Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 4, 2018

H. Birkholz
Fraunhofer SIT
C. Vigano
Universitaet Bremen
C. Bormann
Universitaet Bremen TZI

July 03, 2017

Concise data definition language (CDDL): a notational convention to express CBOR data structures
draft-greevenbosch-appsawg-cbor-cddl-11

Abstract

This document proposes a notational convention to express CBOR data structures (RFC 7049). Its main goal is to provide an easy and unambiguous way to express structures for protocol messages and data formats that use CBOR.

Table of Contents

1. Introduction

In this document, a notational convention to express CBOR [RFC7049] data structures is defined.

The main goal for the convention is to provide a unified notation that can be used when defining protocols that use CBOR. We term the convention "Concise data definition language", or CDDL.

The CBOR notational convention has the following goals:

(G1) Provide an unambiguous description of the overall structure of a CBOR data structure.

(G2) Flexibility to express the freedoms of choice in the CBOR data format.

(G3) Possibility to restrict format choices where appropriate [_format].

(G4) Able to express common CBOR datatypes and structures.

(G5) Human and machine readable and processable.

(G6) Automatic checking of data format compliance.

(G7) Extraction of specific elements from CBOR data for further processing.

Not an explicit goal per se, but a convenient side effect of the JSON generic data model being a subset of the CBOR generic data model, is the fact that CDDL can also be used for describing JSON data structures (see Appendix E.1).

This document has the following structure:

The syntax of CDDL is defined in Section 3. Examples of CDDL and related CBOR data instances are defined in Appendix H. Section 4 discusses usage of CDDL. Examples are provided early in the text to better illustrate concept definitions. A formal definition of CDDL using ABNF grammar is provided in Appendix D. Finally, a prelude of standard CDDL definitions available in every CBOR specification is listed in Appendix E.

1.1. Requirements notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119, BCP 14 [RFC2119].

1.2. Terminology

New terms are introduced in _cursive_. CDDL text in the running text is in "typewriter".

2. The Style of Data Structure Specification

CDDL focuses on styles of specification that are in use in the community employing the data model as pioneered by JSON and now refined in CBOR.

There are a number of more or less atomic elements of a CBOR data model, such as numbers, simple values (false, true, nil), text and byte strings; CDDL does not focus on specifying their structure. CDDL of course also allows adding a CBOR tag to a data item.

The more important components of a data structure definition language are the data types used for composition: arrays and maps in CBOR (called arrays and objects in JSON). While these are only two representation formats, they are used to specify four loosely distinguishable styles of composition:

- A _vector_, an array of elements that are mostly of the same semantics. The set of signatures associated with a signed data item is a typical application of a vector.

- A _record_, an array the elements of which have different, positionally defined semantics, as detailed in the data structure definition. A 2D point, specified as an array of an x coordinate (which comes first) and a y coordinate (coming second) is an example of a record, as is the pair of exponent (first) and mantissa (second) in a CBOR decimal fraction.

- A _table_, a map from a domain of map keys to a domain of map values, that are mostly of the same semantics. A set of language tags, each mapped to a text string translated to that specific language, is an example of a table. The key domain is usually not limited to a specific set by the specification, but open for the application, e.g., in a table mapping IP addresses to MAC addresses, the specification does not attempt to foresee all possible IP addresses.

- A _struct_, a map from a domain of map keys as defined by the specification to a domain of map values the semantics of each of which is bound to a specific map key. This is what many people have in mind when they think about JSON objects; CBOR adds the ability to use map keys that are not just text strings. Structs can be used to solve similar problems as records; the use of explicit map keys facilitates optionality and extensibility.

Two important concepts provide the foundation for CDDL:

1. Instead of defining all four types of composition in CDDL separately, or even defining one kind for arrays (vectors and records) and one kind for maps (tables and structs), there is only one kind of composition in CDDL: the _group_ (Section 2.1).

2. The other important concept is that of a _type_. The entire CDDL specification defines a type (the one defined by its first _rule_), which formally is the set of CBOR instances that are acceptable for this specification. CDDL predefines a number of basic types such as "uint" (unsigned integer) or "tstr" (text string), often making use of a simple formal notation for CBOR data items. Each value that can be expressed as a CBOR data item also is a type in its own right, e.g. "1". A type can be built as a _choice_ of other types, e.g., an "int" is either a "uint" or a "nint" (negative integer). Finally, a type can be built as an array or a map from a group.

2.1. Groups and Composition in CDDL

CDDL Groups are lists of name/value pairs (group _entries_).

In an array context, only the value of the entry is represented; the name is annotation only (and can be left off if not needed). In a map context, the names become the map keys ("member keys").

In an array context, the sequence of elements in the group is important, as it is the information that allows associating actual array elements with entries in the group. In a map context, the sequence of entries in a group is not relevant (but there is still a need to write down group entries in a sequence).

A group can be placed in (round) parentheses, and given a name by using it in a rule:

```
pii = (
  age: int,
  name: tstr,
  employer: tstr,
)
```

Figure 1: A basic group

Or a group can just be used in the definition of something else:

```
person = {(
  age: int,
  name: tstr,
  employer: tstr,
)}
```

Figure 2: Using a group in a map

which, given the above rule for pii, is identical to:

```
person = {
  pii
}
```

Figure 3: Using a group by name

Note that the (curly) braces signify the creation of a map; the groups themselves are neutral as to whether they will be used in a map or an array.

The parentheses for groups are optional when there is some other set of brackets present, so it would be slightly more natural to express Figure 2 as:

```
person = {
  age: int,
  name: tstr,
  employer: tstr,
}
```

Groups can be used to factor out common parts of structs, e.g., instead of writing:

```
person = {
  age: int,
  name: tstr,
  employer: tstr,
}

dog = {
  age: int,
  name: tstr,
  leash-length: float,
}
```

one can choose a name for the common subgroup and write:

```
person = {
  identity,
  employer: tstr,
}

dog = {
  identity,
  leash-length: float,
}

identity = (
  age: int,
  name: tstr,
)
```

Figure 4: Using a group for factorization

Note that the contents of the braces in the above definitions constitute (anonymous) groups, while "identity" is a named group.

2.1.1. Usage

Groups are the instrument used in composing data structures with CDDL. It is a matter of style in defining those structures whether to define groups (anonymously) right in their contexts or whether to define them in a separate rule and to reference them with their respective name (possibly more than once).

With this, one is allowed to define all small parts of their data structures and compose bigger protocol units with those or to have only one big protocol data unit that has all definitions ad hoc where needed.

2.1.2. Syntax

The composition syntax intends to be concise and easy to read:

- The start of a group can be marked by '('
- The end of a group can be marked by ')'
- Definitions of entries inside of a group are noted as follows: _keytype => valuetype,_ (read "keytype maps to valuetype"). The comma is actually optional (not just in the final entry), but it is considered good style to set it. The double arrow can be replaced by a colon in the common case of directly using a text string as a key (see Section 3.5.1).

An entry consists of a _keytype_ and a _valuetype_:

- _keytype_ is either an atom used as the actual key or a type in general. The latter case may be needed when using groups in a table context, where the actual keys are of lesser importance than the key types, e.g in contexts verifying incoming data.
- _valuetype_ is a type, which could be derived from the major types defined in [RFC7049], could be a convenience valuetype defined in this document (Appendix E) or the name of a type defined in the specification.

A group definition can also contain choices between groups, see Section 2.2.2.

2.2. Types

2.2.1. Values

Values such as numbers and strings can be used in place of a type. (For instance, this is a very common thing to do for a keytype, common enough that CDDL provides additional convenience syntax for this.)

