Network Working Group
Internet-Draft
Intended Status: Informational
Expires: July 1, 2015

S. Leonard
Penango, Inc.
December 28, 2014

text/markdown Use Cases
draft-seantek-text-markdown-use-cases-01

Abstract

This document elaborates upon the text/markdown media type for use with Markdown, a family of plain text formatting syntaxes that optionally can be converted to formal markup languages such as HTML. Background information, local storage strategies, and additional syntax registrations are supplied.

Status of this Memo

Copyright Notice

Table of Contents

1. Dive Into Markdown

This document serves as an informational companion to [MDMTREG], the text/markdown media type registration. It should be considered jointly with [MDMTREG].

"Sometimes the truth of a thing is not so much in the think of it, but in the feel of it." --Stanley Kubrick

1.1. On Formats

In computer systems, textual data is stored and processed using a continuum of techniques. On the one end is plain text: a linear sequence of characters in some character set (code), possibly interrupted by line breaks, page breaks, or other control characters. Plain text provides /some/ fixed facilities for formatting instructions, namely codes in the character set that have meanings other than "represent this character on the output medium"; however, these facilities are not particularly extensible. Compare with [RFC6838] Section 4.2.1. Applications may neuter the effects of these special characters by prohibiting them or by ignoring their dictated meanings, as is the case with how modern applications treat most control characters in US-ASCII. On this end, any text reader or editor that interprets the character set can be used to see or manipulate the text. If some characters are corrupted, the corruption is unlikely to affect the ability of a computer system to process the text (even if the human meaning is changed).

On the other end is binary format: a sequence of instructions intended for some computer application to interpret and act upon. Binary formats are flexible in that they can store non-textual data efficiently (perhaps storing no text at all, or only storing certain kinds of text for very specialized purposes). Binary formats require an application to be coded specifically to handle the format; no partial interoperability is possible. Furthermore, if even one byte or bit are corrupted in a binary format, it may prevent an application from processing any of the data correctly.

Between these two extremes lies formatted text, i.e., text that includes non-textual information coded in a particular way, that affects the interpretation of the text by computer programs. Formatted text is distinct from plain text and binary format in that the non-textual information is encoded into textual characters, which are assigned specialized meanings /not/ defined by the character set. With a regular text editor and a standard keyboard (or other standard input mechanism), a user can enter these textual characters to express the non-textual meanings. For example, a character like "<" no longer means "LESS-THAN SIGN"; it means the start of a tag or element that affects the document in some way.

On the formal end of the spectrum is markup, a family of languages for annotating a document in such a way that the annotations are syntactically distinguishable from the text. Markup languages are (reasonably) well-specified and tend to follow (mostly) standardized syntax rules. Examples of markup languages include SGML, HTML, XML, and LaTeX. Standardized rules lead to interoperability between markup processors, but a skill requirement for new (human) users of the language that they learn these rules in order to do useful work. This imposition makes markup less accessible for non-technical users (i.e., users who are unwilling or unable to invest in the requisite skill development).

```
informal        /---------formatted text----------\        formal
<------v-------------v-------------v----------------------v---->
 plain text     informal markup   formal markup    binary format
                (Markdown)        (HTML, XML, etc.)
```

Figure 1: Degrees of Formality in Data Storage Formats for Text

On the informal end of the spectrum are lightweight markup languages. In comparison with formal markup like XML, lightweight markup uses simple syntax, and is designed to be easy for humans to enter with basic text editors. Markdown, the subject of this document, is an /informal/ plain text formatting syntax that is intentionally targeted at non-technical users (i.e., users upon whom little to no skill development is imposed) using unspecialized tools (i.e., text boxes). Jeff Atwood once described these informal markup languages as "humane" [HUMANE].

