UC3Merritt

University of California Curation Center

CAN: A Simple File System-Based Object Store

Rev. 0.15 – 2012-02-23

1 Introduction

A Content Access Node (CAN) is a simple file system convention for storing digital objects. It imposes minimal architectural and policy constraints while reserving a small set of file system names (directories and files) that place certain salient object store features, if available, in well-known locations within a single directory hierarchy that comprises the object store.

While CAN can be deployed usefully on its own, it was designed to interoperate cleanly with other independent, but related, specifications such as Pairtree [Pairtree], and Dflat [Dflat]. All three of these specifications start from the assumption that a file system is an appropriate storage abstraction for the effective management of digital objects. Assuming that these objects are arranged in a tree-like directory hierarchy, CAN specifies the organization and global properties of the tree; Pairtree specifies the form of the branches of the tree; and Dflat specifies the local structure of the leaves of the tree.

2 Notation

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119 [RFC2119].

Note that some care MUST be taken in reading this specification so as not to misinterpret uses of the acronym “CAN” for an imperative indicating optionality. The term “MAY” will always be used to indicate such optionality.

Angle brackets and italics are used to indicate an arbitrary, as opposed to prescribed, file or directory name; for example, the arbitrarily-named file “*<file>*”. When referring to directory names and symbolic links in examples, the names are followed by a solidus (“/”), for example “`directory/`”; this suffix is indicative of type and is not part of the name. All directory and file names are case sensitive.

In examples of file system directory hierarchies, non-required directories or files are enclosed by square brackets (“[“ and “]”), a number sign (“#”) introduces an informative comment, and an ellipsis (“...”) indicates arbitrary repetition of the previous element.

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.

This specification is intended to be applicable, and implementable, in both Unix/Linux and Windows/DOS environments. Consequently, all uses of the term “directory” are interchangeable with “folder” without loss of meaning. Similarly, all uses of the solidus (“/”) as a directory path separator

are interchangeable with a reverse solidus ("\").

The complete set of CAN conventions is provided in Appendix A.

3 Content Access Node

A Content Access Node, or CAN, is a file system hierarchy that comprises a store for digital objects. The directory that is the structural root of a CAN hierarchy is known as the CAN *home directory*. The CAN specification reserves a few key file system names within the home directory and its descendent sub-directories, but it does not dictate how the home directory itself is named, as that name is not visible from inside the CAN itself.

A CAN looks like this:

```
<can_home>/                          # CAN home directory
          [ 0=can_0.15 ]             # CAN Namaste signature
          [ admin/ ]                 # administrative declarations
          [ can-info.txt ]           # CAN properties file
          [ lock.txt ]               # write lock
          [ log/ ]                   # log directory
            store/                   # object store
```

A CAN home directory SHOULD contain a file named "0=`can_0.15`" that is its Namaste [Namaste] signature. A Namaste signature plays the same role for a directory that a magic number plays for a file.

The home directory MAY contain a sub-directory named "`admin`" that holds administrative declarations about the CAN itself, as opposed to the objects managed in it.

The CAN home directory SHOULD contain a file named "`can-info.txt`" that defines the global properties of the CAN itself.

The home directory MAY contain a file named "`lock.txt`" that indicates a write operation is in-process and that the CAN 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 home directory MAY contain a sub-directory named "`log`" holding log information about the use of the CAN and the objects managed in it.

The home directory MUST contain a directory named "`store`", which is the root of the hierarchical object store.

