Applications Area Working Group
Internet-Draft
Intended Status: Informational
Expires: March 13, 2015

S. Leonard
Penango, Inc.

September 9, 2014

The text/markdown Media Type
draft-ietf-appsawg-text-markdown-01.txt

Abstract

This document registers 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.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

1. Introduction

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).

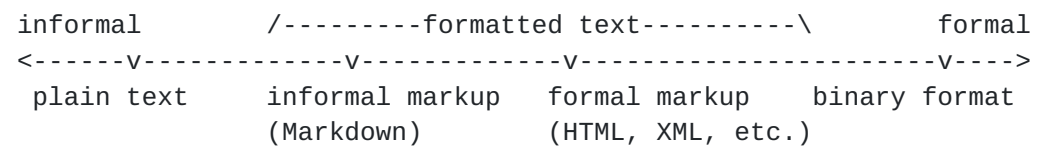

Figure 1: Degrees of Formality in Data Storage Formats for Text

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].

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 proprietary 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.

Due to Markdown's intentional informality, there is no standard specifying the Markdown syntax, and no governing body that guides or impedes its development. Markdown works for users for two 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 very 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.

To support identifying and conveying Markdown (as distinguished from plain text), this document defines a media type and parameters that indicate, in broad strokes, the author's intent on how to interpret the Markdown.

1.1. Requirements Terminology

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. Markdown Media Type Registration Applications

This section provides the media type registration application for the text/markdown media type (see [RFC6838], Section 5.6).

Type name: text

Subtype name: markdown

Required parameters: charset. Per Section 4.2.1 of [RFC6838], charset is REQUIRED. There is no default value. UTF-8 is RECOMMENDED; however, neither [MDSYNTAX] nor popular implementations at the time of this registration actually require or assume any particular encoding. In fact, many Markdown processors can get along just fine by operating on character codes that lie in the Portable Character Set (i.e., printable US-ASCII), blissfully oblivious to coded values outside of that range.

Optional parameters:

The following parameters reflect how the author intends the content to be processed. If rules and processor parameters are both supplied, the processor parameters take precedence.

rules: A whitespace-delimited list of Markdown processing rules that apply to this content. This parameter represents the intent of the author, namely, that the Markdown will be interpreted "best" (i.e., as the author intended) when processed with the rules as specified and ordered in this list. Identifiers MUST match the <token> production of [RFC2045]; whitespace MUST match the <FWS> production of [RFC5322].

Each identifier specifies a set of processing rules that are to be applied to the content during a processing operation. Rules are prioritized in order--later rules override earlier rules. For example, if this parameter is "Example1 Example2", and both Example1 and Example2 have specific rules to process { }-delimited content in conflicting ways, then the author's intent is to apply Example2's rules, not Example1's rules. However, the author intends for Example1's other rules (for example, to process "fenced code blocks") that are not overridden by Example2 to be applied to the content.

Rule identifiers are drawn from and registered in the IANA registry discussed below. Rule identifiers that represent significant variations of Markdown SHOULD begin with a capital letter, e.g., "GitHub", "Original", "Multi". Rule identifiers that represent single or small collections of rules SHOULD begin with a lowercase letter, e.g., "fenced-code-blocks", "fancy-lists", "strikeout".

When this parameter conveyed (even if empty), the implicit first rule is "Original", namely, the original Markdown rules provided in John Gruber's Markdown Syntax from 2004 [MDSYNTAX]. When this parameter is not conveyed, the author does not express any intent about which rules apply: [MDSYNTAX] may not necessarily be the author's intent.

In practice, Markdown implementations that are aware of this parameter will only be able to process a limited list of rules in an automated fashion. Therefore, when composing this parameter, it is RECOMMENDED that the composing process limit itself to small lists of broadly recognizable rules, namely lists with just one item (specifically, some major variation of Markdown).

processor: An identifier for the specific Markdown implementation that processes the Markdown into another format, such as HTML. If conveyed, this value cannot be empty. If not conveyed, the author does not express any intent about which processor should be used.

