Network Working Group
Internet-Draft
Expires: August 23, 2008

A. Phillips, Ed.
Yahoo! Inc.

February 20, 2008

The record-jar Format
draft-phillips-record-jar-02

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 Section 6 of BCP 79.

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/1id-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 August 23, 2008.

Copyright Notice

Abstract

The record-jar format provides a method of storing multiple records with a variable repertoire of fields in a text format. This document provides a description of the format. Comments are solicited and should be addressed to the mailing list 'record-jar@yahoogroups.com' and/or the author.

Table of Contents

1. Introduction

The record-jar format was originally described by The Art of Unix Programming [AOUP]. This format is useful for storing information in a human-readable text form, while making the data available for machine processing. It is a flexible format, since it provides for an arbitrary range of fields in any given record and can be used to store data with variable length and content.

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 [RFC2119].

2. Format and Grammar

The record-jar format is described by the following ABNF ([RFC4234]):

```
record-jar   = [encodingSig] [separator] *record
record       = 1*field separator
field        = ( field-name field-sep field-body CRLF )
field-name   = 1*character
field-sep    = *SP ":" *SP
field-body   = *(continuation 1*character)
continuation = ["\"] [[*SP CRLF] 1*SP]
separator    = [blank-line] *("%%" [comment] CRLF)
comment      = SP *69(character)
character    = SP / ASCCHAR / UNICHAR / ESCAPE
encodingSig  = "%%encoding" field-sep
                 *(ALPHA / DIGIT / "-" / "_") CRLF
blank-line   = WSP CRLF

; ASCII characters except %x26 (&) and %x5C (\)
ASCCHAR      = %x21-25 / %x27-5B / %x5D-7E
; Unicode characters
UNICHAR      = %x80-10FFFF
ESCAPE       = "\" ("\" / "&" / "r" / "n" / "t" )
             / "&#x" 2*6HEXDIG ";"
```

record-jar ABNF

The record-jar format uses plain-text to represent data values. A record-jar document consists of a sequence of records, each of which contains one or more fields. Each record is separated from other records by at least one line beginning with the sequence "%%" (%x25.25). A record MAY contain as many or as few fields as are necessary to convey the necessary data. Empty records and blank lines are ignored.

A field is a single, logical line of characters from the Universal Character Set (Unicode) [Unicode]. Each field is comprised of three parts: the field-name, the field-separator, and the field body.

The field-name is an identifer. Field-names consist of a sequence of Unicode characters. Whitespace characters and colon (":", %x3A) are not permitted in a field-name.

An application can impose additional restrictions on field-names. For example, they might be restricted to the characters permitted in identifiers according to Unicode Standards Annex #31 (UAX#31) [UAX31]. Or they might be restricted to a sequence of letters and digits from the US-ASCII [ISO646] character repertoire.

Field-names are case sensitive. Upper and lowercase letters are often used to visually break up the name, for example using CamelCase. It is a common convention that field names use an initial capital letter, although this is not enforced.

The field separator (field-sep) is the colon character (":", %x3A). The separator MAY be surrounded on either side by any amount of horizontal whitespace (tab or space characters). The normal convention is one space on each side.

The field-body contains the data value. Logically, the field-body consists of a single line of text using any combination of characters from the Universal Character Set followed by a CRLF (newline). The carriage return, newline, and tab characters, when they occur in the data value stored in the field-body, are represented by their common backslash escapes ("\r", "\n", and "\t" respectively). See Section 2.3 for more information on escape sequences.

2.1. Folding of Field Values

Some protocols limit total line length. For example, many Internet plain-text protocols limit lines to 72 total bytes. To accommodate such limits or for readability and presentational purposes, the field-body portion of a field can be split into a multiple-line representation; this is called "folding".

Successive lines in the same field-body begin with one or more whitespace characters. When processing the record-jar format, the linear whitespace (including the newline and any preceeding spaces) is consumed by the processor and the two parts of the field-body joined to form a single, logical line. For example:

```
Eulers-Number : 2.7182818284590452353602874713526624977572470936999595749669676277240766303535475945713821785251664274274663919320030599218174135...
```

Figure 2: Example of Folding

Note that imposing a line length limit effectively limits the length of the field-name, since the field separator MUST appear on the same line with the field-name and the field-name MUST NOT be folded. Also, when imposing a line length limit, note that some encodings (including the Unicode encodings) can use a variable number of bytes per character or commonly use more than one byte per character. Characters MUST NOT be folded in the middle of a byte sequence.