2.2.2. Choices

Many places that allow a type also allow a choice between types, delimited by a "/" (slash). The entire choice construct can be put into parentheses if this is required to make the construction unambiguous (please see Appendix D for the details).

Choices of values can be used to express enumerations:

```
attire = "bow tie" / "necktie" / "Internet attire"
protocol = 6 / 17
```

Similarly as for types, CDDL also allows choices between groups, delimited by a "//" (double slash).

```
address = { delivery }

delivery = (
street: tstr, ? number: uint, city //
po-box: uint, city //
per-pickup: true )

city = (
name: tstr, zip-code: uint
)
```

Both for type choices and for group choices, additional alternatives can be added to a rule later in separate rules by using "/=" and "//=", respectively, instead of "=":

```
attire /= "swimwear"

delivery //= (
lat: float, long: float, drone-type: tstr
)
```

It is not an error if a name is first used with a "/=" or "//=" (there is no need to "create it" with "=").

2.2.2.1. Ranges

Instead of naming all the values that make up a choice, CDDL allows building a _range_ out of two values that are in an ordering relationship. A range can be inclusive of both ends given (denoted by joining two values by ".."), or include the first and exclude the second (denoted by instead using "...").

```
device-address = byte
max-byte = 255
byte = 0..max-byte ; inclusive range
first-non-byte = 256
byte1 = 0...first-non-byte ; byte1 is equivalent to byte
```

CDDL currently only allows ranges between numbers [_range].

2.2.2.2. Turning a group into a choice

Some choices are built out of large numbers of values, often integers, each of which is best given a semantic name in the specification. Instead of naming each of these integers and then accumulating these into a choice, CDDL allows building a choice from a group by prefixing it with a "&" character:

```
terminal-color = &basecolors
basecolors = (
  black: 0, red: 1,  green: 2,  yellow: 3,
  blue: 4,  magenta: 5,  cyan: 6,  white: 7,
)
extended-color = &(
  basecolors,
  orange: 8,  pink: 9,  purple: 10,  brown: 11,
)
```

As with the use of groups in arrays (Section 3.4), the membernames have only documentary value (in particular, they might be used by a tool when displaying integers that are taken from that choice).

2.2.3. Representation Types

CDDL allows the specification of a data item type by referring to the CBOR representation (major and minor numbers). How this is used should be evident from the prelude (Appendix E).

It may be necessary to make use of representation types outside the prelude, e.g., a specification could start by making use of an existing tag in a more specific way, or define a new tag not defined in the prelude:

```
my_breakfast = #6.55799(breakfast)   ; cbor-any is too general!
breakfast = cereal / porridge
cereal = #6.998(tstr)
porridge = #6.999([liquid, solid])
liquid = milk / water
milk = 0
water = 1
solid = tstr
```

2.2.4. Root type

There is no special syntax to identify the root of a CDDL data structure definition: that role is simply taken by the first rule defined in the file.

This is motivated by the usual top-down approach for defining data structures, decomposing a big data structure unit into smaller parts; however, except for the root type, there is no need to strictly follow this sequence.

(Note that there is no way to use a group as a root - it must be a type. Using a group as the root might be employed as a way to specify a CBOR sequence in a future version of this specification; this would act as if that group is used in an array and the data items in that fictional array form the members of the CBOR sequence.)

3. Syntax

In this section, the overall syntax of CDDL is shown, alongside some examples just illustrating syntax. (The definition will not attempt to be overly formal; refer to Appendix D for the details.)

3.1. General conventions

The basic syntax is inspired by ABNF [RFC5234], with

- rules, whether they define groups or types, are defined with a name, followed by an equals sign "=" and the actual definition according to the respective syntactic rules of that definition.

- A name can consist of any of the characters from the set {'A', ..., 'Z', 'a', ..., 'z', '0', ..., '9', '_', '-', '@', '.', '$'}, starting with an alphabetic character (including '@', '_', '$') and ending in one or a digit.

  * Names are case sensitive.

  * It is preferred style to start a name with a lower case letter.

  * The hyphen is preferred over the underscore (except in a "bareword" (Section 3.5.1), where the semantics may actually require an underscore).

  * The period may be useful for larger specifications, to express some module structure (as in "tcp.throughput" vs. "udp.throughput").

  * A number of names are predefined in the CDDL prelude, as listed in Appendix E.

  * Rule names (types or groups) do not appear in the actual CBOR encoding, but names used as "barewords" in member keys do.

- Comments are started by a ';' (semicolon) character and finish at the end of a line (LF or CRLF).

- outside strings, whitespace (spaces, newlines, and comments) is used to separate syntactic elements for readability (and to separate identifiers or numbers that follow each other); it is otherwise completely optional.

- Hexadecimal numbers are preceded by '0x' (without quotes, lower case x), and are case insensitive. Similarly, binary numbers are preceded by '0b'.

- Text strings are enclosed by double quotation '"' characters. They follow the conventions for strings as defined in section 7 of [RFC7159]. (ABNF users may want to note that there is no support in CDDL for the concept of case insensitivity in text strings; if necessary, regular expressions can be used (Section 3.8.3).)

- Byte strings are enclosed by single quotation "'" characters and may be prefixed by "h" or "b64". If unprefixed, the string is interpreted as with a text string, except that single quotes must be escaped and that the UTF-8 bytes resulting are marked as a byte string (major type 2). If prefixed as "h" or "b64", the string is interpreted as a sequence of hex digits or a base64(url) string, respectively (as with the diagnostic notation in section 6 of [RFC7049]; cf. Appendix G.2); any white space present within the string (including comments) is ignored in the prefixed case. [_strings]

- CDDL uses UTF-8 [RFC3629] for its encoding.

Example:

```
; This is a comment
person = { g }

g = (
  "name": tstr,
  age: int,  ; "age" is a bareword
)
```

3.2. Occurrence

An optional _occurrence_ indicator can be given in front of a group entry. It is either one of the characters '?' (optional), '*' (zero or more), or '+' (one or more), or is of the form n*m, where n and m are optional unsigned integers and n is the lower limit (default 0) and m is the upper limit (default no limit) of occurrences.

If no occurrence indicator is specified, the group entry is to occur exactly once (as if 1*1 were specified).

Note that CDDL, outside any directives/annotations that could possibly be defined, does not make any prescription as to whether arrays or maps use the definite length or indefinite length encoding. I.e., there is no correlation between leaving the size of an array "open" in the spec and the fact that it is then interchanged with definite or indefinite length.

Please also note that CDDL can describe flexibility that the data model of the target representation does not have. This is rather obvious for JSON, but also is relevant for CBOR:

```
apartment = {
  kitchen: size,
  * bedroom: size,
}
size = float ; in m2
```

The previous specification does not mean that CBOR is changed to allow to use the key "bedroom" more than once. In other words, due to the restrictions imposed by the data model, the third line pretty much turns into:

```
  ? bedroom: size,
```

(Occurrence indicators beyond one still are useful in maps for groups that allow a variety of keys.)

3.3. Predefined names for types

CDDL predefines a number of names. This subsection summarizes these names, but please see Appendix E for the exact definitions.

The following keywords for primitive datatypes are defined:

"bool" Boolean value (major type 7, additional information 20 or 21).

"uint" An unsigned integer (major type 0).

"nint" A negative integer (major type 1).

"int" An unsigned integer or a negative integer.

"float16" IEEE 754 half-precision float (major type 7, additional information 25).

"float32" IEEE 754 single-precision float (major type 7, additional information 26).

"float64" IEEE 754 double-precision float (major type 7, additional information 27).

"float" One of float16, float32, or float64.

"bstr" or "bytes" A byte string (major type 2).