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

3.1 Namaste signature file (0=can_<*version*>)

UC3Merritt

The RECOMMENDED Namaste signature file "0=can_0.15" self-identifies a directory as a CAN home directory. The "<*version*>" portion of the file name asserts the version of the CAN 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.

```
CAN/0.12
```

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

3.2 Administrative directory (admin/)

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

```
admin/
```

3.3 Properties file (can-info.txt)

The RECOMMENDED properties file "can-info.txt" defines the global properties of that CAN itself, as opposed to the objects managed in it. These properties are expressed in terms of ANVL [ANVL] name/value pairs, for example:

```
  name: Primary
  identifier: 12
  description: Primary UC3 storage node
  nodeScheme: CAN/0.15/0.3
  branchScheme: Pairtree/0.1/0.2
  leafScheme: Dflat/0.19/0.3
  mediaType: magnetic-disk
  mediaConnectivity: SAN
  accessMode: on-line
  accessProtocol: ZFS
[ logicalVolume: dprstore ]
[ externalProvider: sdsc ]
  verifyOnRead: true
  verifyOnWrite: true
  baseURI: http://can01.cdlib.org/
  supportURI: mailto:merritt-support@ucop.edu
```

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

The "identifier" property is an identifier for the CAN that MUST be assigned to be unique within the administrative regime in which the CAN operates.

The "description" property provides a short textual description of the CAN.

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

The "branchScheme" property indicates the specification name and version of the convention used for the branches of the object store hierarchy.

The "leafScheme" property indicates the specification name and version of the convention used for the leaves of the object store hierarchy.

The "mediaType" property indicates the storage technology underlying the CAN: magnetic disk, magnetic tape, optical disk, solid-state, or content-addressable.

The "mediaConnectivity" property indicates the networking technology used between the CAN server and the storage media: direct, NAS, SAN, or cloud.

The "accessMode" property indicates level of availability of stored data: on-line, near-line, or off-line.

The "accessProtocol" property indicates the protocol used to access the underlying storage: ZFS or S3.

The optional "logicalVolume" property indicates the name of the logical volume of the underlying storage.

The optional "externalProvider" property indicates the external service provider supporting the underlying storage.

The "verifyOnRead" property indicates whether a message digest verification SHOULD be performed prior to responding to a request to retrieve an object, version, or file (but *not* their states) from the CAN. The "verifyOnWrite" property indicates whether a message digest verification SHOULD be performed following the addition of a new object version to the CAN. A directive to verify on read or write MAY be ignored if the storage media underlying the CAN is not amenable to such verification, as may be case, for example, with tape storage.

The "baseURI" property indicates the base URI used for CAN method invocations.

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

Within a properties file all property names starting with "can", "merritt", or "mrt", on a case-insensitive basis, are reserved

3.4 Log directory (log/)

The OPTIONAL "`log`" directory log holds log information about the use of the CAN and the objects managed in it.

```
log/
  [ last-actvity.txt ]
  [ lock.txt ]
  [ log-<year><month><day>.txt ]
  [ summary-stats.txt ]
```

The directory SHOULD contain a log file named "`last-actvity.txt`" containing the date/timestamp, in fully-qualified W3C form [DateTime], of the last instance of various activities performed on the objects in the CAN and a unique identifier of the activity's process (such as a process number or thread identifier), for example:

```
lastAddVersion: 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 "`lock.txt`" that conforms to the syntax, semantics, and normative obligations defined in by the LockIt specification.

The sub-directory MAY contain a file named in the form "`log-<year><month><day>.txt`", that holds log information for the CAN 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.

The administrative directory MAY contain a file named "`summary-stats.txt`" holding summary statistics about the CAN expressed as ANVL name/value pairs, for example:

```
numObjects: 18302
numVersions: 27551
numFiles: 405833
totalSize: 730415172
```

The "`numObjects`", "`numVersions`", and "`numFiles`" properties indicate the number of objects, versions, and files, respectively, managed in the CAN.

The "t`otalSize`" property indicates the total size of the CAN, in bytes.

3.5 Object store directory (store/)

The OPTIONAL object store directory "`store`" is the structural root of the file system hierarchy in which digital objects are managed. The local structure of the "`store`" directory and its descendent sub-directories is subject to other specifications, such as Pairtree [Pairtee] and Dflat [Dflat].