It is RECOMMENDED that folding not occur between characters inside a Unicode grapheme cluster (since this will alter the display of characters in the file and might result in unintentional alteration

of the file's semantics). Information on grapheme clusters can be found in [UAX29]

In some cases, the field-body contains spaces that are important to the data. To accurately preserve whitespace in the document, an optional line-continuation character (backslash, %x5C) MAY be included to delimit and separate whitespace to be preserved from whitespace that will be removed by the processor. The line-continuation character and any whitespace that follows it (including whitespace at the beginning of the continuing field-body on the next line) MUST be consumed by the processor when reading the file. Whitespace appearing before the line-continuation MUST NOT be consumed. Use of the line continuation character makes the whitespace visible in the file.

In other cases, the field-body might contain natural language text, and, while it is readily apparent that many languages use spaces to separate words, others, such as Japanese or Thai, do not. Implementations MAY, in the absence of line continuation characters, replace the continuation sequence (the line break and surrounding whitespace) in a folded line with a single ASCII space (%x20), however, implementations SHOULD just remove the continuation sequence altogether in order to avoid causing unnatural breaks in the text.

Here are some examples:

```
SomeField : This is some running text \
 that is continued on several lines \
 and which preserves spaces between \
 the words.
%%
AnotherExample: There are three spaces   \
between 'spaces' and 'between' in this record.
%%
SwallowingExample: There are no spaces between \
       the numbers one and two in this example 1\
       2.
%%
```

Figure 3: Example of Folding with Preserved Whitespace

Note that entirely blank continuation lines are not permitted. That is, this record is illegal, since the field-body of "SomeText" would be the empty string:

```
%%
SomeText:               \
                        \
                        \
%%
```

Figure 4: Whitespace Folding Example

2.2. Comments

Comments MAY be included in the body of the record-jar document by placing them at the end of a separator line. The comment MUST be separated by at least one space from the "%%" sequence that introduces the record separator.

Multiple record separators (including comment lines) MAY appear between records. Logically this appears to result in records that contain no fields: records containing no fields MUST be ignored by a processor.

Folding of comments is not permitted; instead multiple comment lines MUST be used. Comments can not appear in the body of a record. For example:

```
%% this is a comment.
Record: goes here
%%
%% here is another sequence of comments
%% that appear on multiple lines
Record: another record
%% a final comment
%%
```

Figure 5: Comment example

Although comments are not associated with any particular record in the file, processors that preserve comments sometimes treat the comments as if they were associated with the record just following them. Reserialization of a record-jar file would thus restore the comments to their logical position in the file. In many cases, processing a record-jar file loses comment information associated with the file.

2.3. Characters, Encodings, and Escapes

By default, a file containing a record-jar archive uses the UTF-8 character encoding (see [RFC3629]). If an application, protocol, or specification permits a character encoding other than UTF-8 to be used in the file, it SHOULD also support reading the character

encoding from the encoding signature.

The encoding signature, when present, MUST be the very first line of the file. If the encoding signature is not present, an application or protocol MAY attempt to infer the character encoding using other means. Record-jar files SHOULD always include an encoding signature, even if one is not required, whenever the application, protocol, or specification permits one.

A file that uses the UTF-16 or UTF-32 encoding MAY also include a Byte Order Mark (U+FEFF) as the first sequence of two octets (in the case of UTF-16) or four octets (in the case of UTF-32) in the file, just preceeding the encoding signature.

Some applications, protocols, or specifications require that the record-jar file use some other, non-Unicode, legacy character encoding. In particular, some applications, protocols, or specifications only support the US-ASCII character set ([ISO646]).

Here is an example of the encoding signature for the UTF-8 encoding of Unicode:
%%encoding:UTF-8

Figure 6: Example of an Encoding Signature

Printable ASCII characters excepting backslash ("\") and ampersand ("&") are represented as themselves.