Processor identifiers are registered in the IANA registry discussed below. Each processor registration (which is expected to be updated over time) also defines the versions and arguments that are considered valid. The repertoire of this string is any number of characters (e.g., any Unicode character); however, registrations SHOULD stick to US-ASCII-based strings unless there is a compelling reason to do otherwise.

processor-ver: An identifier for the version of the processor identified in the processor parameter. If conveyed, this value cannot be empty. If not conveyed, the author does not express any intent regarding a particular version to be used. This parameter has no meaning if it is conveyed without a sibling processor parameter.

For purposes of this specification, the version is a string; the set of valid strings are registered and updated as a part of the processor registration. A version "2.0" does not necessarily imply that version 2.0 of an executable should be used instead of 2.0.1, 2.1, or even 3.0. If the processor has a way to be invoked "as if" it is a different version (e.g., version 3.0 of a

processor can process some content "as if" it were 2.0), then the receiver is free to use that invocation method. Updates to processor registrations SHOULD only add new versions when those new versions have a material difference on the interpretation of the Markdown content. If a processor has a version 2014.10 and a version 2014.11, for example, but 2014.11 only provides security updates, then the processor registration SHOULD NOT have a separate registration for the 2014.11 version.

The repertoire of this string is any number of characters (e.g., any Unicode character); however, registrations SHOULD stick to US-ASCII-based strings unless there is a compelling reason to do otherwise.

processor-args: A string conforming to a subset of the POSIX Shell Command Language in Volume 3, Chapter 2 of [POSIX.1-2008] for arguments that are to be passed in an invocation to the processor. The format of this parameter also has a facility to reference resources by URI. [[TODO: Put in a separate section? [[Section X]] discusses the details of this parameter.]]

If conveyed but empty, the author's intent is to turn off any optional arguments that the receiver would typically pass to the processor. If not conveyed, the author does not express any intent regarding particular arguments to be used. This parameter has no meaning if it is conveyed without a sibling processor parameter.

If conveyed and not empty, the string MUST be parseable to the <cmd_suffix> item in Volume 3, Section 2.10.2 of [POSIX.1-2008] with accommodations for embedded URIs as specified below; however, <io_redirect> items MUST NOT appear. Effectively, the string is a sequence of <WORD> tokens. The string MUST NOT contain any sequences that would cause any shell processing other than newline and quote removal. For example, the string MUST NOT contain redirects, pipelines, or comments. Section 2.6 Word Expansions (Section 2.6.7 Quote Removal notwithstanding) are right out!

A processor-args string MUST NOT include arguments regarding the input content or the output markup. For example, if a processor normally reads Markdown input using the arguments "-i filename" or "< filename" (i.e., from standard input), those arguments MUST be omitted. A processor-args string MUST NOT include arguments that have no bearing on the output, such as arguments that control verbosity of the processor (-v) or that cause side-effects (such as writing diagnostic messages to some other file). Of course, if warnings or errors are signaled within the output,

arguments enabling that output MAY be used.

Some authors may wish to combine inputs from multiple resources. For security reasons, file references MUST NOT be included in processor-args. Instead, references to resources are encoded in strictly-conforming URIs [RFC3986] delimited with angle brackets <>, which MUST NOT be escaped according to [POSIX.1-2008] (i.e., the brackets cannot be escaped with preceding backslash characters). A receiver may retrieve the resource specified by the URI, and then pass it to the processor in an appropriate way, such as via a temporary file. The intent of this option is to provide a means to include additional data that might accompany the Markdown content, for example, using cid: or mid: URLs [RFC2392] in the context of MIME messages.

Prior to invoking a Markdown processor, the preprocess routine MUST first analyze the processor-args string for URIs. Depending on privacy and security considerations, the routine either dereferences the URIs--retrieving the contents--or rejects the string. The URIs (including <> delimiters) shall then be replaced with appropriate, complete file paths or descriptors, and the resulting string shall be checked for conformance with a sequence of arguments as defined by the POSIX Shell Command Language in Volume 3, Chapter 2 of [POSIX.1-2008].

