Applications Area Working Group
Internet-Draft
Intended Status: Informational
Expires: March 26, 2015

S. Leonard
Penango, Inc.

September 22, 2014

The text/markdown Media Type
draft-ietf-appsawg-text-markdown-02.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."

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

[[TODO: add table of contents.]]

1. Introduction

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. [[TODO: CITE.]] 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).

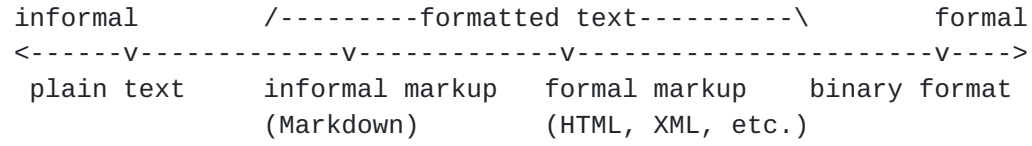

Figure 1: Degrees of Formality in Data Storage Formats for Text

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].

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" [XML1.0-3].) 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 [CITE], 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

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. This registration draws particular inspiration from the text/troff registration [RFC4263]; troff is an informal plain text formatting syntax primarily intended for output to monospace line-oriented printers and screen devices. In that sense, Markdown is a kind of troff for modern computing.

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 by this specification; 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 this specification is to reduce all of these burdens by having one media type that accommodates diversity and eases registration.

1.3. 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. Example

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

```
MIME-Version: 1.0
Content-Type: text/markdown; charset=UTF-8; flavor=Original;
 processor="Markdown.pl-1.0.2b8 --html4tags"
Content-Disposition: attachment; filename=readme.md

Sample HTML 4 Markdown
==========

This is some sample Markdown. [Hooray!][foo]
(Remember that link names are not case-sensitive.)

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.

[fOo]: http://example.com/some/foo/location
  'This Title Will Not Work with Markdown.pl-1.0.1'
```

3. Markdown Media Type Registration Application

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 the author's intent regarding the content. A detailed specification can be found in Section 4.

flavor: The variant, or "flavor" of the Markdown content, with optional rules (qualifiers). Default value: "Original".

processor: A specific Markdown implementation, with optional arguments. Default value: none (receiver's choice).

output-type: The Content-Type (Internet media type) of the output, with optional parameters. Default value: "text/html".

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.

Specific security considerations apply to the optional parameters;

for details, consult Section 4.

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

4. Optional Parameters

The following optional parameters can be used by an author to indicate the author's intent regarding how the Markdown ought to be processed. For security and accuracy, IANA registries will be created. However, authors who wish to use custom values by private agreement may do so via an extension mechanism; all unregistered identifiers MUST start with an exclamation mark "!".

All identifiers are case-sensitive; receivers MUST compare for exact equality. Identifiers MUST NOT be registered if another registration differs only in the casing, as these registrations may cause confusion.

The following ABNF definitions are used in this section:

```
EXTCHAR  = <any character outside the US-ASCII range,
            essentially amounting to any Unicode
            code point beyond U+007F without requiring
            Unicode or any particular encoding>

REXTCHAR = <EXTCHAR without spaces (Zs category) or
            control characters>
```

Figure X: ABNF Used in This Section

The discussion in this section presumes that the parameter values are discrete strings. When encoded in protocols such as MIME [RFC2045], however, the value strings MUST be escaped properly.

4.1. flavor

The flavor parameter indicates the Markdown variant in which the author composed the content. The overall intent of this parameter is to provide a facility for Markdown tools, such as graphical editors, to be able to broadly categorize the content and perform useful services such as syntax highlighting without resorting to executing the Markdown processor. Of course, actual recipients may use this information for any useful purpose, including picking and configuring an appropriate Markdown processor. The entire parameter is case-sensitive.

An IANA registry of flavors will be created as discussed in Section 5. A flavor identifier is composed of two or more Unicode characters excluding spaces (Zs category), control characters, the hyphen-minus "-", quotation marks """, and the plus sign "+"; however, ASCII characters alone SHOULD be used. Additionally, registered flavor identifiers MUST NOT begin with "!", the exclamation mark. By convention, flavor identifiers start with a capital letter (when using Roman characters), but this is not a requirement. Unregistered flavor identifiers MUST begin with "!" (plus two additional characters).

