“ <month> “-“ <day>
<year>         = 4DIGIT
<month>        = 2DIGIT             ; normal constraints apply, 01-12
<day>          = 2DIGIT             ; normal constraints apply, 01-31
<time>         = <hour> “:” <minute> “:” <second> <zzzz>
<hour>         = 2DIGIT             ; normal constraints apply, 00-23
<minute>       = 2DIGIT             ; normal constraints apply, 00-59
<second>       = 2DIGIT             ; normal constraints apply, 00-59
<zzzz>         = “Z” / ((“+” | “-” ) <hour> “:” <minute>)
<process>      = 1*VCHAR
```

It is RECOMMENDED that only a single white space character (WSP) is used to demarcate the date/timestamp and process identifier.

The production rules for the nodes file “`nodes.txt`” are:

```
<nodes>          = 1 <node> EOL)
<node>           = <rfc-3986-compliant-uri> / <posix-file-pathname>
```

The production rules for the global service properties file “`store-info.txt`” are:

```
<properties>    = 1*(<property> EOL )
<property>      = <name> / <identifier> / <description> /
                  <service> / <base> / <support>
<name>          = “name:” 1*WSP 1*VCHAR
<identifier>    = “identifier:” 1*WSP 1*VCHAR
<description>   = “description:” 1*WSP 1*VCHAR
<service>       = “serviceScheme:” 1*WSP <scheme>
<scheme>        = <name> “/”<version> [ “/” <version> ]
<name>          = 1*VCHAR
<base>          = “baseURI:” <rfc-3986-compliant-uri>
<support>       = “supportURI:” <rfc-3986-compliant-uri>
```

The first version of a `<scheme>` refers to the specification to which the scheme conforms; the second, optional version refers to the implementation of the scheme.

References

[ANVL] UC3, *ANVL: A Name Value Language*, 2009.

[Abott] Daisy Abbott,*What is Digital Curation?* April 3, 2008 <http://www.dcc.ac.uk/resource/briefing-papers/what-is-digital-curation/>.

[BagIt] A. Boyko, J. Kunze, J. Littman, L. Madden, and B. Vargas, *The BagIt File Packaging Format*, version 0.97, April 15, 2011 <http://tools.ietf.org/html/draft-kunze-bagit-06>.

[CAN] UC3, *CAN: A Simple File System-Based Object Store*, 2010.

[Checkm] UC3, *Checkm: A Checksum-Based Manifest Format*, 2009.

[DataONE] DataONE, *Data Packaging*, July 19, 2011 <http://mule1.dataone.org/ArchitectureDocs-current/design/DataPackage.html>.

[Denning] Peter J. Denning, Chris Gunderson, and Rich Hayes-Roth, “Evolutionary system development,” *Communications of the ACM* 51:17 (December 2008): 29-31.

[DateTime] Misha Wolf and Charles Wicksteed, *Date and Time Formats*, September 15, 1997 <http://www.w3.org/TR/NOTE-datetime>.

[Dflat] UC3, *Dflat: A Simple File System Convention for Digital Object Storage*, 2010.

[Fielding] Roy Fielding and Richard Taylor, “Principled design of the modern web architecture,” *ACM Transactions on Internet Technology* 2:2 (May 2002): 115-150 <doi:10.1145/514183.514185>.

[Foundations] UC3, *Digital Preservation Program: Foundations*, 2010.

[LockIt] UC3, *LockIt: A Simple File-based Convention for Resource Locking*, 2009.

[Merritt] UC3, *Merritt: An Emergent Approach to Digital Curation Infrastructure*, 2010.

[ReDD] UC3, *Reverse Directory Deltas (ReDD)*, 2009.

[RFC1952] P. Deutsch, *GZIP File Format Specification 4.3*, RFC 1952, May 1996 <http://www.ietf.org/rfc/rfc1952.txt>.

[RFC2119] S. Bradner., *Key Words for Use in RFCs to Indicate Requirement Levels*, BCP 14, RFC 2119, March 1997 <http://www.ietf.org/rfc/rfc2119.txt>.

[RFC5234] D. Crocker (ed.) and P. Overell, *Augmented BNF for Syntax Specifications: ABNF*, STD 68, RFC 5234, January 2008 <http://www.ietf.org/rfc/rfc5234.txt>.

[ZIP] PKWARE, Inc., *.ZIP File Format Specification*, Version 6.3.2, September 28, 2007 <http://www.pkware.com/documents/casestudies/APPNOTE.TXT>.