"tstr" or "text" Text string (major type 3)

(Note that there are no predefined names for arrays or maps; these are defined with the syntax given below.)

In addition, a number of types are defined in the prelude that are associated with CBOR tags, such as "tdate", "bigint", "regexp" etc.

3.4. Arrays

Array definitions surround a group with square brackets.

For each entry, an occurrence indicator as specified in Section 3.2 is permitted.

For example:

```
unlimited-people = [* person]
one-or-two-people = [1*2 person]
at-least-two-people = [2* person]
person = (
    name: tstr,
    age: uint,
)
```

The group "person" is defined in such a way that repeating it in the array each time generates alternating names and ages, so these are four valid values for a data item of type "unlimited-people":

```
["roundlet", 1047, "psychurgy", 2204, "extrarhythmical", 2231]
[]
["aluminize", 212, "climograph", 4124]
["penintime", 1513, "endocarditis", 4084, "impermeator", 1669,
 "coextension", 865]
```

3.5. Maps

The syntax for specifying maps merits special attention, as well as a number of optimizations and conveniences, as it is likely to be the focal point of many specifications employing CDDL. While the syntax does not strictly distinguish struct and table usage of maps, it caters specifically to each of them.

3.5.1. Structs

The "struct" usage of maps is similar to the way JSON objects are used in many JSON applications.

A map is defined in the same way as defining an array (see Section 3.4), except for using curly braces "{}" instead of square brackets "[]".

An occurrence indicator as specified in Section 3.2 is permitted for each group entry.

The following is an example of a structure:

```
Geography = [
  city           : tstr,
  gpsCoordinates : GpsCoordinates,
]

GpsCoordinates = {
  longitude      : uint,            ; multiplied by 10^7
  latitude       : uint,            ; multiplied by 10^7
}
```

When encoding, the Geography structure is encoded using a CBOR array with two entries (the keys for the group entries are ignored), whereas the GpsCoordinates are encoded as a CBOR map with two key/value pairs.

Types used in a structure can be defined in separate rules or just in place (potentially placed inside parentheses, such as for choices). E.g.:

```
located-samples = {
  sample-point: int,
  samples: [+ float],
}
```

where "located-samples" is the datatype to be used when referring to the struct, and "sample-point" and "samples" are the keys to be used. This is actually a complete example: an identifier that is followed by a colon can be directly used as the text string for a member key (we speak of a "bareword" member key), as can a double-quoted string or a number. (When other types, in particular multi-valued ones, are used as keytypes, they are followed by a double arrow, see below.)

If a text string key does not match the syntax for an identifier (or if the specifier just happens to prefer using double quotes), the text string syntax can also be used in the member key position, followed by a colon. The above example could therefore have been written with quoted strings in the member key positions.

All the types defined can be used in a keytype position by following them with a double arrow. A string also is a (single-valued) type, so another form for this example is:

```
located-samples = {
  "sample-point" => int,
  "samples" => [+ float],
}
```

A better way to demonstrate the double-arrow use may be:

```
located-samples = {
  sample-point: int,
  samples: [+ float],
  * equipment-type => equipment-tolerances,
}
equipment-type = [name: tstr, manufacturer: tstr]
equipment-tolerances = [+ [float, float]]
```

The example below defines a struct with optional entries: display name (as a text string), the name components first name and family name (as a map of text strings), and age information (as an unsigned integer).

```
PersonalData = {
  ? displayName: tstr,
  NameComponents,
  ? age: uint,
}

NameComponents = (
  ? firstName: tstr,
  ? familyName: tstr,
)
```

Note that the group definition for NameComponents does not generate another map; instead, all four keys are directly in the struct built by PersonalData.

In this example, all key/value pairs are optional from the perspective of CDDL. With no occurrence indicator, an entry is mandatory.

If the addition of more entries not specified by the current specification is desired, one can add this possibility explicitly:

```
PersonalData = {
  ? displayName: tstr,
  NameComponents,
  ? age: uint,
  * tstr => any
}

NameComponents = (
  ? firstName: tstr,
  ? familyName: tstr,
)
```

Figure 5: Personal Data: Example for extensibility

The cddl tool (Appendix F) generated as one acceptable instance for this specification:

```
{"familyName": "agust", "antiforeignism": "pretzel",
 "springbuck": "illuminatingly", "exuviae": "ephemeris",
 "kilometrage": "frogfish"}
```

(See Section 3.9 for one way to explicitly identify an extension point.)

3.5.2. Tables

A table can be specified by defining a map with entries where the keytype is not single-valued, e.g.:

```
square-roots = {* x => y}
x = int
y = float
```

Here, the key in each key/value pair has datatype x (defined as int), and the value has datatype y (defined as float).

If the specification does not need to restrict one of x or y (i.e., the application is free to choose per entry), it can be replaced by the predefined name "any".

As another example, the following could be used as a conversion table converting from an integer or float to a string:

```
tostring = {* mynumber => tstr}
mynumber = int / float
```

3.6. Tags

A type can make use of a CBOR tag (major type 6) by using the representation type notation, giving #6.nnn(type) where nnn is an unsigned integer giving the tag number and "type" is the type of the data item being tagged.

For example, the following line from the CDDL prelude (Appendix E) defines "biguint" as a type name for a positive bignum N:

```
biguint = #6.2(bstr)
```

The tags defined by [RFC7049] are included in the prelude. Additional tags since registered need to be added to a CDDL specification as needed; e.g., a binary UUID tag could be referenced as "buuid" in a specification after defining

```
buuid = #6.37(bstr)
```

In the following example, usage of the tag 32 for URIs is optional:

```
my_uri = #6.32(tstr) / tstr
```

3.7. Unwrapping

The group that is used to define a map or an array can often be reused in the definition of another map or array. Similarly, a type defined as a tag carries an internal data item that one would like to refer to. In these cases, it is expedient to simply use the name of the map, array, or tag type as a handle for the group or type defined inside it.

The "unwrap" operator (written by preceding a name by a tilde character "~") can be used to strip the type defined for a name by one layer, exposing the underlying group (for maps and arrays) or type (for tags).

For example, an application might want to define a basic and an advanced header. Without unwrapping, this might be done as follows:

```
basic-header-group = (
  field1: int,
  field2: text,
)

basic-header = { basic-header-group }

advanced-header = {
  basic-header-group,
  field3: bytes,
  field4: number, ; as in the tagged type "time"
}
```

Unwrapping simplifies this to:

```
basic-header = {
  field1: int,
  field2: text,
}

advanced-header = {
  ~basic-header,
  field3: bytes,
  field4: ~time,
}
```

(Note that leaving out the first unwrap operator in the latter example would lead to nesting the basic-header in its own map inside the advanced-header, while, with the unwrapped basic-header, the definition of the group inside basic-header is essentially repeated inside advanced-header, leading to a single map. This can be used for various applications often solved by inheritance in programming languages. The effect of unwrapping can also be described as "threading in" the group or type inside the referenced type, which suggested the thread-like "~" character.)

3.8. Controls

A _control_ allows to relate a _target_ type with a _controller_ type via a _control operator_.

The syntax for a control type is "target .control-operator controller", where control operators are special identifiers prefixed by a dot. (Note that _target_ or _controller_ might need to be parenthesized.)

A number of control operators are defined at his point. Note that the CDDL tool does not currently support combining multiple controls on a single target.