1.2. Markdown Design Philosophy

Markdown specifically is a family of syntaxes that are based on the original work of John Gruber with substantial contributions from Aaron Swartz, released in 2004 [MARKDOWN]. Since its release a number of web or web-facing applications have incorporated Markdown into their text entry systems, frequently with custom extensions. Fed up with the complexity and security pitfalls of formal markup languages (e.g., HTML5) and proprietary binary formats (e.g., commercial word processing software), yet unwilling to be confined to the restrictions of plain text, many users have turned to Markdown for document processing. Whole toolchains now exist to support Markdown for online and offline projects.

Informality is a bedrock premise of Gruber’s design. Gruber created Markdown after disastrous experiences with strict XML and XHTML processing of syndicated feeds. In Mark Pilgrim’s "thought experiment", several websites went down because one site included invalid XHTML in a blog post, which was automatically copied via trackbacks across other sites [DIN2MD]. These scenarios led Gruber to believe that clients (e.g., web browsers) SHOULD try to make sense of data that they receive, rather than rejecting data simply because it fails to adhere to strict, unforgiving standards. (In [DIN2MD], Gruber compared Postel’s Law [RFC0793] with the XML standard, which says: "Once a fatal error is detected [...] the processor MUST NOT continue normal processing" [XML1.0-5].) As a result, there is no such thing as "invalid" Markdown; there is no standard demanding adherence to the Markdown syntax; there is no governing body that guides or impedes its development. If the Markdown syntax does not result in the "right" output (defined as output that the author wants, not output that adheres to some dictated system of rules), Gruber’s view is that the author either should keep on experimenting, or should change the processor to address the author’s particular needs (see [MARKDOWN] Readme and [MD102b8] perldoc; see also [CATPICS]).

1.3. Uses of Markdown

Since its introduction in 2004, Markdown has enjoyed remarkable success. Markdown works for users for three key reasons. First, the markup instructions (in text) look similar to the markup that they represent; therefore the cognitive burden to learn the syntax is low. Second, the primary arbiter of the syntax’s success is *running code*. The tool that converts the Markdown to a presentable format, and not a series of formal pronouncements by a standards body, is the basis for whether syntactic elements matter. Third, Markdown has become something of an Internet meme [INETMEME], in that Markdown gets received, reinterpreted, and reworked as additional communities encounter it. There are communities that are using Markdown for scholarly writing [CITE], for screenplays [FOUNTAIN], for mathematical formulae [CITE], and even for music annotation [CITE]. Clearly, a screenwriter has no use for specialized Markdown syntax for mathematicians; likewise, mathematicians do not need to identify characters or props in common ways. The overall gist is that all of these communities can take the common elements of Markdown (which are rooted in the common elements of HTML circa 2004) and build on them in ways that best fit their needs.

1.4. Uses of Labeling Markdown Content as text/markdown

The primary purpose of an Internet media type is to label "content" on the Internet, as distinct from "files". Content is any computer-readable format that can be represented as a primary sequence of octets, along with type-specific metadata (parameters) and type-agnostic metadata (protocol dependent). From this description, it is apparent that appending ".markdown" to the end of a filename is not a sufficient means to identify Markdown. Filenames are properties of files in file systems, but Markdown frequently exists in databases or content management systems (CMSes) where the file metaphor does not apply. One CMS [RAILFROG] uses media types to select appropriate processing, so a media type is necessary for the safe and interoperable use of Markdown.

Unlike complete HTML documents, [MDSYNTAX] provides no means to include metadata into the content stream. Several derivative flavors have invented metadata incorporation schemes (e.g., [MULTIMD]), but these schemes only address specific use cases. In general, the metadata must be supplied via supplementary means in an encapsulating protocol, format, or convention. The relationship between the content and the metadata is not directly addressed here or in [MDMTREG]; however, by identifying Markdown with a media type, Markdown content can participate as a first-class citizen with a wide spectrum of metadata schemes.

