Network Working Group Internet-Draft Expires: May 29, 2009 J. Kunze M. Haye E. Hetzner M. Reyes California Digital Library C. Snavely University of Michigan Library IT Core Services November 25, 2008

Pairtrees for Object Storage (V0.1) http://www.ietf.org/internet-drafts/draft-kunze-pairtree-01.txt

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with <u>Section 6 of BCP 79</u>.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/lid-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on May 29, 2009.

Copyright Notice

Copyright (C) The IETF Trust (2008).

Internet-Draft

Pairtree

Abstract

This document specifies Pairtree, a filesystem hierarchy for holding objects that are located within that hierarchy by mapping identifier strings to object directory (or folder) paths two characters at a time. If an object directory (folder) holds all the files, and nothing but the files, that comprise the object, a "pairtree" can be imported by a system that knows nothing about the nature or structure of the objects but can still deliver any object's files by requested identifier. The mapping is reversible, so the importing system can also walk the pairtree and reliably enumerate all the contained object identifiers. To the extent that object dependencies are stored inside the pairtree (e.g., fast indexes stored outside contain only derivative data), simple or complex collections built on top of pairtrees can recover from index failures and reconstruct a collection view simply by walking the trees. Pairtrees have the advantage that many object operations, including backup and restore, can be performed with native operating system tools.

Kunze, et al.Expires May 29, 2009[Page 2]

1. The basic pairtree algorithm

The pairtree algorithm maps an arbitrary UTF-8 [<u>RFC3629</u>] encoded identifier string into a filesystem directory path based on successive pairs of characters, and also defines the reverse mapping (from pathname to identifier).

In this document the word "directory" is used interchangeably with the word "folder" and all examples conform to Unix-based filesystem conventions which should tranlate easily to Windows conventions after substituting the path separator ('\' instead of '/'). Pairtree places no limitations on file and path lengths, so implementors thinking about maximal interoperation may wish to consider the issues listed in the Interoperability section of this document.

The mapping from identifier string to path has two parts. First, the string is cleaned by converting characters that would be illegal or especially problemmatic in Unix or Windows filesystems. The cleaned string is then split into pairs of characters, each of which becomes a directory name in a filesystem path: successive pairs map to successive path components until there are no characters left, with the last component being either a 1- or 2-character directory name. The resulting path is known as a _pairpath_, or _ppath_.

abcd -> ab/cd/ abcdefg -> ab/cd/ef/g/ 12-986xy4 -> 12/-9/86/xy/4/

Armed with specific knowledge of a given namespace's identifier distribution, one might achieve more balanced or efficient trees by mapping to paths from character groupings other than successive pairs. Pairtree assumes that this sort of optimization, however, being tailored to individual and transient namespace conditions, is often less important than having a single generalized and shareable mapping. It uses pairs of characters to achieve hierarchies that exhibit a reasonable balance of path length and fanout (number of probable entries in any component directory).

[Page 3]

2. Pairpath termination and object encapsulation

A ppath (pairpath) terminates when it reaches an object. A little jargon helps explain this. A _shorty_ is a 1- or 2-character directory name, or any file or directory name that begins with "pairtree" (these are reserved for future use). A ppath consists of a sequence of "shorties" ending in a non-shorty, such as a 3-character directory name or the 2-character file name "xy". The pairtree below contains two objects with identifiers "abcd" and "abcde".

```
ab/
I
```

```
\--- cd/
    |--- foo/
            README.txt
    thumbnail.gif
    |--- master_images/
    | ...
        . . .
    \--- gh/
    \--- e∕
        \--- bar/
                metadata
             54321.wav
             index.html
```

An object is reached when a non-shorty is detected. An object is _properly encapsulated_ if it is entirely contained in a non-shorty directory that is the immediate child of a shorty directory, in other words, if the 1- or 2-char directory name ending the object's ppath contains exactly one non-shorty directory that holds all the object's descendants. The two objects "abcd" and "abcde" above are properly encapsulated. Any shorty directory found at the same level as the non-shorty extends the pairtree. So while the "foo/" directory above does not subsume "e/" at the same level, by encapsulation, it does subsume the "gh/" underneath it (i.e., "gh/" is invisible to the pairtree algorithm, at least on a first pass).

Practice will vary according to local custom as to how to name the encapsulating object directory beneath that last shorty. Its name is completely independent of the object identifier. For example, every object directory in a pairtree could have the uniform name "thingy".