```
store/
```

The particular schemes used to define the branches and leaves of the CAN's object store MUST be declared in the signature file.

4 Implementation

The CAN specification was developed with the intention that it could be implemented on top of any file system that supports hierarchical directory structures and arbitrary directory and file names, such as POSIX [POSIX].

CAN is based on a number of conceptual entities defined in terms of their state properties and behaviors that can manipulate and retrieve that state.

- CAN.
- Object.
- Version.
- File.
- Local-to-primary identifier map.

NOTE Entity definitions are based on those established by the UC3 Merritt Storage Service [Storage].

4.1 CAN Entities

A CAN is based on the following conceptual entities, each defined in terms of its specific state properties.

4.1.1 CAN

A CAN is a storage *node*, a persistent data store for digital objects. The CAN state properties MUST minimally include:

- CAN name. [can:name
- CAN identifier, assigned to be locally unique among all CANs known by a single instance of the storage service. [can:identifier]
- CAN description. [can:description]
- CAN implementation version. [can:nodeVersion]
- Total number of objects. [can:numObjects]
- Total number of versions. [can:numVersions]
- Total number of files, when fully instantiated. [can:numFiles]
- Total size (in octets), when fully instantiated. [can:totalSize]
- Total number of actual files. [can:numActualFiles]
- Total size (in octets) of actual files. [can:totalActualSize]
- Creation date/timestamp. [can:created]
- Modification date/timestamp. [can:lastModified]
- Last add version date/timestamp. [can:lastAddVersion]
- Underlying storage technology: `magnetic-disk`, `magnetic-tape`, `optical-disk`, `solid-state`. [can:mediaType]

- Underlying storage access modality: `on-line`, `near-line`, `off-line`. [can:accessMode]
- Pre-read verification status: `true` or `false`. [can:verifyOnRead]
- Post-write verification status: `true` or `face`. [can:verifyOnWrite]
- Node specification and version. [can:nodeScheme]
- Base URI for the node method invocations. [can:baseURI]
- Customer support URI. [can:supportURI]

Additional state properties MAY be defined and managed.

A CAN encapsulates an arbitrary number of *objects*.

4.1.2 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]
    - 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 CAN has a weak definition of digital objects, especially in relation to those used byother common repository services. This is by explicit design. The CAN 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*.

4.1.3 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]
  - 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*.

4.1.4 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.

4.1.5 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]

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

4.2 CAN Methods

CAN defines the following methods.

4.2.1 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: can.cdlib.org
UA: Accept: response/form
```

```
UA:

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

- Command line API

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

4.2.2 Get Node State

METHOD *Get-node-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	Node state.
SIDE EFFECTS	—		
ERRORS	400 Badly-formed request.		
	401 Unauthorized user agent.		
	415 Unsupported response form.		
	503 Service unavailable.		
	500 Service error.		

UC3Merritt

- RESTful API

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

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

- Command line API

```
% can getNodeState [-t form] [-o file]
```

UC3Merritt

4.2.3 Get Object State

METHOD *Get-object-state*			[*idempotent*, *safe*]
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 Object not found.		
	415 Unsupported response form.		
	503 Service unavailable.		
	500 Service error.		

- RESTful API

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

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

- Command line API

```
% can getObjectState object [-t form] [-o file]
```

UC3Merritt

4.2.4 Get Version State

METHOD *Get-version-state*			[*idempotent*, *safe*]
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 Object not found.		
	404 Version not found.		
	415 Unsupported response form.		
	503 Service unavailable.		
	500 Service error.		

- RESTful API

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

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

- Command line API

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

4.2.5 Get File State

METHOD *Get-file-state*			[*idempotent*, *safe*]
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 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/object/version/file[?t=form] HTTP/1.x
UA: Host: can.cdlib.org
UA: Accept: response/form
UA:

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

- Command line API

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

UC3Merritt

4.2.6 Get Object

METHOD *Get-object*			[*idempotent*, *unsafe*]
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 Object not found.		
	415 Unsupported response form.		
	501 Unsupported response mode.		
	503 Service unavailable.		
	500 Service error.		

- RESTful API