Finally, registering a media type through the IETF process is not trivial. Markdown can no longer be considered a "vendor"-specific innovation, but the registration requirements even in the vendor tree have proven to be overly burdensome for most Markdown implementers. Moreover, registering hundreds of Markdown variants with distinct media types would impede interoperability: virtually all Markdown content can be processed by virtually any Markdown processor, with varying degrees of success. The goal of [MDMTREG] is to reduce all of these burdens by having one media type that accommodates diversity and eases registration.

2. Strategies for Preserving Media Type and Parameters

The purpose of this document and [MDMTREG] is to promote interoperability between different Markdown-related systems, preserving the author’s intent. While [MARKDOWN] was designed by Gruber in 2004 as a simple way to write blog posts and comments, as of 2014 Markdown and its derivatives are rapidly becoming the formats of record for many communities and use cases. While an individual member of (or software tool for) a community can probably look at some "Markdown" and declare its meaning intuitively obvious, software systems in different communities (or different times) need help. [MDSYNTAX] does not have a signaling mechanism like <!DOCTYPE>, so tagging Markdown internally is simply out of the question. Once tags or metadata are introduced, the content is no longer "just" Markdown.

Some commentators have suggested that an in-band signaling mechanism, such as in Markdown link definitions at the top of the content, could be used to signal the variant. Unfortunately this signaling mechanism is incompatible with other Markdown variants (e.g., [PANDOC]) that expect their own kinds of metadata at the top of the file. Markdown content is just a stream of text; the semantics of that text can only be furnished by context.

The media type and variant parameter in [MDMTREG] furnish this missing context, while allowing for additional extensibility. This section covers strategies for how an application might preserve metadata when it leaves the domain of IETF protocols.

[MDMTREG] (draft-05) only defines two parameters: the charset parameter (required for all text/* media types) and the variant parameter. Character set interoperability is well-studied territory [NB: CITE?] and so is not further covered here. The variant parameter provides a simple identifier--nothing less or more. Variants are allowed to define additional parameters when sent with the text/markdown media type; the variant can also introduce control information into the textual content stream (such as via a metadata block). Neither [MDMTREG] nor this specification recommend any particular approach. However, the philosophy behind [MDMTREG] is to preserve formats rather than create new ones, since supporting existing toolchains is more realistic than creating novel ones that lack traction in the Markdown community.

2.1. Map to Filename and Attributes

This strategy is to map the media type, variant, and parameters to "attributes" or "forks" in the local convention. Firstly, Markdown content saved to a file should have an appropriate file extension ending in .md or .markdown, which serves to disambiguate it from other kinds of files. The character repertoire of variant identifiers in [MDMTREG] is designed to be compatible with most filename conventions. Therefore, a recommended strategy is to record the variant identifier as the prefix to the file extension. For example, for [PANDOC] content, a file could be named "example.pandoc.markdown".

Many filesystems are case-sensitive or case-preserving; however, file extensions tend to be all-lowercase. This document takes no position on whether variant identifiers should be case-preserved or all-lowercase when Markdown content is written to a file. However, when the variant identifier is read to influence operational behavior, it needs to be compared case-insensitively.

Many modern filesystems support "extended attributes", "alternate data streams", or "resource forks". Some version control systems support named properties. If the variant defines additional parameters, these parameters should be stored in these resources, where the parameter name includes the name of the resource, and the parameter value is the value of the resource (data in the resource), preferably UTF-8 encoded (unless the parameter definition explicitly defines a different encoding or repertoire). The variant identifier itself should be stored in a resource with a name including the term "variant".

2.2. Store Headers in Adjacent File

This strategy is to save the Markdown content in a first file, and to save the metadata (specifically the Content-Type: header) in a second file with a filename that is rationally related to the first filename. For example, if the first file is named "readme.markdown", the second file could be named "readme.markdown.headers". (If stored in a database, the analogy would be to store the metadata in a second table with a field that is a key to the first table.) This header file has the media type "message/global-headers" [RFC6533] (".u8hdr" suggestion notwithstanding).