When omitted, the default value is "Original". Its meaning is covered in Section 5. Generators MUST NOT emit empty flavor parameters, but parsers MUST treat empty flavor parameters the same as if omitted.

The full ABNF of the flavor parameter is:

```
flavor-param     = flavor *( *WSP rule ) *WSP

flavor           = registered-fid / unregistered-fid

registered-fid   = fid-char 1*("!" / fid-char)

unregistered-fid = "!" 2*fid-char

fid-char         = %d35-%d42 / %d44 / %d46-%d126 / REXTCHAR

rule             = "+" (should-rule / any-rule)

should-rule      = should-rule-char [ *(should-rule-char / "_")
                                      should-rule-char ]
any-rule         = 1*rule-char

rule-char        = %d35-%d42 / %d44-%d126 / REXTCHAR
```

Figure X: ABNF of the flavor parameter

4.1.1. flavor rules

[[TODO: consider. This section is mainly inspired from pandoc.]] Most flavors are self-contained, with no options. However, some flavors have optional rules that may be applied with discretion. For those flavors where optional rules are an integral feature, the author MAY indicate that those extra rules be applied in a plus sign-delimited list.

Because Markdown has no inherent concept of validity, authors SHOULD be aware that receivers are not required to honor these optional rules--the special characters in the Markdown content may well be interpreted as plain text, rather than Markdown markup. Generally speaking, defining a new (simple) flavor is preferable to defining a complex flavor with multiple optional rules.

A flavor rule identifier is composed of any sequence of Unicode characters excluding spaces (Zs category), control characters, quotation marks """, exclamation marks "!", and the plus sign "+"; however, lowercase ASCII letters and the underscore "_" alone SHOULD be used, where the underscore SHOULD NOT be at the beginning or end. The syntax for flavor rules derives in significant part from pandoc [PANDOC].

[[TODO: There are no requirements about exclamation marks for unregistered rules...flavor rules SHOULD be registered along with the

flavor, but a receiver does not need to reject the flavor parameter simply because it does not recognize a rule...it can just ignore the rule.]]

4.2. processor

The processor parameter indicates the specific Markdown implementation that the author intends be used. The purpose of this parameter is to control the automatic processing of Markdown into some output format, but of course actual recipients may use this information for any useful purpose. The entire parameter is case-sensitive.

An IANA registry of processors will be created as discussed in Section 5. A processor identifier is composed of two or more Unicode characters excluding spaces (Zs category), control characters, the hyphen-minus "-", quotation marks """, the less-than sign "<", and the greater-than sign ">"; however, ASCII characters alone SHOULD be used. Additionally, registered processor identifiers MUST NOT begin with "!", the exclamation mark. Unregistered processor identifiers MUST begin with "!" (plus two additional characters).

When omitted, the default value is to use whatever processor the receiver prefers. Generators MUST NOT emit empty processor parameters, but parsers MUST treat empty processor parameters the same as if omitted.

The full ABNF of the processor parameter is:

```
processor-param  = processor [ "-" version ]
                   *( 1*WSP argument ) *WSP

processor        = registered-pid / unregistered-pid

registered-pid   = pid-char 1*("!" / pid-char)

unregistered-pid = "!" 2*pid-char

version          = pid-char *("!" / pid-char)

argument         = regular-argument / uri-argument

regular-argument = 1*(regular-char / quoted-chars)

pid-char         = %d35-%d44 / %d46-%d59 / %d61 /
                   %d63-126 / REXTCHAR

regular-char     = %d33 / %d35-%d59 / %d61 / %d63-126 / REXTCHAR
```

```
quoted-chars     = DQUOTE *pqcontent DQUOTE

pqcontent        = %d1-%d33 / %d35-127 / EXTCHAR / DQUOTE DQUOTE

uri-argument     =  "<" URI-reference ">"         ; from [RFC3986]
```

Figure X: processor parameter ABNF

4.2.1. processor version

For better precision, an author MAY include the processor version. The version is delimited from the processor identifier with a hyphen-minus "-"; the version string itself is an opaque string. Version strings (e.g., "2.0", "3.0.5") are registered and updated along with the processor registration. 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 performance updates, then the processor registration SHOULD NOT separately register the "2014.11" version. The repertoire of the version string is the same as the processor identifier (and like the processor identifier, ASCII characters alone SHOULD be used).