Not all processors are literally invoked from an operating system's command facility; some may be invoked from within another process as a library call. In such cases, the processor SHOULD be invoked in such a way to communicate the semantics of the arguments. One strategy might be to provide a library call with one or more explicit argument parameters; for example, either a string type of parameter (if the library does the parsing), or an "argc" plus "argv" pair of parameters (if the caller does the parsing). Another strategy might be to provide several different library calls, which the caller would choose to invoke depending on the directions of the arguments. In the registration for the processor, argument handling MUST be discussed.

The repertoire of this string is any number of characters that conform to a [POSIX.1-2008] implementation. (Note that the NULL character is excluded, because POSIX uses it to terminate strings.) When characters in the arguments lie outside of the Portable Character Set (i.e., outside of US-ASCII), this parameter MUST be encoded to preserve those characters and to signal the required encoding to the receiver. Then, the processor MUST be invoked in such a way that it properly understands these characters in the required encoding (or a superset thereof). When

encoded in a MIME Content-Type header, use of [RFC2231] is RECOMMENDED. The rationale for this (convoluted) requirement is because POSIX defines command lines and arguments with the C language char * data type, but leaves the character set dependent on locale environment variables (see Volume 1, Chapter 7 of [POSIX.1-2008]). Therefore, it is not sufficient to pass arguments from the processor-args parameter "as is" to the processor: the routine MUST change the locale or transform the arguments to an appropriate character encoding so that there is no ambiguity.

Encoding considerations: Text.

Security considerations:

Markdown interpreted as plain text is relatively harmless. A text editor need only display the text. The editor SHOULD take care to handle control characters appropriately, and to limit the effect of the Markdown to the text editing area itself; malicious Unicode-based Markdown could, for example, surreptitiously change the directionality of the text. An editor for normal text would already take these control characters into consideration, however.

Markdown interpreted as a precursor to other formats, such as HTML, carry all of the security considerations as the target formats. For example, HTML can contain instructions to execute scripts, redirect the user to other webpages, download remote content, and upload personally identifiable information. Markdown also can contain islands of formal markup, such as HTML. These islands of formal markup may be passed as-is, transformed, or ignored (perhaps because the islands are conditional or incompatible) when the Markdown is interpreted into the target format. Since Markdown may have different interpretations depending on the tool and the environment, a better approach is to analyze (and sanitize or block) the output markup, rather than attempting to analyze the Markdown.

[[TODO: discuss the implications of processor-args, and safeguards.]] [[TODO: discuss the security implications of combining supplementary resources in processor-args...the supplementary resources could be config files or scripts.]] [[TODO: discuss the privacy implications of dereferencing URIs.]]

Interoperability considerations:

Markdown flavors are designed to be broadly compatible with humans ("humane"), but not necessarily with each other. Therefore, syntax in one Markdown flavor may be ignored or treated differently in

another flavor. The overall effect is a general degradation of the output, proportional to the quantity of flavor-specific Markdown used in the text. When it is desirable to reflect the author's intent in the output, stick with the flavor identified in the flavor parameter.

Published specification: This specification.

Applications that use this media type:

Markdown conversion tools, Markdown WYSIWYG editors, and plain text editors and viewers; target markup processors indirectly use Markdown (e.g., web browsers for Markdown converted to HTML).

Additional information:

Magic number(s): None
File extension(s): .md, .markdown
Macintosh file type code(s): TEXT

Person & email address to contact for further information:

Sean Leonard <dev+ietf@seantek.com>

Restrictions on usage: None.

Author/Change controller: Sean Leonard <dev+ietf@seantek.com>

Intended usage: COMMON

Provisional registration? Yes

3. Example

The following is an example of Markdown as an e-mail attachment:

```
MIME-Version: 1.0
Content-Type: text/markdown; charset=UTF-8; rules=GitHub
Content-Disposition: attachment; filename=readme.md

Sample GitHub Markdown
==========

This is some sample GitHub Flavored Markdown (*GFM*).
The generated HTML is then run through filters in the
[html-pipeline](https://github.com/jch/html-pipeline)
to perform things like [sanitization](#html-sanitization) and
[syntax highlighting](#syntax-highlighting).

Bulleted Lists
-------

Here are some bulleted lists...

* One Potato
* Two Potato
* Three Potato

- One Tomato
- Two Tomato
- Three Tomato

More Information
-----------

[.markdown, .md](http://daringfireball.net/projects/markdown/)
has more information.
```

4. IANA Considerations

IANA is asked to register the media type text/markdown in the Standards tree using the application provided in Section 2 of this document.