3.8.1. Control operator .size

A ".size" control controls the size of the target in bytes by the control type. Examples:

```
full-address = [[+ label], ip4, ip6]
ip4 = bstr .size 4
ip6 = bstr .size 16
label = bstr .size (1..63)
```

Figure 6: Control for size in bytes

When applied to an unsigned integer, the ".size" control restricts the range of that integer by giving a maximum number of bytes that should be needed in a computer representation of that unsigned integer. In other words, "uint .size N" is equivalent to "0...BYTES_N", where BYTES_N == 256**N.

```
audio_sample = uint .size 3 ; 24-bit, equivalent to 0..16777215
```

Figure 7: Control for integer size in bytes

Note that, as with value restrictions in CDDL, this control is not a representation constraint; a number that fits into fewer bytes can still be represented in that form, and an inefficient implementation could use a longer form (unless that is restricted by some format constraints outside of CDDL, such as the rules in Section 3.9 of [RFC7049]).

3.8.2. Control operator .bits

A ".bits" control on a byte string indicates that, in the target, only the bits numbered by a number in the control type are allowed to be set. (Bits are counted the usual way, bit number "n" being set in "str" meaning that "(str[n >> 3] & (1 << (n & 7))) != 0".) [_bitsendian]

Similarly, a ".bits" control on an unsigned integer "i" indicates that for all unsigned integers "n" where "(i & (1 << n)) != 0", "n" must be in the control type.

```
tcpflagbytes = bstr .bits flags
flags = &(
  fin: 8,
  syn: 9,
  rst: 10,
  psh: 11,
  ack: 12,
  urg: 13,
  ece: 14,
  cwr: 15,
  ns: 0,
) / (4..7) ; data offset bits

rwxbits = uint .bits rwx
rwx = &(r: 2, w: 1, x: 0)
```

Figure 8: Control for what bits can be set

The CDDL tool generates the following ten example instances for "tcpflagbytes":

h'906d' h'01fc' h'8145' h'01b7' h'013d' h'409f' h'018e' h'c05f' h'01fa' h'01fe'

These examples do not illustrate that the above CDDL specification does not explicitly specify a size of two bytes: A valid all clear instance of flag bytes could be "h''" or "h'00'" or even "h'000000'" as well.

3.8.3. Control operator .regexp

A ".regexp" control indicates that the text string given as a target needs to match the PCRE regular expression given as a value in the control type, where that regular expression is anchored on both sides. (If anchoring is not desired for a side, ".*" needs to be inserted there.)

```
nai = tstr .regexp "\\w+@\\w+(\\.\\w+)+"
```

Figure 9: Control with a PCRE regexp

The CDDL tool proposes:

"N1@CH57HF.4Znqe0.dYJRN.igjf"

3.8.4. Control operators .cbor and .cborseq

A ".cbor" control on a byte string indicates that the byte string carries a CBOR encoded data item. Decoded, the data item matches the type given as the right-hand side argument (type1 in the following example).

```
"bytes .cbor type1"
```

Similarly, a ".cborseq" control on a byte string indicates that the byte string carries a sequence of CBOR encoded data items. When the data items are taken as an array, the array matches the type given as the right-hand side argument (type2 in the following example).

```
"bytes .cborseq type2"
```

(The conversion of the encoded sequence to an array can be effected for instance by wrapping the byte string between the two bytes 0x9f and 0xff and decoding the wrapped byte string as a CBOR encoded data item.)

3.8.5. Control operators .within and .and

A ".and" control on a type indicates that the data item matches both that left hand side type and the type given as the right hand side. (Formally, the resulting type is the intersection of the two types given.)

```
"type1 .and type2"
```

A variant of the ".and" control is the ".within" control, which expresses an additional intent: the left hand side type is meant to be a subset of the right-hand-side type.

```
"type1 .within type2"
```

While both forms have the identical formal semantics (intersection), the intention of the ".within" form is that the right hand side gives guidance to the types allowed on the left hand side, which typically is a socket (Section 3.9):

```
message = $message .within message-structure
message-structure = [message_type, *message_option]
message_type = 0..255
message_option = any

$message /= [3, dough: text, topping: [* text]]
$message /= [4, noodles: text, sauce: text, parmesan: bool]
```

For ".within", a tool might flag an error if type1 allows data items that are not allowed by type2. In contrast, for ".and", there is no expectation that type1 already is a subset of type2.

3.8.6. Control operators .lt, .le, .gt, .ge, .eq, .ne, and .default

The controls .lt, .le, .gt, .ge, .eq, .ne specify a constraint on the left hand side type to be a value less than, less than or equal, equal to, not equal to, greather than, or greater than or equal to a value given as a (single-valued) right hand side type. In the present specification, the first four controls (.lt, .le, .gt, .ge) are defined only for numeric types, as these have a natural ordering relationship.

```
speed = number .ge 0  ; unit: m/s
```

A variant of the ".ne" control is the ".default" control, which expresses an additional intent: the value specified by the right-hand-side type is intended as a default value for the left hand side type given, and the implied .ne control is there to prevent this value from being sent over the wire. This control is only meaningful when the controld type is used in an optional context; otherwise there would be no way to express the default value.

```
timer = {
  time: uint,
  ? displayed-step: (number .gt 0) .default 1
}
```

3.9. Socket/Plug

Both for type choices and group choices, a mechanism is defined that facilitates starting out with empty choices and assembling them later, potentially in separate files that are concatenated to build the full specification.

Per convention, CDDL extension points are marked with a leading dollar sign (types) or two leading dollar signs (groups). Tools honor that convention by not raising an error if such a type or group is not defined at all; the symbol is then taken to be an empty type choice (group choice), i.e., no choice is available.

```
tcp-header = {seq: uint, ack: uint, * $$tcp-option}

; later, in a different file

$$tcp-option //= (
sack: [+(left: uint, right: uint)]
)

; and, maybe in another file

$$tcp-option //= (
sack-permitted: true
)
```

Names that start with a single "$" are "type sockets", names with a double "$$" are "group sockets". It is not an error if there is no definition for a socket at all; this then means there is no way to satisfy the rule (i.e., the choice is empty).

All definitions (plugs) for socket names must be augmentations, i.e., they must be using "/=" and "//=", respectively.

To pick up the example illustrated in Figure 5, the socket/plug mechanism could be used as shown in Figure 10:

```
PersonalData = {
  ? displayName: tstr,
  NameComponents,
  ? age: uint,
  * $$personaldata-extensions
}

NameComponents = (
  ? firstName: tstr,
  ? familyName: tstr,
)

; The above already works as is.
; But then, we can add later:

$$personaldata-extensions //= (
  favorite-salsa: tstr,
)

; and again, somewhere else:

$$personaldata-extensions //= (
  shoesize: uint,
)
```

Figure 10: Personal Data example: Using socket/plug extensibility

3.10. Generics

Using angle brackets, the left hand side of a rule can add formal parameters after the name being defined, as in:

```
messages = message<"reboot", "now"> / message<"sleep", 1..100>
message<t, v> = {type: t, value: v}
```

When using a generic rule, the formal parameters are bound to the actual arguments supplied (also using angle brackets), within the scope of the generic rule (as if there were a rule of the form parameter = argument).

(There are some limitations to nesting of generics in Appendix F at this time.)

3.11. Operator Precedence

As with any language that has multiple syntactic features such as prefix and infix operators, CDDL has operators that bind more tightly than others. This is becoming more complicated than, say, in ABNF,

as CDDL has both types and groups, with operators that are specific to these concepts. Type operators (such as "/" for type choice) operate on types, while group operators (such as "//" for group choice) operate on groups. Types can simply be used in groups, but groups need to be bracketed (as arrays or maps) to become types. So, type operators naturally bind closer than group operators.

For instance, in

```
t = [group1]
group1 = (a / b // c / d)
a = 1 b = 2 c = 3 d = 4
```

group1 is a group choice between the type choice of a and b and the type choice of c and d. This becomes more relevant once member keys and/or occurrences are added in:

```
t = {group2}
group2 = (? ab: a / b // cd: c / d)
a = 1 b = 2 c = 3 d = 4
```

is a group choice between the optional member "ab" of type a or b and the member "cd" of type c or d. Note that the optionality is attached to the first choice ("ab"), not to the second choice.