A receiver that recognizes the processor but not the processor version MAY use any version of the processor, preferably the latest version.

4.2.2. processor arguments

Processor arguments MAY be supplied for finer-grained control over how the processor behaves. Multiple arguments and URI references are supported.

4.2.2.1. Quoted Arguments

According to the ABNF above, arguments are delimited by whitespace. Quotation marks are used to support zero-length arguments, as well as whitespace or quotation marks in a single argument. If a quotation mark appears anywhere in the argument, the following text is considered quoted; two successive quotation marks "" mean one quotation mark. A single quotation mark ends the quoting. Because of this rule, quotation marks do not have to appear at the termini of an argument; embedded quotation marks start (and end) quoting within a single argument. For example:

```
   a""b
```

means:

```
   ab
```

for the actual argument.

4.2.2.2. URI Reference Arguments

Certain processors can take supplementary content, such as metadata, from other resources. To support these workflows, an author MAY use the URI delimiters <> to signal a URI, such as cid: or mid: URLs [RFC2392] in the context of MIME messages. The URI MUST comply with [RFC3986], and MAY be a relative reference if the subject Markdown content has a base URI. The receiver is to interpret this as a request to retrieve the resource, and to supply that resource in a local reference form that the processor can use (e.g., via a temporary file). The URI MUST be entire argument; the URI cannot be combined with other text to constitute the argument (and the ABNF above supports this restriction). The reason for this restriction is security, so that a maliciously constructed argument string cannot resolve to some other file reference (such as parent directories like ../ or special files such as /dev/hd0). If the processor accepts URI strings directly, the string is to be supplied as a regular string without <> delimiters. For security reasons, direct file references MUST NOT be included in the processor arguments.

The prior paragraph notwithstanding, certain workflows may require file references. In such cases, file: URLs [RFC1738] (including relative references) are appropriate. The receiver SHOULD apply the same security and privacy analyses to file: URLs as it would to any other URI.

4.2.2.3. Appropriate Arguments and Security Considerations

Not all arguments are appropriate for inclusion in the processor parameter. Appropriate arguments are basically limited to those that affect the output markup, without side-effects. Arguments MUST NOT identify input sources or output destinations. 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. Arguments that have no bearing on the output MUST be omitted as well, 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.

When in doubt, a receiver SHOULD omit arguments with unknown or undocumented effects, and MAY ignore author-supplied arguments entirely, but SHALL NOT reorder arguments. An author has very little assurance that a receiver will honor unregistered arguments. Consequently, the burden is squarely on processor registrants

(Section 5.2) to document their arguments properly.

For security reasons, the parsed argument array (or a string unambiguously representing the delimited argument array) MUST be passed directly to the processor. Emitting the argument array as-is in a batch script (for example) may cause risky side effects, such as automatic substitutions, alias activation, or macro execution. The arguments in this parameter MUST be encoded to preserve characters outside of US-ASCII, and to signal the required encoding to the receiver. When going between (system) processes, some implementations may interpret character codes based on locale environment variables. Therefore, it is not sufficient to pass arguments from this 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. Furthermore, the NUL character (%d0, U+0000) is not permitted because most common operating systems use that code point as a delimiter.

4.2.3. Examples of processor parameters

[[TODO: provide examples.]]

4.3. output-type

The output-type parameter indicates the Internet media type (and parameters) of the output from the processor.

When omitted, the default value is "text/html". Generators MUST NOT emit empty output-type parameters, but parsers MUST treat empty output-type parameters the same as if omitted.

The default value of text/html ought to be suitable for the majority of current purposes. However, Markdown is increasingly becoming integral to workflows where HTML is not the target output; examples range from TeX [CITE], to PDF [CITE], to OPML [CITE], and even to entire e-books [CITE].

