



CDL/UC Curation Center

CAN: A Simple File System-Based Object Store

Rev. 0.8 – 2009-09-03

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 discursive text or 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_0.8 ]                             # CAN Nameste signature
  [ admin/ ]                                 # administrative declarations
  [ can-info.txt ]                           # CAN properties file
  [ lock.txt ]                               # write lock
  [ log/ ]                                   # log directory
  store/                                     # object store
```

A CAN home directory MAY contain a file named “0=can_<version>” that is its Namaste [Nameste] 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.

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.

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

The RECOMMENDED Namaste signature file “0=cab_<version>” self-identifies a directory as a CAN home directory. The “<version>” portion of the file name asserts the version of the Dflat specification to which the directory conforms. If present, the contents of the file MUST exactly duplicate the file name.

```
0=can_<version>
```

Note that this value is duplicative of the “Node-scheme” 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/  
  [ lock.txt ]  
  [ summary-stats.txt ]
```

The administrative directory MAY contain a file named “lock.txt” that conforms to the syntax, semantics, and normative obligations defined in Section § 3.5.

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:

```
[ Object-count: 18302 ]  
[ Version-count: 27551 ]  
[ File-count: 405833 ]  
[ Total-size: 730415172 ]
```

The “Object-count”, “Version-count”, and “File-count” properties indicate the number of objects, versions, and files, respectively, managed in the CAN.

The “Total-size” property indicates the total size of all versioned files, in bytes.

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: can01.cdlib.org ]  
[ Node-scheme: CAN/0.6 ]  
[ Branch-scheme: Pairtree/0.1 ]  
[ Leaf-scheme: Dflat/0.16 ]  
[ Class-scheme: CLOP/0.3 ]  
[ Media-type: magnetic-disk ]  
[ Access-type: on-line ]  
[ Verify-on-read: true ]  
[ Verify-on-write: true ]  
[ Access-uri: http://can01.cdlib.org/ ]
```

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 “Node-scheme” property indicates the version of the CAN specification to which the CAN conforms. Note that this value is duplicative to the information provided by the OPTIONAL Namaste tag. If both are present, the properties file is considered authoritative.

The “Branch-scheme” property indicates the specification name and version of the convention used



for the branches of the object store hierarchy.

The “Leaf-scheme” property indicates the specification name and version of the convention used for the leaves of the object store hierarchy.

The “Class-scheme” property indicates the specification name and version of the convention used for managing the administrative properties of managed objects.

The “Media-type” property indicates the storage technology underlying the CAN: magnetic disk, magnetic tape, optical disk, or solid-state.

The “Access-type” property indicates level of availability of stored data: on-line, near-line, or off-line.

The “Access-uri” property indicates the base URI used for user access to the CAN.

3.4 Lock file (lock.txt)

The OPTIONAL lock file “lock.txt” indicates that the CAN is being processed in a manner that may temporarily put it in an unknown, inconsistent, or incomplete state. The file holds the date/timestamp, in fully-qualified W3C form [DateTime], at which the lock was established and an identifier of the process holding the lock (such as a process or thread identifier), for example:

```
Lock: 2009-02-14T11:04:23+0800 50124
```

Any process intending to manipulate a CAN in a manner that affects the state of its home directory MUST first obtain a write lock on that directory. Any process holding a write lock on a CAN home directory SHOULD release it at the earliest safe moment. All processes reading a CAN SHOULD look for the presence of the lock file before attempting the operation; if present, the process SHOULD continue only under an assumption that the data read may be incomplete or inconsistent.

3.5 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-access.txt ]  
  [ last-fixity.txt ]  
  [ lock.txt ]
```

The sub-directory MAY contain a file named “last-access.txt” containing the date/timestamp of the last end-user access of an object in the CAN in fully-qualified W3C form and an identifier of the accessing process, for example:

```
Last-access: 2009-01-29T05:33:24+0800 66212
```

The directory SHOULD contain a file named “last-fixity.txt” containing the date/timestamp in fully-qualified W3C form of the last complete fixity verification of the objects in the CAN and an



identifier of the fixity process, for example:

```
Last-fixity: 2008-12-22T23:00:10+0800 18002
```

The sub-directory MAY contain a file named “lock.txt” that conforms to the syntax, semantics, and normative obligations defined in Section § 3.5.