```
UA: GET /content/object[?[r=mode][[&]X]] HTTP/1.x
UA: Host: can.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

```
% can getObject bject [-t form] [-r mode] [-X] [-o file]
```

4.2.7 Get Version

METHOD *Get-version*			[*idempotent*, *unsafe*]
Object	Identifier	Mandatory	Object primary identifier.
Version	Number	Mandatory	Version number. The number "0" indicates the current version.
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
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 Object not found.		
	404 Version not found.		
	415 Unsupported response form.		
	501 Unsupported response mode.		
	503 Service unavailable.		
	500 Service error.		

- RESTful API

```
UA: GET /content/object/version[?r=mode] HTTP/1.x
UA: Host: can.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

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

4.2.8 Get File

METHOD *Get-file*			[*idempotent*, *unsafe*]
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 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/object[/version]/file[?[r=mode][[&]f]] HTTP/1.x
UA: Host: can.cdlib.org
UA:

OS: HTTP/1.x 200 OK
OS: Content-type: mime/type | text/checkm
OS:
OS: content | manifest
```

- Command line API

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

UC3Merritt

4.2.9 Add Version

METHOD *Add-version*			[*non-idempotent*, *unsafe*]
Object	Identifier	Mandatory	Object primary identifier.
RequestMode	Enum	Optional	Request mode: by-reference (default) or by-value.
URI	URI	Mandatory	If request mode is by-reference, URI of manifest containing network accessible references to version files.
Manifest	Checkm		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		Hexadecimal representation of the manifest message digest value.
LocalContext	Identifier	Optional	Object local identifier context.
LocalIdentifier	Identifier		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. Note that *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*. A persistent mapping is maintained between the local identifier and context, if supplied, and the object primary identifier.		
ERRORS	400 Badly-formed request.		
	401 Unauthorized user agent.		
	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.		

** Multiple local identifiers can 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/object[?t=form] HTTP/1.x
```

```
UA: Host: can.cdlib.org
UA: Accept: response/form
UA: Content-type: application/x-www-form-urlencoded | text/checkm
UA: Content-length: size
UA:
UA: manifest-uri=uri[&manifest-size=size][&digest-type=type&
    digest-value=value][&local-context=context&local-identifier=identi
    fier[; identifier[;...]]] |
    manifest

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

- Command line API

```
% can addVersion object uri | 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/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 will be ignored if provided.
- The first five entries in the manifest SHOULD be structured comments specifying the Checkm conformance level, profile identifier, 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/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 ]
```

4.2.10 Delete Object

METHOD *Delete-object*			[*idempotent*, *unsafe*]
Object	Identifier	Mandatory	Object identifier.

UC3Merritt

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 Object not found.		
	415 Unsupported response form.		
	503 Service unavailable.		
	500 Service error.		

- RESTful API

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

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

- Command line API

```
% can deleteObject object [-t form] [-o file]
```

UC3Merritt

4.2.11 Delete Version

METHOD *Delete-version*			[*idempotent*, *unsafe*]
Object	Identifier	Mandatory	Object 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 Object not found.		
	404 Version not found.		
	415 Unsupported response form.		
	503 Service not available.		
	500 Service error.		

- RESTful API

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

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

- Command line API

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

4.2.12 Get Primary Identifier

METHOD Get-primary-identifier				[*idempotent*,*safe*]
	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		—		

UC3Merritt

ERRORS	400	Badly-formed request.
	401	Unauthorized user agent.
	503	Service unavailable.
	500	Service error

- RESTful API

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

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

- Command line API

```
% node getPrimaryIdentifier context identifier
```

NOTE As an optimization for efficient performance, a CAN SHOULD define a persistent hashtable, or some similar mechanism, to hold the comprehensive mapping from local identifiers and contexts to associated primary identifiers.