IANA is also asked to establish a subtype registry called "Markdown Parameters". Entries in these registries is by Expert Review [RFC5226]. The registry has two sub-registries: a registry of rules and a registry of processors.

4.1 Registry of Rules

Each entry in this registry shall consist of a) a rule identifier and b) whether the rule is defined in the registry entry, or in some external document.

If the rule is defined in the registry entry, then the entry must also include: i) a list of rules in prose text, and ii) for each rule, an example illustrating the rule. Additionally, each registry entry shall describe in prose text iii) which rules take precedence over other rules, or how conflicts between rules may be resolved. The Expert will review the rule to determine whether the rule is plausible and whether the rule can be implemented.

If the rule is defined in some external document, the Expert will determine whether the registration represents a bona-fide variation of the Markdown syntax (i.e., neither a duplicate of an existing registration nor a syntax that is something other than Markdown; [MDSYNTAX] SHALL be used as a normative basis), a brief description, one or more responsible parties, whether the document is being maintained at the time of registration, and the existence of at least one complete tool (with or without documentation) that processes the Markdown syntax into a formal document language.

A responsible party can be an individual author or maintainer, a corporate author or maintainer (plus an individual contact), or a representative of a community of interest dedicated to the Markdown syntax.

The registry shall have the following initial value:

Identifier: Original

Description:
 The Markdown syntax as it exists in the Markdown 1.0.1 Perl script
 at [MARKDOWN], with accompanying documentation at [MDSYNTAX].

Responsible Parties:
 (individual)
 John Gruber <http://daringfireball.net/>
 <comments@daringfireball.net>

Currently Maintained? No

Tool:
 Name: Markdown 1.0.1
 Reference: <http://daringfireball.net/projects/markdown/>
 Purpose: Converts to HTML or XHTML circa 2004.

Additionally, the registry shall have the following identifiers reserved for future versions of this draft:

- Standard
- Common
- Regular
- Community
- Uniform
- Vanilla
- Compatible
- Gruber
- GitHub
- Multi
- PageDown

4.2 Registry of Processors

Each entry in this registry shall consist of a) a processor identifier, b) a concise description of the processor, c) one or more responsible parties, d) whether the processor is being maintained at the time of the registration (or registration update), (optionally) e) a list of version strings, and (optionally) f) documentation about the arguments.

If arguments are to be used, documentation MUST be provided as a part of the registry entry. However, the documentation MAY merely refer to external documentation, such as a manpage, webpage, or user manual.

[[TODO: figure out if the list should be more formal, so a receiver can validate the safety/correctness of the arguments before passing them along.]]

5. Security Considerations

See the answer to the Security Considerations template questions in Section 2.

6. References

6.1. Normative References

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

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

[POSIX.1-2008] IEEE Std 1003.1, 2013 Edition (incorporates IEEE Std 1003.1-2008 and IEEE Std 1003.1-2008/Cor 1-2013, "Standard for Information Technology - Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7" (incorporating Technical Corrigendum 1), April 2013.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

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

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC5226] Narten, T., and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, May 2008.

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

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

6.2. Informative References

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

[RFC2392] Levinson, E., "Content-ID and Message-ID Uniform Resource Locators", RFC 2392, August 1998.

Appendix A. Change Log

This draft is a continuation from draft-seantek-text-markdown-media-type-00.txt (since renamed). These technical changes were made:

1. The flavor parameter was replaced with the rules, processor, processor-ver, and processor-args parameters.
2. The IANA Considerations now covers the rules and processors.
3. The charset parameter was modified.
4. The example was updated to reflect the current specification.

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/