2.3. "Arm" Content with MIME Headers

This strategy is to save the Markdown content along with its headers in a file, "arming" the content by prepending the MIME headers (specifically the Content-Type: header). It should be appreciated that the file is no longer a "Markdown file"; rather, it is an Internet Message Format file (e.g., [RFC5322]) with a Markdown content part. Therefore, the file should have an Internet message extension (e.g., ".eml", ".msg", or ".u8msg"), not a Markdown extension (e.g., ".md" or ".markdown").

2.4. Create a Local Batch Script

This strategy is to translate the processing instructions inferred from the Content-Type and other parameters (e.g., Content-Disposition) into a sequence of commands in the local convention, storing those commands in a batch script. For example, when a MIME-aware client stores some Markdown to disk, the client can save a Makefile in the same directory with commands that are appropriate (and safe) for the local system.

2.5. Process the Markdown

This strategy is to process the Markdown into the formal markup, which eliminates ambiguities. Once the Markdown is processed into (for example) valid XHTML, an application can save a file as "doc.xhtml" with no further loss of metadata. While unambiguous, this process may not be reversible.

2.6. Rely on Context

This last strategy is to use or create context to determine how to interpret the Markdown. For example, Markdown content that is of the Fountain.io type [FOUNTAIN] could be saved with the filename "script.fountain" instead of "script.markdown". Alternatively, scripts could be stored in a "/screenplays" directory while other kinds of Markdown could be stored elsewhere. For reasons that should be intuitively obvious, this method is the most error-prone. "Context" can be easily lost over time, and the trend of passing

Markdown between systems--taking them *out* of context--is increasing.

2.7. Specific Strategies

2.7.1. Subversion

This subsection covers a preservation strategy in Subversion [SVN], a common client-server version control system.

Subversion supports named properties. The "svn:mime-type" property duplicates the entire Content-Type header, so parameters SHOULD be stored there. The filename SHOULD be consistent with this Content-Type header, i.e., the extension SHOULD be the variant identifier plus ".markdown".

[[TODO: Versions of Subversion after [[1.x]] treat svn:mime-type as UTF-8 encoded, rather than US-ASCII. (See [RFC6532].) Therefore, the encoding of [RFC2231] will not be necessary in the vast majority of cases in newer versions. However, both for backwards compatibility and for support for non-Unicode character sets, [RFC2231] still needs to be supported.]]

[[TODO: Where to store Content-Disposition?]]

2.7.2. Git

This subsection covers a preservation strategy in Git [GIT], a common distributed version control system.

Versions of Git as of the time of this writing do not support arbitrary metadata storage; however, third-party projects add this support.

If Git is used without a metadata storage service, then a reasonable strategy is to include the variant identifier in the filename. The encoding of the file should be transcoded to UTF-8. For other properties, a header file should be recorded alongside the Markdown file in accordance with Section 2.2. The contents of the header file should be consistent with the rest of this paragraph, i.e., the charset parameter should be "UTF-8" and the variant parameter should match the identifier in the filename.

If a metadata storage service is used with Git, then use a convention that is most analogous to the service. For example, the "metastore" project emulates extended attributes (xattrs) of a POSIX-like system, so whatever "xattr" methodology is developed would be usable with metastore and Git.

3. Registration Templates for Common Markdown Syntaxes

The purpose of this section is to register certain syntaxes in the Markdown Syntaxes Registry [MDMTREG] because they illustrate particularly interesting use cases or are broadly applicable to the Internet community; thus, these syntaxes would benefit from the level of review associated with publication as IETF documents.

3.1. MultiMarkdown

Identifier: MultiMarkdown

Name: MultiMarkdown

Description:
MultiMarkdown (MMD) is a superset of "Original". It adds multiple syntax features (tables, footnotes, and citations, to name a few), and is intended to output to various formats. Additionally, it builds in "smart" typography for various languages (proper left- and right-sided quotes, for example).