Security provides a significant motivator for this parameter. Most Markdown processors emit byte (octet) streams; without a well-defined means for a Markdown processor to pass metadata onwards, it is perilous for post-processing to assume that the content is always HTML. A processor might emit PostScript (application/postscript) content, for example, in which case an HTML sanitizer would fail to excise dangerous instructions.

The value of output-type is an Internet media type with optional parameters. The syntax (including case sensitivity considerations) is the same as specified in [RFC2045] for the Content-Type header (with

updates over time), namely:

```
type "/" subtype *(";" parameter)
                ; Matching of media type and subtype
                ; is ALWAYS case-insensitive.
```

Figure X: Content-Type ABNF (from [RFC2045])

The Internet media type in the output-type parameter MUST be observed. Processors or processor arguments that conflict with the output-type parameter MUST be re-chosen, ignored, or rejected.

Although arbitrary optional parameters may be passed along with the Internet media type, receivers are under no obligation to honor or interpret them in any particular way. For example, the parameter value "text/plain; format=flowed; charset=ISO-2022-JP" obligates the receiver to output text/plain (and to treat the output as plain text--no sneaking in or labeling the output as HTML!). In contrast, such a parameter value neither obligates the receiver to follow [RFC3676] (for flowed output) nor to output ISO-2022-JP Japanese character encoding (see [RFC1468]).

Markdown implementations for all kinds of formats already exist, including formats that are not registered Internet media types, or that are inexpressible as Internet media types. For example, one Markdown processor for the mass media industry outputs formatted screenplays [CITE to fountain.io]: none of applicable media types application/pdf, text/html, or text/plain adequately distinguish this kind of output. Such distinctions SHOULD be made in the processor parameter (and to a lesser extent, the flavor parameter), underscoring that the primary concern of the output-type parameter is making technical and security-related decisions.

The output-type parameter does not distinguish between fragment content and whole-document content. A Markdown processor MAY (and typically will) output HTML or XHTML fragment content, without preambles or postambles such as <!DOCTYPE>, <html>, <head>, </head>, <body>, </body>, or </html> elements. Receivers MUST be aware of this behavior and take appropriate precautions.

[[TODO: consider.]]
The author may specify the output-type "text/markdown", which has a special meaning. "text/markdown" means that the author does not want to invoke Markdown processing at all: the receiver SHOULD view the Markdown source as-is. In this case, the processor choice has little practical effect because the Markdown is not actually processed, but other tools can use the flavor parameter (and secondarily if so inclined, the processor parameter) to perform useful services such as

syntax highlighting. This output-type is not the default because one generally assumes that Markdown is meant for composing rather than reading: readers expect to see the output format (or dual-display of the output and the Markdown). However, if authors are collaboratively editing a document or are discussing Markdown, "text/markdown" may make sense. While the optional parameter output-type may be used recursively (as a sneaky way to stash the author's follow-on or secondary intent), receivers are not obligated to recognize it; optional parameters internal to output-type MAY be ignored.

5. 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". The registry has two sub-registries: a registry of flavors and a registry of processors.

5.1. Registry of Flavors

Each entry in this registry shall consist of a flavor identifier and information about the flavor, as follows:

5.1.1. Flavor Template

```
Identifier: [Identifier]

Description: [Concise, prose description of the syntax,
              with emphasis on its purpose, the community
              that it addresses, and notable variations
              from [MDSYNTAX] or another flavor.]

Documentation: [References to documentation.]

Rules:
 {for each rule}
  Identifier: [Identifier]
  Description: [Concise, prose description of the rule.]
  Documentation: [References to documentation.]

Responsible Parties:
 {for each party}
  ([type: individual, corporate, representative])
  [Name] <contact info 1>...<contact info n>

Currently Maintained? [Yes/No]
```

```
Tools:
 {for each tool}
  Name: [Name]
  Version(s): [Significant version or versions that
               implement the flavor]
  Type: ["Processor" or some other type]
  Reference(s): <contact info 1>...<contact info n>
  Purpose: [Concise, prose description of the tool.]
```

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.

Multiple tools MAY be listed, but only one is necessary for a successful registration. If a tool is a Markdown processor, it MUST be registered; however, any Markdown-related tool (for example, graphical editors, emacs "major modes", web apps) is acceptable. The purpose of the tool requirement is to ensure that the flavor is actually used in practice.