[Page 4]

It is common for the directory name to be a terminal substring of the object identifier, as in:

id: 13030_45xqv_793842495
ppath: 13/03/0_/45/xq/v_/79/38/42/49/5/793842495

All objects should be properly encapsulated. If an object is detected that is _improperly encapsulated_, that is, when a ppath ends with a shorty directory that contains more than one non-shorty, the detecting system should take corrective action. In this situation, also known as a "split end", all those non-shorties (directories and files) are considered to belong to one object (not properly encapsulated) identified by the containing ppath. Excluding shorties from the object permits one identifier to be a substring of another (e.g., "abcd" and "abcde" can co-exist in a pairtree), and defining ppath termination in this way prevents "hidden riders", or data residing in a pairtree that is not contained or accounted for in any object. Here is an example of an improperly encapsulated object named "bent".

If a "split end" is encountered, an importing system is encouraged to normalize it by creating a single object directory called "obj" and pushing the non-shorties in question underneath it, as in:

```
be/
|
\--- nt/
|
|--- obj/ [ split end repaired with "obj" directory ]
| | README.txt
| | report.pdf
|
\--- ef/
| ...
```

[Page 5]

<u>3</u>. Identifier string cleaning

Prior to splitting into character pairs, identifier strings are cleaned in two separate steps. One step would be simpler, but pairtree is designed so that commonly used characters in reasonably opaque identifiers (e.g., not containing natural language words, phrases, or hints) result in reasonably short and familiar-looking paths. For completeness, the pairtree algorithm specifies what to do with all possible UTF-8 characters, and relies for this on a kind of URL hex-encoding. To avoid conflict with URLs, pairtree hex-encoding is introduced with the '^' character instead of '%'.

First, the identifier string is cleaned of characters that are expected to occur rarely in object identifiers but that would cause certain known problems for file systems. In this step, every UTF-8 octet outside the range of visible ASCII (94 characters with hexadecimal codes 21-7e) [ASCII], as well as the following visible ASCII characters,

	hex 22	< hex 3c	\	hex 5c
*	hex 2a	= hex 3d	\wedge	hex 5e
+	hex 2b	> hex 3e		hex 7c
,	hex 2c	? hex 3f		

must be converted to their corresponding 3-character hexadecimal encoding, ^hh, where ^ is a circumflex and hh is two hex digits. For example, ' ' (space) is converted to ^20 and '*' to ^2a.

In the second step, the following single-character to singlecharacter conversions must be done.

/ -> = : -> + . -> ,

These are characters that occur quite commonly in opaque identifiers but present special problems for filesystems. This step avoids requiring them to be hex encoded (hence expanded to three characters), which keeps the typical ppath reasonably short. Here are examples of identifier strings after cleaning and after ppath mapping.

[Page 6]

- id: ark:/13030/xt12t3
 - -> ark+=13030=xt12t3
- -> ar/k+/=1/30/30/=x/t1/2t/3/
- id: <u>http://n2t.info/urn:nbn:se:kb:repos-1</u>
- -> http+==n2t,info=urn+nbn+se+kb+repos-1
- -> ht/tp/+=/=n/2t/,i/nf/o=/ur/n+/nb/n+/se/+k/b+/re/po/s-/1/
- id: what-the-*@?#!^!?
- -> what-the-^2a@^3f#!^5e!^3f
- -> wh/at/-t/he/-^/2a/@^/3f/#!/^5/e!/^3/f/

After this character cleaning procedure, directory names resulting from splitting the string into character pairs will be legal and not terribly inconvenient for mainstream Unix and Windows systems, for their command interpreters, and as web-exposed URL paths.

Kunze, et al.Expires May 29, 2009[Page 7]

Pairtree

<u>4</u>. Pairpath initiation

The top of a pairtree hierarchy is signaled by the presence of a directory called "pairtree_root". There may be other filenames beginning with "pairtree" accompanying it, as in the example below. Lines of file content, when shown, appear in parentheses beneath the file name.

```
current_directory/
```

```
pairtree version0 1 [which version of pairtree]
    ( This directory conforms to Pairtree Version 0.1. Updated spec: )
( <u>http://www.cdlib.org/inside/diglib/pairtree/pairtreespec.html</u> )
pairtree_prefix
    ( http://n2t.info/ark:/13030/xt2
                                                                    )
Τ
\--- pairtree_root/
    |--- aa/
        |--- cd/
     1
             |--- foo/
         1
              README.txt
         thumbnail.gif
              I
         . . .
        |--- ab/ ...
         |--- af/ ...
     |--- ag/ ...
         . . .
    |--- ab/ ...
     . . .
    \--- zz/ ...
         | . . .
```

The "pairtree_prefix" contains a string that should be prepended to every identifier inferred from the pairtree rooted at "pairtree_root". This may be used to reduce path lengths when every identifier in a given pairtree shares the same initial substring. In the example above, the pairpath "/aa/cd/" would thus correspond to the identifier "http://n2t.info/ark:/13030/xt2aacd".