Additional Parameters:

```
 options: String with zero or more of the following WSP-delimited
          tokens:

             "memoir" / "beamer"
             "full" / "snippet"
             "process-html"
             "random-footnote-identifiers"
             "accept"
             "reject"
             "nosmart"
             "nonotes"
             "nolabels"
             "nomask"

          The meanings of these tokens are defined in the
          MultiMarkdown documentation.
```

References:
<http://fletcher.github.io/MultiMarkdown-4/syntax>

Contact Information:

```
  (individual) Fletcher T. Penney <fletcher@fletcherpenney.net>
                      <http://fletcherpenney.net/multimarkdown/>
```

3.2. GitHub Flavored Markdown

Identifier: GFM

Name: GitHub Flavored Markdown

Description:
"Original" with the following differences:

1. Multiple underscores in words
2. URL (URI) autolinking
3. Strikethrough
4. Fenced code blocks
5. Syntax highlighting
6. Tables (- for rows; | for columns; : for alignment)
7. Only some HTML allowed; sanitization is integral
   to the format

References:
<https://help.github.com/articles/github-flavored-markdown/>
<https://github.com/github/markup/tree/master#html-sanitization>

Contact Information:
(corporate) GitHub, Inc. <https://github.com/contact>
[[Vicent Marti <vicent@github.com>??]]

3.3. Pandoc

Identifier: pandoc

Name: Pandoc

Description:
Markdown is designed to be easy to write and to read: the content should be publishable as-is, as plain text, without looking like it has been marked up with tags or formatting instructions. Yet whereas "Original" has HTML generation in mind, pandoc is designed for multiple output formats. Thus, while pandoc allows the embedding of raw HTML, it discourages it, and provides other, non-HTMLish ways of representing important document elements like definition lists, tables, mathematics, and footnotes.

Additional Parameters:
extensions: String with an optional starting syntax token, followed by a "+" and "-" delimited list of extension tokens. "+" preceding an extension token turns the extension on; "-" turns the extension off. The starting syntax tokens are "markdown", "markdown_strict", "markdown_phpextra", and "markdown_github". If no starting syntax token is given, "markdown" is assumed. The extension tokens include:

```
[[Stuff to turn off:]]

escaped_line_breaks
blank_before_header
header_attributes
auto_identifiers
implicit_header_references
blank_before_blockquote
fenced_code_blocks
fenced_code_attributes
line_blocks
fancy_lists
startnum
definition_lists
example_lists
table_captions
simple_tables
multiline_tables
grid_tables
pipe_tables
pandoc_title_block
yaml_metadata_block
all_symbols_escapable
intraword_underscores
strikeout
superscript
subscript
inline_code_attributes
tex_math_dollars
raw_html
markdown_in_html_blocks
native_divs
native_spans
raw_tex
latex_macros
implicit_figures
footnotes
inline_notes
citations

[[New stuff:]]

lists_without_preceding_blankline
hard_line_breaks
ignore_line_breaks
tex_math_single_backslash
tex_match_double_backslash
markdown_attribute
```

```
                   mmd_title_block
                   abbreviations
                   autolink_bare_uris
                   ascii_identifiers
                   link_attributes
                   mmd_header_identifiers
                   compact_definition_lists
```

Fragment Identifiers:
Pandoc defines fragment identifiers using the <id> in the
{#<id> .class ...} production (PHP Markdown Extra attribute block).
This syntax works for Header Identifiers and Code Block Identifiers.

References:
<http://johnmacfarlane.net/pandoc/README.html#pandocs-markdown>

Contact Information:
  (individual) Prof. John MacFarlane <jgm@berkeley.edu>
                         <http://johnmacfarlane.net/>

3.4. Fountain (Fountain.io)

Identifier: Fountain

Name: Fountain

Description:
Fountain is a simple markup syntax for writing, editing and sharing
screenplays in plain, human-readable text. Fountain allows you to
work on your screenplay anywhere, on any computer or tablet, using
any software that edits text files.