Similarly, in

```
t = [group3]
group3 = (+ a / b / c)
a = 1 b = 2 c = 3
```

group3 is a repetition of a type choice between a, b, and c [unflex]; if just a is to be repeatable, a group choice is needed to focus the occurrence:

```
t = [group4]
group4 = (+ a // b / c)
a = 1 b = 2 c = 3
```

group4 is a group choice between a repeatable a and a single b or c.

In general, as with many other languages with operator precedence rules, it is best not to rely on them, but to insert parentheses for readability:

```
t = [group4a]
group4a = ((+ a) // (b / c))
a = 1 b = 2 c = 3
```

The operator precedences, in sequence of loose to tight binding, are defined in Appendix D and summarized in Table 1. (Arities given are 1 for unary prefix operators and 2 for binary infix operators.)

Operator	Ar	Operates on	Prec
=	2	name = type, name = group	1
/=	2	name /= type	1
//=	2	name //= group	1
//	2	group // group	2
,	2	group, group	3
*	1	* group	4
N*M	1	N*M group	4
+	1	+ group	4
?	1	? group	4
=>	2	type => type	5
:	2	name: type	5
/	2	type / type	6
&	1	&group	6
..	2	type..type	7
...	2	type...type	7
.anno	2	type .anno type	7

Table 1: Summary of operator precedences

4. Making Use of CDDL

In this section, we discuss several potential ways to employ CDDL.

4.1. As a guide to a human user

CDDL can be used to efficiently define the layout of CBOR data, such that a human implementer can easily see how data is supposed to be encoded.

Since CDDL maps parts of the CBOR data to human readable names, tools could be built that use CDDL to provide a human friendly representation of the CBOR data, and allow them to edit such data while remaining compliant to its CDDL definition.

4.2. For automated checking of CBOR data structure

CDDL has been specified such that a machine can handle the CDDL definition and related CBOR data (and, thus, also JSON data). For example, a machine could use CDDL to check whether or not CBOR data is compliant to its definition.

The need for thoroughness of such compliance checking depends on the application. For example, an application may decide not to check the data structure at all, and use the CDDL definition solely as a means to indicate the structure of the data to the programmer.

On the other end, the application may also implement a checking mechanism that goes as far as checking that all mandatory map members are available.

The matter in how far the data description must be enforced by an application is left to the designers and implementers of that application, keeping in mind related security considerations.

In no case the intention is that a CDDL tool would be "writing code" for an implementation.

4.3. For data analysis tools

In the long run, it can be expected that more and more data will be stored using the CBOR data format.

Where there is data, there is data analysis and the need to process such data automatically. CDDL can be used for such automated data processing, allowing tools to verify data, clean it, and extract particular parts of interest from it.

Since CBOR is designed with constrained devices in mind, a likely use of it would be small sensors. An interesting use would thus be automated analysis of sensor data.

5. Security considerations

This document presents a content rules language for expressing CBOR data structures. As such, it does not bring any security issues on itself, although specification of protocols that use CBOR naturally need security analysis when defined.

Topics that could be considered in a security considerations section that uses CDDL to define CBOR structures include the following:

- Where could the language maybe cause confusion in a way that will enable security issues?

6. IANA considerations

This document does not require any IANA registrations.

7. Acknowledgements

CDDL was originally conceived by Bert Greevenbosch, who also wrote the original five versions of this document.

Inspiration was taken from the C and Pascal languages, MPEG's conventions for describing structures in the ISO base media file format, Relax-NG and its compact syntax [RELAXNG], and in particular from Andrew Lee Newton's "JSON Content Rules" [I-D.newton-json-content-rules].

Useful feedback came from Joe Hildebrand, Sean Leonard and Jim Schaad.

The CDDL tool was written by Carsten Bormann, building on previous work by Troy Heninger and Tom Lord.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <http://www.rfc-editor.org/info/rfc3629>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <http://www.rfc-editor.org/info/rfc7049>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <http://www.rfc-editor.org/info/rfc7493>.

8.2. Informative References

[I-D.ietf-anima-grasp]
Bormann, C., Carpenter, B., and B. Liu, "A Generic Autonomic Signaling Protocol (GRASP)", draft-ietf-anima-grasp-14 (work in progress), July 2017.

[I-D.ietf-core-senml]
Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Media Types for Sensor Measurement Lists (SenML)", draft-ietf-core-senml-10 (work in progress), July 2017.

[I-D.ietf-cose-msg]
Schaad, J., "CBOR Object Signing and Encryption (COSE)", draft-ietf-cose-msg-24 (work in progress), November 2016.

[I-D.newton-json-content-rules]
Newton, A. and P. Cordell, "A Language for Rules Describing JSON Content", draft-newton-json-content-rules-08 (work in progress), March 2017.

[RELAXNG] OASIS, "RELAX-NG Compact Syntax", November 2002, <http://relaxng.org/compact-20021121.html>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <http://www.rfc-editor.org/info/rfc4648>.

[RFC7071] Borenstein, N. and M. Kucherawy, "A Media Type for Reputation Interchange", RFC 7071, DOI 10.17487/RFC7071, November 2013, <http://www.rfc-editor.org/info/rfc7071>.

[RFC8007] Murray, R. and B. Niven-Jenkins, "Content Delivery Network Interconnection (CDNI) Control Interface / Triggers", RFC 8007, DOI 10.17487/RFC8007, December 2016, <http://www.rfc-editor.org/info/rfc8007>.

Appendix A. Cemetery

The following ideas have been buried in the discussions leading up to the present specification:

- <...> as syntax for enumerations. We view values to be just another type (a very specific type with just one member), so that an enumeration can be denoted as a choice using "/" as the delimiter of choices. Because of this, no evidence is present that a separate syntax for enumerations is needed.

A.1. Resolved Issues

- The key/value pairs in maps have no fixed ordering. One could imagine situations where fixing the ordering may be of use. For example, a decoder could look for values related with integer keys 1, 3 and 7. If the order were fixed and the decoder encounters the key 4 without having encountered key 3, it could conclude that key 3 is not available without doing more complicated bookkeeping. Unfortunately, neither JSON nor CBOR support this, so no attempt was made to support this in CDDL either.

- CDDL distinguishes the various CBOR number types, but there is only one number type in JSON. There is no effect in specifying a precision (float16/float32/float64) when using CDDL for specifying JSON data structures. (The current validator implementation Appendix F does not handle this very well, either.)

Appendix B. (Not used.)

Appendix C. Change Log

Changes from version 00 to version 01:

- Removed constants
- Updated the tag mechanism
- Extended the map structure
- Added examples

Changes from version 01 to version 02:

- Fixed example

Changes from version 02 to version 03:

- Added information about characters used in names
- Added text about an overall data structure and order of definition of fields
- Added text about encoding of keys
- Added table with keywords
- Strings and integer writing conventions

- Added ABNF

Changes from version 03 to version 04:

- Removed optional fields for non-maps
- Defined all key/value pairs in maps are considered optional from the CDDL perspective
- Allow omission of type of keys for maps with only text string and integer keys
- Changed order of definitions
- Updated fruit and moves examples
- Renamed the "Philosophy" section to "Using CDDL", and added more text about CDDL usage
- Several editorials

Changes from version 04 to version 05:

- Added text about alternative datatypes and any datatype
- Fixed typos
- Restructured syntax and semantics

Changes from version 05 to version 05:

- Fixed the ABNF for choices (no longer need to write a: (b/c))
- Added group choices (//)
- Added /= and //=
- Added experimental socket/plug
- Added aliases text, bytes, null to prelude
- Documented generics
- Fixed more typos

Changes from 06 to 07:

- .cbor, .cborseq, .within, .and

- Define .size on uint
- Extended Diagnostic Notation
- Precedence discussion and table
- Remove some of the "issues" that can only be understood with historical context
- Prefer "text" over "tstr" in some of the examples
- Add "unsigned" to the prelude

Changes from 07 to 08:

- .lt, .le, .eq, .ne, .gt, .ge
- .default

Changes from 08 to 09:

- Take annotations and socket/plug out of the nursery; they have been battle-proven enough.
- Define a value notation for byte strings as well.
- Removed discussion section that was no longer relevant; move "Resolved Issues" to appendix.

