Content Access Node (CAN)

Rev. 0.1 – 2009-02-27

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 object store features, if present, in well-known locations within a single directory hierarchy that comprises the entire object store.

CAN is was designed to interoperate cleanly with other independent, but related specifications such as CAN (Content Access Node) [CAN] and Pairtree [Pairtree]. All three of these specifications start from the assumption that a file system is a sufficient storage abstraction for the effective management of digital objects. Assuming that these objects are stored 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 D-flat 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.

All uses of the term “directory” in this document MAY be interchanged with “folder” without loss of intended meaning. Similarly, all uses of the solidus (“/”) as a directory path separator MAY be interchanged with a reverse solidus (“\”). In other words, this specification SHOULD be applicable in both Unix/Linux and Windows/DOS contexts.

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.

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 know as the CAN *home directory*. The CAN specification reserves a few key file system names within the home directory and its subdirectories, but it does not dictate how the home directory itself is names, as that name is not visible from inside the CAN.

A CAN looks like this:

```
<can-home>/
            can-info.txt
```

The CAN home directory MUST contain a file named "can-info.txt" that functions as the indicative CAN signature and defines the global properties of the CAN itself.

3.1 CAN Signature File (can-info.txt)

The REQUIRED "`can-info.txt`" file self-identifies a directory hierarchy as a CAN and defines the properties of that CAN itself, as opposed to the objects stored in it. These properties are expressed in terms of ANVL [ANVL] name/value pairs, for example:

```
CAN-version: 0.1
CAN-name: preservation01
Root-pathname: /repository/can-root
Branch-scheme: Pairtree, v. 0.1; properly-encapsulated
Leaf-scheme: D-flat, v. 0.7
```

The "`CAN-version`" property MUST indicate the version of the CAN specification to which the CAN conforms. The "`CAN-name`" property MUST specify the name of the CAN. The "`Root-pathname`" property MUST specify the absolute pathname of the root of the object store hierarchy. The "`Branch-scheme`" property MUST specify the organizational convention used for the branches of the object store hierarchy. The "`Leaf-scheme`" property MUST specify the organizational convention used for the leaves of the object store hierarchy.

The store consisting of object replication classes, a workspace (for indexes, catalogs, etc), and a [Pairtree] to hold the collection of objects, each of which is laid out as a [D-flat]. Each instance of a CAN has a configuration file, "`can-opener.txt`", which defines these values:

```
rclass_A: <digests_at_URL>               # remote held checksums
          <verification_frequency>       # how often to recompute
          <optional_first_replica_URL>   # if replicating, copy 1
          <optional_second_replica_URL> # if replicating, copy 2
           ...
rclass_B: .....                          # other classes as needed
object_dirname_scheme: <scheme>          # eg, "tail_9"
workspace: <full_path/URL>               # indexes, catalogs, etc.
... <hand_waving> ...
```

Each object in a CAN declares its replication (and fixity) class via its D-flat `admin/classes.txt` file. If only fixity checking (without replication) is desired, the "rclass" is defined with only the location of the digests location and frequency. If no "rclass" is defined for an object, the CAN will provide neither fixity nor replication for it.

The `object_dirname_scheme` specifies how the d-flat home directory name will be formed or derived from the object identifier. If no scheme is given, the home directory will simply be the entire identifier (after pairpath cleaning). As this can sometimes result in an uncomfortably long home directory name, such as

```
ark:/13030/xt12t384r9q43wv → ark+=13030=xt12t384r9q43wv
```

a scheme such as "tail_9" will take the "tail" of the identifier (the last lexical component, and if that is longer than 9 characters, only the last 9 characters). A scheme name of the form "tail_N" will be

understood for any value of N.

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, D-flat is capable of holding files that might contain malicious executable content, but it is no vulnerable in this regard than any file system.

Appendix A: Complete CAN Conventions

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>.

[D-flat] *D-Flat: A Simple File System Convention for Object Storage*.

[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>.