Fragment Identifiers:
See <http://fountain.io/syntax#section-titlepage> and
<http://fountain.io/syntax#section-sections>. In the following
fragment identifiers, the <key> and <sec*> productions MUST have "/"
characters percent-encoded.

```
#/       Title Page (acts as metadata).
#/<key>  Title Page; <key> is the key string.
#<sec1> *("/" <secn>)
         Section or subsection. The <sec1>..<secn>
         productions are the text of the Section line,
         with whitespace trimmed from both ends.
         Sub-sections (sections with multiple # at
         at the beginning of the line in the source)
         are addressed hierarchically by preceding
         the sub-section with higher-order
```

            sections. If the section hierarchy "skips",
            e.g., # to ###, use a blank section name,
            e.g., #Section/ACT%20I//PATIO%20SCENE.

References:
<http://fountain.io/syntax>

Contact Information:
  (individual) Stu Maschwitz <http://prolost.com/>
  (individual) John August <http://johnaugust.com/>

3.5. CommonMark

Identifier: CommonMark

Name: CommonMark

Description:
CommonMark is a standard, unambiguous syntax specification for Markdown, along with a suite of comprehensive tests to validate Markdown implementations against this specification. The maintainers believe that CommonMark is necessary, even essential, for the future of Markdown.

Compared to "Original", CommonMark is much longer and in a few instances contradicts "Original" based on seasoned experience. Although CommonMark specifically does not mandate any particular encoding for the input content, CommonMark draws in more of Unicode, UTF-8, and HTML (including HTML5) than "Original".

This registration always refers to the latest version or an unspecified version (receiver’s choice). Version 0.13 of the CommonMark specification was released 2014-12-10.

References:
<http://spec.commonmark.org/>

Contact Information:
  (individual) John MacFarlane <jgm@berkeley.edu>
  (individual) David Greenspan <david@meteor.com>
  (individual) Vicent Marti <vicent@github.com>
  (individual) Neil Williams <neil@reddit.com>
  (individual) Benjamin Dumke-von der Ehe <ben@stackexchange.com>
  (individual) Jeff Atwood <jatwood@codinghorror.com>

3.6. kramdown-rfc2629 (Markdown for RFCs)

Identifier: kramdown-rfc2629

Name: Markdown for RFCs

Description:
kramdown is a markdown parser by Thomas Leitner, which has a number of backends for generating HTML, Latex, and Markdown again. kramdown-rfc2629 is an additional backend to that: It allows the generation of XML2RFC XML markup (also known as RFC 2629 compliant markup).

References:
<https://github.com/cabo/kramdown-rfc2629>

Contact Information:
(individual) Carsten Bormann <cabo@tzi.org>

3.7. rfc7328 (Pandoc2rfc)

Identifier: rfc7328

Name: Pandoc2rfc

Description:
Pandoc2rfc allows authors to write in "pandoc" that is then transformed to XML and given to xml2rfc. The conversions are, in a way, amusing, as we start off with (almost) plain text, use elaborate XML, and end up with plain text again.

References:
RFC 7328
<https://github.com/miekg/pandoc2rfc>

Contact Information:
(individual) R. (Miek) Gieben <miek@google.com>

3.8. PHP Markdown Extra

Identifier: Extra

Name: Markdown Extra

Description:
Markdown Extra is an extension to PHP Markdown implementing some features currently not available with the plain Markdown syntax. Markdown Extra is available as a separate parser class in PHP Markdown Lib. Other implementations include Maruku (Ruby) and Python Markdown. Markdown Extra is supported in several content management systems, including Drupal, TYPO3, and MediaWiki.

Fragment Identifiers:
Markdown Extra defines fragment identifiers using the <id> in the {#<id> .class ...} production (attribute block). This syntax works for headers, fenced code blocks, links, and images.

References:
<https://michelf.ca/projects/php-markdown/extra/>

Contact Information:
(individual) Michel Fortin <michel.fortin@michelf.ca>

4. Examples for Common Markdown Syntaxes

This section provides examples of the variants registered in Appendix C.