3.6 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

It SHOULD be possible to implement CAN on top of any file system that supports hierarchical directory structures and arbitrary directory and file names.

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/
    [ lock.txt ]
    [ summary-stats.txt ] ]
  [ can-info.txt ]
  [ lock.txt ]
  [ log/
    [ last-access.txt ]
    [ last-fixity.txt ]
    [ lock.txt ] ]
  store/
```

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

```
<cansig>      = “0=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

```

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 rules for the summary statistics file “summary-stats.txt” are:

```

<stats>       = 1*<stat>
<stat>        = (<objects> / <versions> / <files> / <size>) EOL
<objects>     = “Object-count:” 1*WSP NONNEG
<versions>    = “Version-count:” 1*WSP NONNEG
<files>       = “File-count:” 1*WSP NONNEG
<size>        = “Total-size:” 1*WSP NONNEG

```

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

```

<properties>  = 1*<property>
<property>    = (<canname> / <node> / <branch> / <leaf> / <class> /
<media> / <access> / <uri>) EOL
<canname>     = “Name:” 1*WSP 1*VCHAR
<node>        = “Node-scheme:” 1*WSP <scheme>
<branch>      = “Branch-scheme:” 1*WSP <scheme>
<leaf>        = “Leaf-scheme:” 1*WSP <scheme>
<class>       = “Class-scheme:” 1*WSP <scheme>
<scheme>      = <name> “/” <version>
<name>        = 1*VCHAR
<media>       = “Media-type:” 1*WSP (“magnetic-disk” / “magnetic-tape” /
“optical-disk” / “solid-state”)
<access>      = “Access-type:” 1*WSP (“on-line” / “near-line” /
“off-line”)
<uri>         = <<rfc-3986-compliant-uri>

```

The production rule for the lock file “lock.txt” is:

```

<lock>        = “Lock:” 1*WSP <date-time> 1*WSP <process> EOL
<date-time>   = <date> “T” <time>
<date>        = <year> “-“ <month> “-“ <day>
<year>        = 4DIGIT
<month>       = 2DIGIT ; normal constraints apply, 01 through 12
<day>         = 2DIGIT ; normal constraints apply, 01 through 31

```



```

<time>           = <hour> ":" <minute> ":" <second> "+" <zzzz>
<hour>           = 2DIGIT ; normal constraints apply, 00 through 23
<minute>        = 2DIGIT ; normal constraints apply, 00-59
<second>        = 2DIGIT ; normal constraints apply, 00-59
<zzzz>          = 4DIGIT ; normal constraints apply, 0000 through 2300
<process>       = 1*VCHAR

```

The production rule for the access log file "last-access.txt" is:

```

<last-access> = "Last-access:" 1*WSP <date-time> 1*WSP <process> EOL
<date-time>  = <date> "T" <time>
<date>       = <year> "-" <month> "-" <day>
<year>       = 4DIGIT
<month>      = 2DIGIT ; normal constraints apply, 01 through 12
<day>        = 2DIGIT ; normal constraints apply, 01 through 31
<time>       = <hour> ":" <minute> ":" <second> "+" <zzzz>
<hour>       = 2DIGIT ; normal constraints apply, 00 through 23
<minute>     = 2DIGIT ; normal constraints apply, 00-59
<second>     = 2DIGIT ; normal constraints apply, 00-59
<zzzz>      = 4DIGIT ; normal constraints apply, 0000 through 2300
<process>    = 1*VCHAR

```

The production rule for the fixity log file "last-fixity.txt" is:

```

<last-access> = "Last-access:" 1*WSP <date-time> 1*WSP <process> EOL

```

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

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>>.
- [CLOP] *CLOP: A Class-based System for Managing Object Properties*, rev. 0.3, April 3, 2009.
- [Dflat] *DFlat: A Simple File System Convention for Object Storage*, rev. 0.10, April 4, 2009.
- [Pairtree] J. Kunze, M. Haye, E. Hetzner, M. Reyes, and C. Snively, *Pairtrees for Object Storage (V0.1)*, Internet draft, November 25, 2008 <<http://www.ietf.org/internet-drafts/draft-kunze-pairtree-01.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>>.