Changes from 09 to 10:

- Remove a long but not very elucidating example. (Maybe we'll add back some shorter examples later.)
- A few clarifications.
- Updated author list.

Changes from 10 to 11:

- Define unwrapping operator ~
- Change term for annotation into "control" (but leave "annotate" for when it actually is meant in that sense)

Appendix D. ABNF grammar

The following is a formal definition of the CDDL syntax in Augmented Backus-Naur Form (ABNF, [RFC5234]). [_abnftodo]

```
cddl = S 1*rule
rule = typename [genericparm] S assign S type S
     / groupname [genericparm] S assign S grpent S

typename = id
groupname = id

assign = "=" / "/=" / "//="

genericparm = "<" S id S *("," S id S ) ">"
genericarg = "<" S type1 S *("," S type1 S ) ">"

type = type1 S *("/" S type1 S)

type1 = type2 [S (rangeop / ctlop) S type2]

type2 = value
      / typename [genericarg]
      / "(" type ")"
      / "~" S groupname [genericarg]
      / "#" "6" ["." uint] "(" S type S ")" ; note no space!
      / "#" DIGIT ["." uint]                ; major/ai
      / "#"                                 ; any
      / "{" S group S "}"
      / "[" S group S "]"
      / "&" S "(" S group S ")"
      / "&" S groupname [genericarg]

rangeop = "..." / ".."

ctlop = "." id

group = grpchoice S *("//" S grpchoice S)

grpchoice = *grpent

grpent = [occur S] [memberkey S] type optcom
       / [occur S] groupname [genericarg] optcom ; preempted by above
       / [occur S] "(" S group S ")" optcom

memberkey = type1 S "=>"
          / bareword S ":"
          / value S ":"
```

```
   bareword = id

   optcom = S ["," S]

   occur = [uint] "*" [uint]
         / "+"
         / "?"

   uint = ["0x" / "0b"] "0"
        / DIGIT1 *DIGIT
        / "0x" 1*HEXDIG
        / "0b" 1*BINDIG

   value = number
         / text
         / bytes

   int = ["-"] uint

   ; This is a float if it has fraction or exponent; int otherwise
   number = int ["." fraction] ["e" exponent ]
   fraction = 1*DIGIT
   exponent = int

   text = %x22 *SCHAR %x22
   SCHAR = %x20-21 / %x23-5B / %x5D-10FFFD / SESC
   SESC = "\" %x20-10FFFD

   bytes = [bsqual] %x27 *BCHAR %x27
   BCHAR = %x20-26 / %x28-5B / %x5D-10FFFD / SESC / CRLF
   bsqual = %x68 ; "h"
          / %x62.36.34 ; "b64"

   id = EALPHA *(*("-" / ".") (EALPHA / DIGIT))
   ALPHA = %x41-5A / %x61-7A
   EALPHA = %x41-5A / %x61-7A / "@" / "_" / "$"
   DIGIT = %x30-39
   DIGIT1 = %x31-39
   HEXDIG = DIGIT / "A" / "B" / "C" / "D" / "E" / "F"
   BINDIG = %x30-31

   S = *WS
   WS = SP / NL
   SP = %x20
   NL = COMMENT / CRLF
   COMMENT = ";" *PCHAR CRLF
   PCHAR = %x20-10FFFD
   CRLF = %x0A / %x0D.0A
```

Figure 11: CDDL ABNF

Appendix E. Standard Prelude

The following prelude is automatically added to each CDDL file [tdate]. (Note that technically, it is a postlude, as it does not disturb the selection of the first rule as the root of the definition.)

```
any = #

uint = #0
nint = #1
int = uint / nint

bstr = #2
bytes = bstr
tstr = #3
text = tstr

tdate = #6.0(tstr)
time = #6.1(number)
number = int / float
biguint = #6.2(bstr)
bignint = #6.3(bstr)
bigint = biguint / bignint
integer = int / bigint
unsigned = uint / biguint
decfrac = #6.4([e10: int, m: integer])
bigfloat = #6.5([e2: int, m: integer])
eb64url = #6.21(any)
eb64legacy = #6.22(any)
eb16 = #6.23(any)
encoded-cbor = #6.24(bstr)
uri = #6.32(tstr)
b64url = #6.33(tstr)
b64legacy = #6.34(tstr)
regexp = #6.35(tstr)
mime-message = #6.36(tstr)
cbor-any = #6.55799(any)

float16 = #7.25
float32 = #7.26
float64 = #7.27
float16-32 = float16 / float32
float32-64 = float32 / float64
float = float16-32 / float64

false = #7.20
true = #7.21
bool = false / true
nil = #7.22
null = nil
undefined = #7.23
```

Figure 12: CDDL Prelude

Note that the prelude is deemed to be fixed. This means, for instance, that additional tags beyond [RFC7049], as registered, need to be defined in each CDDL file that is using them.

A common stumbling point is that the prelude does not define a type "string". CBOR has byte strings ("bytes" in the prelude) and text strings ("text"), so a type that is simply called "string" would be ambiguous.

E.1. Use with JSON

The JSON generic data model (implicit in [RFC7159]) is a subset of the generic data model of CBOR. So one can use CDDL with JSON by limiting oneself to what can be represented in JSON. Roughly speaking, this means leaving out byte strings, tags, and simple values other than "false", "true", and "null", leading to the following limited prelude:

```
any = #

uint = #0
nint = #1
int = uint / nint

tstr = #3
text = tstr

number = int / float

float16 = #7.25
float32 = #7.26
float64 = #7.27
float16-32 = float16 / float32
float32-64 = float32 / float64
float = float16-32 / float64

false = #7.20
true = #7.21
bool = false / true
nil = #7.22
null = nil
```

Figure 13: JSON compatible subset of CDDL Prelude

(The major types given here do not have a direct meaning in JSON, but they can be interpreted as CBOR major types translated through Section 4 of [RFC7049].)

There are a few fine points in using CDDL with JSON. First, JSON does not distinguish between integers and floating point numbers; there is only one kind of number (which may happen to be integral). In this context, specifying a type as "uint", "nint" or "int" then becomes a predicate that the number be integral. As an example, this means that the following JSON numbers are all matching "uint":

10 10.0 1e1 1.0e1 100e-1

(The fact that these are all integers may be surprising to users accustomed to the long tradition in programming languages of using decimal points or exponents in a number to indicate a floating point literal.)

Fundamentally, the number system of JSON itself is based on decimal numbers and decimal fractions and does not have limits to its precision or range. In practice, JSON numbers are often parsed into a number type that is called float64 here, creating a number of limitations to the generic data model [RFC7493]. In particular, this means that integers can only be expressed with interoperable exactness when they lie in the range [-(2**53)+1, (2**53)-1] -- a smaller range than that covered by CDDL "int".