5.1.2. Initial Registration

The registry shall have the following initial registration:

Identifier: Original

Description: Gruber's original Markdown syntax.

Documentation: [MDSYNTAX]

Rules: None.

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

Currently Maintained? No

Tools:
Name: Markdown.pl
Version(s): 1.0.1, 1.0.2b8
Type: Processor
Reference(s): <http://daringfireball.net/projects/markdown/>
Purpose: Converts Markdown to HTML or XHTML circa 2004.

5.1.3. Reserved Identifiers

The flavors registry SHALL have the following identifiers RESERVED. No one is allowed to register them (or any case variations of them).

Standard
Common
Markdown

5.1.4. Standard of Review

Registrations are made by a highly constrained Expert Review [RFC5226] that amounts more-or-less to First-Come, First-Served with sanity checking.

The designated expert SHALL review the flavor registration. The identifier MUST comply with the syntax specified in this document. Additionally, the identifier MUST NOT differ from other registered identifiers merely by case. The description and documentation SHOULD provide sufficient guidance to an implementer to implement a tool to handle the flavor. The designated expert SHOULD warn the registrant if the description and documentation are inadequate; however, inadequacy (in the opinion of the designated expert) will not bar a registration.

All references (including contact information) MUST be verified as functional at the time of the registration.

If rules are included in the registration, the rule identifiers MUST comply with the syntax specified in this document. The description and documentation of each rule SHOULD provide sufficient guidance to an implementer to implement a tool to handle the rule. The designated expert SHOULD warn the registrant if the description and documentation are inadequate; however, inadequacy (in the opinion of the designated expert) will not bar a registration.

The designated expert MUST determine that all tools listed in the registration are real implementations. If a tool is a Markdown processor, the processor MUST be registered in the Registry of Flavors in Section 5.2. The designated expert MAY request that the registrant provide evidence that a tool actually works (for example, that it passes certain test suites); however, the failure of a tool to work according to the flavor registration will not bar a registration. (For example, not even Gruber's own Markdown.pl implementation complies with [MDSYNTAX]. C'est la vie!)

If a registration is being updated, the designated expert SHOULD verify that the updating registrant matches the contact information on the prior registration, and if not, that the updating registrant has authority from the prior registrant to update it. All fields may be updated except the Identifier, which is permanent: not even case

may be changed.

5.2. Registry of Processors

Each entry in this registry SHALL consist of a processor identifier and information about the processor, as follows:

5.2.1. Processor Template

```
Identifier: [Identifier]

Description: [Concise, prose description of the processor,
              with emphasis on its purpose, the community
              that it addresses, and notable variations
              from [MDSYNTAX] or another flavor.]

Documentation: [References to documentation.]

Versions:
 {for each version}
  Identifier: [Identifier]
  Description: [Optional, concise, prose description of the
    version. "N/A" SHALL be used to indicate no description.]

Arguments:
 {in general}
  Argument Ordering: [Concise, prose description of how
                      arguments need to be ordered.]
 {for each argument}
  Argument Syntax: [Syntax here; multiple consecutive argument
    positions are allowed, separated by a single space. Use
    braces for variable information (add : for example input),
    <URI> for URI references, and .. for sequences of arguments
    with # as a placeholder for the number of arguments or
    ..-.. to indicate the first character of the subsequent
    argument that ends the sequence, e.g.:
                    -c
                    --title {title: "The Rain in Spain"}
                    --metadata <URI>
                    --bullet-chars:{#} {char 1}..{char #}
                    --verbs {verb: walk, run, sleep}..-..
                    ]
  Description: [Concise, prose description of the argument.]
  Documentation: [References to documentation.]

Output Type(s): [Internet media types, comma-separated
                 (with optional LWSP)]
```

```
Security Considerations: [Sufficient description of risks and
                          other considerations; "N/A" or
                          "None" responses are insufficient.]

Responsible Parties:
 {for each party}
  ([type: individual, corporate, representative])
  [Name] <contact info 1>...<contact info n>

Currently Maintained? [Yes/No]
```

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 processor.