5 Security Considerations

CAN poses no direct risk to computers or networks. As a file system convention, CAN is capable of holding files that might contain malicious executable content, but it is no more vulnerable in this regard than any file system.

Appendix A: Complete CAN Conventions

The overall file system structure of a CAN is:

```
<can_home>/
          [ 0=can_<version> ]
          [ admin/ ]
          [ can-info.txt ]
          [ 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 ]]
        [ summary-stats.txt ] ]
      store/
```

The production rule for the CAN Namaste signature file "0=can_<*version*>" is:

```
<cansig>      = "CAN/" <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*(<activities> 1*WSP <date-time> EOL)
<activities>  = <add> / <object> / <version> / <fixity>
<add>         = "lastAddVersion:"
<object>      = "lastDeleteObject:"
<version>     = "lastDeleteVersion:"
<fixity>      = "lastFixity:" 1*WSP <date-time>
<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 summary statistics file "summary-stats.txt" are:

```
<stats>       = 1*(<stat> EOL)
<stat>        = <objects> / <versions> / <files> / <size>
<objects>     = "numObjects:" 1*WSP NONNEG
<versions>    = "numVersions:" 1*WSP NONNEG
<files>       = "numFiles:" 1*WSP NONNEG
<size>        = "totalSize:" 1*WSP NONNEG
```

The production rules for the CAN global properties file "`can-info.txt`" are:

```
<properties>  = 1*(<property> EOL )
<property>    = <canname> / <identifier> / <description> / <node> /
                <branch> / <leaf> / <media> / <connect> / <mode> /
                <protocol> / <volume> / <provider> / <read> / <write>
/ <base> / <support>
<canname>     = "name:" 1*WSP 1*VCHAR
<identifier>  = "identifier:" 1*WSP 1*VCHAR
<description> = "description:" 1*WSP 1*VCHAR
<node>        = "nodeScheme:" 1*WSP <scheme>
<branch>      = "branchScheme:" 1*WSP <scheme>
<leaf>        = "leafScheme:" 1*WSP <scheme>
<scheme>      = <name> "/" <version> [ "/" <version> ]
<name>        = 1*VCHAR
<media>       = "mediaType:" 1*WSP ("magnetic-disk" / "magnetic-tape" /
                                    "optical-disk" / "solid-state") /
                                   "content-addressable"
<connect>     = "mediaConnectivity" 1*WSP ("direct" / "nas" / "san" /
                                           "cloud")
<mode>        = "accessMode" 1*WSP ("on-line" / "near-line" /
                                    "off-line")
<protocol>    = "accessProtocol:" 1*WSP ("zfs" / "s3")
<volume>      = "logicalVolume:" 1*WSP 1*VCHAR
<provider>    = "externalprovider:" 1*WSP 1*VCHAR
<read>        = "verifyOnRead:" 1*WSP ("true" / "false")
<write>       = "verifyOnWrite:" 1*WSP ("true" / "false")
<base>        = "baseURI:" 1*WSP <rfc-3986-compliant-uri>
<support>     = "supportURI:" 1*WSP <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] J. Kunze, B. Kahle, J. Masanes, and G. Mohr, *A Name-Value Language (ANVL)*, Internet draft, February 14, 2005 <http://www.ietf.org/internet-draft/draft-kunze-anvl-01.txt>.

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

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

UC3Merritt

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

[Pairtree] J. Kunze, M. Haye, E. Hetzner, M. Reyes, and C. Snavely, *Pairtrees for Object Storage (V0.1)*, Internet draft, November 25, 2008 <http://www.ietf.org/internet-drafts/draft-kunze-pairtree-01.txt>.

[POSIX] ISO/IEC 9945:2003/IEEE Std 1003.1, *Information technology – Portable operating system interface (POSIX) – Part 2: System interface*.

[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>.

[Storage] UC3, *Merritt Storage Service*, 2010.

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