[Page 8]

<u>5</u>. Pairtree benefits

Pairtree can be used with any object identifier, but its real strength comes when two main assumptions are also in effect. The first assumption is that every component on which an object depends will be stored in the filesystem. Increasingly, digital library systems recognize that the risk of scattering components among databases and files can be reduced when all primary data is kept in non-volatile storage that can be backed up and manipulated using core, ubiquitous operating system tools. While database indexes are important for supporting fast or complex query execution, this precondition merely requires that those indexes hold secondary copies of object components (e.g., metadata).

The second assumption is that all the components of an object, and only the components of that object, are stored in an object's directory. Thus an object's directory contains no components belonging to another object. Of course complex objects will still contain other objects, and possibly other pairtrees, but such object containment is not visible to the pairtree algorithm except with a recursive pass.

With these two pre-conditions met, a pairtree can be imported by a system that knows nothing about the nature or structure of the objects but can still deliver any object's files by requested identifier. The mapping is reversible, so the importing system can also walk the pairtree and reliably enumerate all the contained object identifiers. To the extent that object dependencies are stored inside the pairtree, simple or complex collections built on top of pairtrees can recover from index failures and reconstruct a collection catalog simply by walking the trees. Finally, pairtrees have the advantage that many object operations, including backup and restore, can be performed with native operating system tools.

Kunze, et al.Expires May 29, 2009[Page 9]

6. Interoperability: Windows and Unix File Naming

Besides the fundamental difference between path separators (' $\$ and '/'), generally, Windows filesystems have more limitations than Unix filesystems. Windows path names have a maximum of 255 characters, and none of these characters may be used in a path component:

< > : " / | ? *

Windows also reserves the following names: CON, PRN, AUX, NUL, COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, and LPT9. See [MSFNAM] for more information.

Kunze, et al.Expires May 29, 2009[Page 10]

7. Security Considerations

Pairtree poses no direct risk to computers and networks. As a filesystem format, pairtree is capable of holding files that might contain malicious executable content, but it is no more vulnerable in this regard than formats such as TAR and ZIP.

Pairtree

<u>Appendix A</u>. Sample Implementation

There is a [PAIRTREE] Perl module at CPAN the implements two mappings. The routine, id2ppath, maps an identifier to a pairpath, and another routine, ppath2id, performs the inverse mapping. The usage synopsis follows.

use File::Pairtree;	#	imports	routines	into	a Perl	scri	ot
id2ppath(\$id);	#	returns	pairpath	corre	espondi	ng to	\$id
<pre>ppath2id(\$path);</pre>	#	returns	id corres	spondi	ng to	\$path	

Pairtree

8. References

- [ASCII] "Coded Character Set -- 7-bit American Standard Code for Information Interchange, ANSI X3.4", 1986.

[PAIRTREE]

Kunze, "File::Pairtree Perl Module", November 2008, <http: //search.cpan.org/~jak/Pairtree-0.2/lib/File/Pairtree.pm>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, <u>RFC 3629</u>, November 2003.

Kunze, et al.Expires May 29, 2009[Page 13]

Pairtree

Authors' Addresses John A. Kunze California Digital Library 415 20th St, 4th Floor Oakland, CA 94612 US Fax: +1 510-893-5212 Email: jak@ucop.edu Martin Haye California Digital Library 415 20th St, 4th Floor Oakland, CA 94612 US Fax: +1 503-234-3581 Email: martin.haye@ucop.edu Erik Hetzner California Digital Library 415 20th St, 4th Floor Oakland, CA 94612 US Fax: +1 503-234-3581 Email: erik.hetzner@ucop.edu Mark Reyes California Digital Library 415 20th St, 4th Floor Oakland, CA 94612 US Fax: +1 503-234-3581 Email: mark.reyes@ucop.edu

Kunze, et al.Expires May 29, 2009[Page 14]

Cory Snavely University of Michigan Library IT Core Services 920 N University Ave, 300D Hatcher Library N Ann Arbor, MI 48109 US

Fax: +1 734-647-6897 Email: csnavely@umich.edu Full Copyright Statement

Copyright (C) The IETF Trust (2008).

This document is subject to the rights, licenses and restrictions contained in $\frac{BCP}{78}$, and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in <u>BCP 78</u> and <u>BCP 79</u>.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).