4.1. MultiMarkdown

4.2. GitHub Flavored Markdown

4.3. Pandoc

4.4. Fountain (Fountain.io)

4.5. CommonMark

4.6. kramdown-rfc2629 (Markdown for RFCs)

4.7. rfc7328 (Pandoc2rfc)

[[TODO: complete.]]

5. IANA Considerations

IANA is asked to register the syntaxes specified in Section 3 in the Markdown Variants Registry.

6. Security Considerations

See the respective syntax descriptions and output media type registrations for their respective security considerations.

7. References

7.1. Normative References

[MARKDOWN] Gruber, J., "Daring Fireball: Markdown", December 2004, <http://daringfireball.net/projects/markdown/>.

[MDSYNTAX] Gruber, J., "Daring Fireball: Markdown Syntax Documentation", December 2004, <http://daringfireball.net/projects/markdown/syntax>.

[MDMTREG] Leonard, S., "The text/markdown Media Type", draft-ietf-appsawg-text-markdown-03 (work in progress), October 2014.

[RFC5147] Wilde, E. and M. Duerst, "URI Fragment Identifiers for the text/plain Media Type", RFC 5147, April 2008.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, October 2008.

7.2. Informative References

[HUMANE] Atwood, J., "Is HTML a Humane Markup Language?", May 2008, <http://blog.codinghorror.com/is-html-a-humane-markup-language/>.

[DIN2MD] Gruber, J., "Dive Into Markdown", March 2004, <http://daringfireball.net/2004/03/dive_into_markdown>.

[MD102b8] Gruber, J., "[ANN] Markdown.pl 1.0.2b8", May 2007, <http://six.pairlist.net/pipermail/markdown-discuss/2007-May/000615.html>, <http://daringfireball.net/projects/downloads/Markdown_1.0.2b8.tbz>.

[CATPICS] Gruber, J. and M. Arment, "The Talk Show: Ep. 88: ’Cat Pictures’ (Side 1)", July 2014, <http://daringfireball.net/thetalkshow/2014/07/19/ep-088>.

[INETMEME] Solon, O., "Richard Dawkins on the internet’s hijacking of the word ’meme’", June 2013, <http://www.wired.co.uk/news/archive/2013-06/20/richard-dawkins-memes>, <http://www.webcitation.org/6HzDGE9Go>.

[MULTIMD] Penney, F., "MultiMarkdown", April 2014, <http://fletcherpenney.net/multimarkdown/>.

[PANDOC] MacFarlane, J., "Pandoc", 2014, <http://johnmacfarlane.net/pandoc/>.

[RAILFROG] Railfrog Team, "Railfrog", April 2009, <http://railfrog.com/>.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC2231] Freed, N. and K. Moore, "MIME Parameter Value and Encoded Word Extensions: Character Sets, Languages, and Continuations", RFC 2231, November 1997.

[RFC4263] Lilly, B., "Media Subtype Registration for Media Type text/troff", RFC 4263, January 2006.

[RFC6533] Hansen, T., Ed., Newman, C. and A. Melnikov, "Internationalized Delivery Status and Disposition Notifications", RFC 6533, February 2012.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, January 2013.

[XML1.0-5] Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <http://www.w3.org/TR/2008/REC-xml-20081126#dt-fatal>.

[FOUNTAIN] Maschwitz, S. and J. August, "Fountain | A markup language for screenwriting.", 2014, <http://fountain.io/>.

[FTSYNTAX] Maschwitz, S. and J. August, "Syntax - Fountain | A markup language for screenwriting.", 1.1, March 2014, <http://fountain.io/syntax>.

[SVN] Apache Subversion, December 2014, <https://subversion.apache.org/>.

[GIT] Git, December 2014, <http://git-scm.com/>.

Author's Address

Sean Leonard
Penango, Inc.
5900 Wilshire Boulevard
21st Floor
Los Angeles, CA 90036
USA

EMail: dev+ietf@seantek.com
URI: http://www.penango.com/