JSON applications that want to stay compatible with I-JSON therefore may want to define integer types with more limited ranges, such as in Figure 14. Note that the types given here are not part of the prelude; they need to be copied into the CDDL specification if needed.

```
ij-uint = 0..9007199254740991
ij-nint = -9007199254740991..-1
ij-int = -9007199254740991..9007199254740991
```

Figure 14: I-JSON types for CDDL (not part of prelude)

JSON applications that do not need to stay compatible with I-JSON and that actually may need to go beyond the 64-bit unsigned and negative integers supported by "int" (= "uint"/"nint") may want to use the following additional types from the standard prelude, which are expressed in terms of tags but can straightforwardly be mapped into JSON (but not I-JSON) numbers:

```
biguint = #6.2(bstr)
bignint = #6.3(bstr)
bigint = biguint / bignint
integer = int / bigint
unsigned = uint / biguint
```

CDDL at this point does not have a way to express the unlimited floating point precision that is theoretically possible with JSON; at the time of writing, this is rarely used in protocols in practice.

Note that a data model described in CDDL is always restricted by what can be expressed in the serialization; e.g., floating point values such as NaN (not a number) and the infinities cannot be represented in JSON even if they are allowed in the CDDL generic data model.

Appendix F. The CDDL tool

A rough CDDL tool is available. For CDDL specifications, it can check the syntax, generate one or more instances (expressed in CBOR diagnostic notation or in pretty-printed JSON), and validate an existing instance against the specification:

```
Usage:
cddl spec.cddl generate [n]
cddl spec.cddl json-generate [n]
cddl spec.cddl validate instance.cbor
cddl spec.cddl validate instance.json
```

Figure 15: CDDL tool usage

Install on a system with a modern Ruby via:

```
gem install cddl
```

Figure 16: CDDL tool installation

The accompanying CBOR diagnostic tools (which are automatically installed by the above) are described in https://github.com/cabo/cbor-diag; they can be used to convert between binary CBOR, a pretty-printed form of that, CBOR diagnostic notation, JSON, and YAML.

Appendix G. Extended Diagnostic Notation

Section 6 of [RFC7049] defines a "diagnostic notation" in order to be able to converse about CBOR data items without having to resort to binary data. Diagnostic notation is based on JSON, with extensions for representing CBOR constructs such as binary data and tags.

(Standardizing this together with the actual interchange format does not serve to create another interchange format, but enables the use of a shared diagnostic notation in tools for and documents about CBOR.)

This section discusses a few extensions to the diagnostic notation that have turned out to be useful since RFC 7049 was written. We refer to the result as extended diagnostic notation (EDN).

G.1. White space in byte string notation

Examples often benefit from some white space (spaces, line breaks) in byte strings. In extended diagnostic notation, white space is ignored in prefixed byte strings; for instance, the following are equivalent:

```
h'48656c6c6f20776f726c64'
h'48 65 6c 6c 6f 20 77 6f 72 6c 64'
h'4 86 56c 6c6f
  20776 f726c64'
```

G.2. Text in byte string notation

Diagnostic notation notates Byte strings in one of the [RFC4648] base encodings,, enclosed in single quotes, prefixed by >h< for base16, >b32< for base32, >h32< for base32hex, >b64< for base64 or base64url. Quite often, byte strings carry bytes that are meaningfully interpreted as UTF-8 text. Extended Diagnostic Notation allows the use of single quotes without a prefix to express byte strings with UTF-8 text; for instance, the following are equivalent:

```
'hello world'
h'68656c6c6f20776f726c64'
```

The escaping rules of JSON strings are applied equivalently for text-based byte strings, e.g., \ stands for a single backslash and ' stands for a single quote. White space is included literally, i.e., the previous section does not apply to text-based byte strings.

G.3. Embedded CBOR and CBOR sequences in byte strings

Where a byte string is to carry an embedded CBOR-encoded item, or more generally a sequence of zero or more such items, the diagnostic notation for these zero or more CBOR data items, separated by commata, can be enclosed in << and >> to notate the byte string resulting from encoding the data items and concatenating the result. For instance, each pair of columns in the following are equivalent:

```
<<1>>              h'01'
<<1, 2>>           h'0102'
<<"foo", null>>    h'63666F6FF6'
<<>>               h''
```

G.4. Concatenated Strings

While the ability to include white space enables line-breaking of encoded byte strings, a mechanism is needed to be able to include text strings as well as byte strings in direct UTF-8 representation into line-based documents (such as RFCs and source code).

We extend the diagnostic notation by allowing multiple text strings or multiple byte strings to be notated separated by white space, these are then concatenated into a single text or byte string, respectively. Text strings and byte strings do not mix within such a concatenation, except that byte string notation can be used inside a sequence of concatenated text string notation to encode characters that may be better represented in an encoded way. The following four values are equivalent:

```
"Hello world"
"Hello " "world"
"Hello" h'20' "world"
"" h'48656c6c6f20776f726c64' ""
```

Similarly, the following byte string values are equivalent

```
'Hello world'
'Hello ' 'world'
'Hello ' h'776f726c64'
'Hello' h'20' 'world'
'' h'48656c6c6f20776f726c64' '' b64''
h'4 86 56c 6c6f' h' 20776 f726c64'
```

(Note that the approach of separating by whitespace, while familiar from the C language, requires some attention - a single comma makes a big difference here.)

G.5. Hexadecimal, octal, and binary numbers

In addition to JSON's decimal numbers, EDN provides hexadecimal, octal and binary numbers in the usual C-language notation (octal with 0o prefix present only).

The following are equivalent:

```
4711
0x1267
0o11147
0b1001001100111
```

As are:

```
1.5
0x1.8p0
0x18p-4
```

G.6. Comments

Longer pieces of diagnostic notation may benefit from comments. JSON famously does not provide for comments, and basic RFC 7049 diagnostic notation inherits this property.

In extended diagnostic notation, comments can be included, delimited by slashes ("/"). Any text within and including a pair of slashes is considered a comment.

Comments are considered white space. Hence, they are allowed in prefixed byte strings; for instance, the following are equivalent:

```
h'68656c6c6f20776f726c64'
h'68 65 6c /doubled l!/ 6c 6f /hello/
  20 /space/
  77 6f 72 6c 64' /world/
```

This can be used to annotate a CBOR structure as in:

```
/grasp-message/ [/M_DISCOVERY/ 1, /session-id/ 10584416,
                 /objective/ [/objective-name/ "opsonize",
                              /D, N, S/ 7, /loop-count/ 105]]
```

(There are currently no end-of-line comments. If we want to add them, "//" sounds like a reasonable delimiter given that we already use slashes for comments, but we also could go e.g. for "#".)

Appendix H. Examples

This section contains various examples of structures defined using CDDL.

The theme for the first example is taken from [RFC7071], which defines certain JSON structures in English. For a similar example, it may also be of interest to examine Appendix A of [RFC8007], which contains a CDDL definition for a JSON structure defined in the main body of the RFC.

The second subsection in this appendix translates examples from [I-D.newton-json-content-rules] into CDDL.

These examples all happen to describe data that is interchanged in JSON. Examples for CDDL definitions of data that is interchanged in

CBOR can be found in [I-D.ietf-cose-msg], [I-D.ietf-anima-grasp], or [I-D.ietf-core-senml].