Non-ASCII values MAY be included in a record-jar file in several ways. For portability, the best mechanism is to use escape sequences in the field-body. Exclusive use of escape sequences results in a pure ASCII text file.

Non-ASCII characters MAY be represented using the character's Unicode value represented using the Numeric Character Reference format adapted from XML; the sequence "&#x" (%x26.23.78) is followed by the character's Unicode scalar value in hex followed directly by the semi-colon character (";", %x3B). Leading zeroes MAY be omitted. For example, the EURO SIGN is U+20AC and could be represented as "€".

Non-ASCII characters MAY also be represented as their associated octet sequence in the file's character encoding. For example, the EURO SIGN would be represented as the octet sequence %xE2.82.AC, since those three bytes encode that character in UTF-8.

The characters for carriage return, newline, and tab when considered as part of the data (and not the file format itself) are represented

by the traditional escape sequences "\r" (%x5C.72), "\n" (%x5C.6E), and "\t" (%x5C.74) respectively. The character backslash is represented by "\\" (%x5C.5C), while the ampersand character is represented by "\&" (%x5C.26). A single backslash at the end of a line indicates continuation, as discussed in Section 2.1. Otherwise a single backslash followed by some other character in the data is an error, although a record-jar processor MAY choose to interpret it as a backslash.

3. Examples

Here is the canonical example from [AOUP]:

```
Planet: Mercury
Orbital-Radius: 57,910,000 km
Diameter: 4,880 km
Mass: 3.30e23 kg
%%
Planet: Venus
Orbital-Radius: 108,200,000 km
Diameter: 12,103.6 km
Mass: 4.869e24 kg
%%
Planet: Earth
Orbital-Radius: 149,600,000 km
Diameter: 12,756.3 km
Mass: 5.972e24 kg
Moons: Luna
```

A more complete example showing more of the various features in the format is described in [RFC4646]. The data shown here is taken from the Language Subtag Registry defined that document:

```
%%
Type: language
Subtag: ia
Description: Interlingua (International Auxiliary Language \
  Association)
Added: 2005-08-16
%%
Type: language
Subtag: id
Description: Indonesian
Added: 2005-08-16
Suppress-Script: Latn
%%
Type: language
Subtag: nb
Description: Norwegian Bokm&#xE5;l
Added: 2005-08-16
Suppress-Script: Latn
%%
```

4. References

4.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

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

[RFC4234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", draft-crocker-abnf-rfc2234bis-00 (work in progress), October 2005, <ftp://ftp.rfc-editor.org/in-notes/rfc4234.txt>.

[UAX31] Davis, M., "Unicode Standard Annex #31: Identifier and Pattern Syntax", 09 2006.

[Unicode] Unicode Consortium, "The Unicode Consortium. The Unicode Standard, Version 5.0, (Boston, MA, Addison-Wesley, 2003. ISBN 0-321-49081-0)", January 2007.

4.2. Informative References

[AOUP] Raymond, E., "The Art of Unix Programming", 2003, <urn:isbn:0-13-142901-9>.

[ISO646] International Organization for Standardization, "ISO/IEC 646:1991, Information technology -- ISO 7-bit coded character set for information interchange.", 1991.

[RFC4646] Phillips, A., Ed. and M. Davis, Ed., "Tags for the Identification of Languages", September 2006, <http://www.ietf.org/rfc/rfc4646.txt>.

[UAX29] Davis, M., "Unicode Standard Annex #29: Text Boundaries", 10 2006, <UAX29>.

Appendix A. Acknowledgements

Thanks to Eris S. Raymond for his gracious permission to both reference and quote The Art of Unix Programming in this document. Without his work, this document would likely not exist.

Contributors to this document include: Stephane Bortzmeyer, John Cowan, Frank Ellerman, Doug Ewell.

The IETF LTRU working group adopted record-jar format on John Cowan's suggestion. That effort required record-jar to be documented and many people in that group contributed to this work there: the author thanks everyone who participated in that effort, even though names cannot be mustered here.

Author's Address

Addison Phillips (editor)
Yahoo! Inc.

Email: addison@inter-locale.com
URI: http://www.inter-locale.com

Full Copyright Statement

Intellectual Property

Acknowledgment

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