5.2.2. Initial Registration

The registry shall have the following initial registration:

```
Identifier: Markdown.pl

Description: Gruber's original Markdown processor, written in
             Perl. Requires Perl 5.6.0 or later. "Welcome to
             the 21st Century." Works with Movable Type 2.6+,
             Blosxom 2.0+, BBEdit 6.1+, and the command-line.

Documentation: [MARKDOWN]

Versions:
 Identifier: 1.0.1
 Description: The 2004-12-17 version.

 Identifier: 1.0.2b8
 Description: The 2007-05-09 version. Fixes many bugs
              and adds several new features; see
              VERSION HISTORY in Markdown.pl.
```

```
Arguments:
 Argument Syntax: --html4tags
 Description:
   "Use the --html4tags command-line switch to produce HTML
   output from a Unix-style command line."
   Without this argument, Markdown.pl outputs XHTML style
   tags by default, e.g.: <br />. Even though XHTML style
   is the default, the output SHOULD be analyzed as
   text/html; the processor makes no attempt to make
   its output well-formed application/html+xml
   (not surprising--see the design philosophy).
 Documentation: [MARKDOWN]

Output Type: text/html

Security Considerations: The security of this implementation
                         has not been fully analyzed.

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

Currently Maintained? No [[TODO: maybe?]]
```

5.2.3. Reserved Identifiers

The processors registry SHALL have the following identifiers RESERVED. No one is allowed to register them (or any case variations of them).

Standard
Markdown
md

5.2.4. Standard of Review

Registrations are First-Come, First-Served [RFC5226]. The checks prescribed by this section can be performed automatically.

The identifier MUST comply with the syntax specified in this document. Additionally, the identifier MUST NOT differ from other registered identifiers merely by case. The description and documentation SHOULD provide sufficient guidance to an implementer to know how to invoke the processor and handle the output.

All references (including contact information) MUST be verified as functional at the time of the registration.

If arguments are included in the registration, the Argument Syntax

MUST comply with the template instructions in Section 5.2.1. Each description and documentation field SHOULD provide sufficient guidance to an implementer to know how to invoke the processor and handle the output.

The Security Considerations field is not optional; it MUST be provided.

If a registration is being updated, the contact information MUST either match the prior registration and be verified, or the prior registrant MUST confirm that the updating registrant has authority to update the registration. All fields may be updated except the Identifier, which is permanent: not even case may be changed.

6. Security Considerations

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

Security considerations for the optional parameters are integrated throughout Section 4.

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>.

[RFC1738] Berners-Lee, T., Masinter, L., and M. McCahill, "Uniform Resource Locators (URL)", RFC 1738, December 1994.

[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.

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.

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

[RFC4263] Lilly, B., "Media Subtype Registration for Media Type

text/troff", RFC 4263, January 2006.

[XML1.0-3] Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Third Edition)", World Wide Web Consortium Recommendation REC-xml-20040204, February 2004, <http://www.w3.org/TR/2004/REC-xml-20040204#dt-fatal>.

[TODO] [[Add remaining references.]]

Appendix A. Change Log

This draft is a continuation from draft-ietf-appsawg-text-markdown-01.txt. These technical changes were made:

1. The entire document was reorganized: optional parameters now have their own section, and the Introduction section is divided into four subsections.
2. The Introduction section provides substantial background information, along with goals and use cases for both Markdown and the Internet media type registration.
3. The rules parameter was reverted back to flavor, and flavor was beefed up.
4. The processor parameters were consolidated and simplified.
5. Dependencies on POSIX were removed.
6. The output-type parameter was added.
7. Unregistered identifiers can be used with their own ! syntax.
8. The IANA Considerations section was fleshed out in great detail, with emphasis on easing the registration process.
9. Security considerations were weaved throughout the specification. Overall, most of the complexity in this specification comes directly from the security considerations. Those considerations are necessary since a lot of bad things can and will happen when HTML, URIs, and executable code get together.
10. Changed the example in Section 2 to use initially registered identifiers.
11. Added output-type="text/markdown" for recursive handling (i.e., don't process this Markdown, just show it like it is).

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/