H.1. RFC 7071

[RFC7071] defines the Reputon structure for JSON using somewhat formalized English text. Here is a (somewhat verbose) equivalent definition using the same terms, but notated in CDDL:

```
reputation-object = {
  reputation-context,
  reputon-list
}

reputation-context = (
  application: text
)

reputon-list = (
  reputons: reputon-array
)

reputon-array = [* reputon]

reputon = {
  rater-value,
  assertion-value,
  rated-value,
  rating-value,
  ? conf-value,
  ? normal-value,
  ? sample-value,
  ? gen-value,
  ? expire-value,
  * ext-value,
}

rater-value = ( rater: text )
assertion-value = ( assertion: text )
rated-value = ( rated: text )
rating-value = ( rating: float16 )
conf-value = ( confidence: float16 )
normal-value = ( normal-rating: float16 )
sample-value = ( sample-size: uint )
gen-value = ( generated: uint )
expire-value = ( expires: uint )
ext-value = ( text => any )
```

An equivalent, more compact form of this example would be:

```
reputation-object = {
  application: text
  reputons: [* reputon]
}

reputon = {
  rater: text
  assertion: text
  rated: text
  rating: float16
  ? confidence: float16
  ? normal-rating: float16
  ? sample-size: uint
  ? generated: uint
  ? expires: uint
  * text => any
}
```

Note how this rather clearly delineates the structure somewhat shrouded by so many words in section 6.2.2. of [RFC7071]. Also, this definition makes it clear that several ext-values are allowed (by definition with different member names); RFC 7071 could be read to forbid the repetition of ext-value ("A specific reputon-element MUST NOT appear more than once" is ambiguous.)

The CDDL tool (which hasn't quite been trained for polite conversation) says:

```
{
  "application": "tridentiferous",
  "reputons": [
    {
      "rater": "loamily",
      "assertion": "Dasyprocta",
      "rated": "uncommensurableness",
      "rating": 0.05055809746548934,
      "confidence": 0.7484706448605812,
      "normal-rating": 0.8677887734049299,
      "sample-size": 4059,
      "expires": 3969,
      "bearer": "nitty",
      "faucal": "postulnar",
      "naturalism": "sarcotic"
    },
    {
      "rater": "precreed",
      "assertion": "xanthosis",
      "rated": "balsamy",
      "rating": 0.36091333590593955,
      "confidence": 0.3700759808403371,
      "sample-size": 3904
    },
    {
      "rater": "urinosexual",
      "assertion": "malacostracous",
      "rated": "arenariae",
      "rating": 0.9210673488013762,
      "normal-rating": 0.4778762617112776,
      "sample-size": 4428,
      "generated": 3294,
      "backfurrow": "enterable",
      "fruitgrower": "flannelflower"
    },
    {
      "rater": "pedologistically",
      "assertion": "unmetaphysical",
      "rated": "elocutionist",
      "rating": 0.42073613384304287,
      "misimagine": "retinaculum",
      "snobbish": "contradict",
      "Bosporanic": "periostotomy",
      "dayworker": "intragyral"
    }
  ]
}
```

H.1.1. Examples from JSON Content Rules

Although JSON Content Rules [I-D.newton-json-content-rules] seems to address a more general problem than CDDL, it is still a worthwhile resource to explore for examples (beyond all the inspiration the format itself has had for CDDL).

Figure 2 of the JCR I-D looks very similar, if slightly less noisy, in CDDL:

```
root = [2*2 {
  precision: text,
  Latitude: float,
  Longitude: float,
  Address: text,
  City: text,
  State: text,
  Zip: text,
  Country: text
}]
```

Figure 17: JCR, Figure 2, in CDDL

Apart from the lack of a need to quote the member names, text strings are called "text" or "tstr" in CDDL ("string" would be ambiguous as CBOR also provides byte strings).

The CDDL tool creates the below example instance for this:

```
 [{"precision": "pyrosphere", "Latitude": 0.5399712314350172,
   "Longitude": 0.5157523963028087, "Address": "resow",
   "City": "problemwise", "State": "martyrlike", "Zip": "preprove",
   "Country": "Pace"},
  {"precision": "unrigging", "Latitude": 0.10422704368372193,
   "Longitude": 0.6279808663725834, "Address": "picturedom",
   "City": "decipherability", "State": "autometry", "Zip": "pout",
   "Country": "wimple"}]
```

Figure 4 of the JCR I-D in CDDL:

```
root = { image }

image = (
  Image: {
    size,
    Title: text,
    thumbnail,
    IDs: [* int]
  }
)

size = (
  Width: 0..1280
  Height: 0..1024
)

thumbnail = (
  Thumbnail: {
    size,
    Url: ~uri
  }
)
```

This shows how the group concept can be used to keep related elements (here: width, height) together, and to emulate the JCR style of specification. (It also shows referencing a type by unwrapping a tag from the prelude, "uri" - this could be done differently.) The more compact form of Figure 5 of the JCR I-D could be emulated like this:

```
root = {
  Image: {
    size, Title: text,
    Thumbnail: { size, Url: ~uri },
    IDs: [* int]
  }
}

size = (
  Width: 0..1280,
  Height: 0..1024,
)
```

The CDDL tool creates the below example instance for this:

```
{"Image": {"Width": 566, "Height": 516, "Title": "leisterer",
  "Thumbnail": {"Width": 1111, "Height": 176, "Url": 32("scrog")},
  "IDs": []}}
```

Editorial Comments

[_format] So far, the ability to restrict format choices have not been needed beyond the floating point formats. Those can be applied to ranges using the new .and control now. It is not clear we want to add more format control before we have a use case.

[_range] TO DO: define this precisely. This clearly includes integers and floats. Strings - as in "a".."z" - could be added if desired, but this would require adopting a definition of string ordering and possibly a successor function so "a".."z" does not include "bb".

[_strings] TO DO: This still needs to be fully realized in the ABNF and in the CDDL tool.

[_bitsendian] How useful would it be to have another variant that counts bits like in RFC box notation? (Or at least per-byte? 32-bit words don't always perfectly mesh with byte strings.)

[unflex] A comment has been that this is counter-intuitive. One solution would be to simply disallow unparenthesized usage of occurrence indicators in front of type choices unless a member key is also present like in group2 above.

[_abnftodo] Potential improvements: the prefixed byte strings are more liberally specified than they actually are. [^_abnfdontdo]: representation indicators are not supported. - and this will stay so.

[tdate] The prelude as included here does not yet have a .regexp control on tdate, but we probably do want to have one.

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
Darmstadt 64295
Germany

Email: henk.birkholz@sit.fraunhofer.de

Christoph Vigano
Universitaet Bremen

Email: christoph.vigano@uni-bremen.de

Carsten Bormann
Universitaet Bremen TZI
Bibliothekstr. 1
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

INTERNET-DRAFT
Intended Status: Informational
Expires: April 16, 2015

R. Huang
H.Song
Huawei

October 13, 2014

Problem Statement and Use Cases for Application Intelligent Policy Interface (AIPI)
draft-huang-appsawg-aipi-ps-uc-00

Abstract

This document discusses the problem statement and use cases of an interface named Application Intelligent Policy Interface (AIPI) that translates the real application requirements (e.g., network should ensure how many users of the OTT provider watch high definition video simultaneously.) into network level requirements, e.g., configurations or service deployments requirements and network QoS requirements, are useful and necessary for modern networks.

Table of Contents

1 Introduction

With the rapid development of network technologies, the network is more and more open. Many new technologies, e.g., SDN and NFV, etc., are emerging to help OTT application providers to get control of the network more flexibly than before. So far, these technologies are all network oriented, which based on network elements, ports, links, protocols or virtualization of the above. For those OTT application developers with little knowledge of network or small enterprise, they may not be able to fully utilize the network resources or correctly set network services to meet their requirements from the services and applications they provide.

This document further argues that an interface that translates the real application requirements (e.g., network should ensure how many users of the OTT provider watch high definition video simultaneously.) into network level requirements, e.g., configurations or service deployments requirements and network QoS requirements, are useful and necessary for modern networks. How the interface looks like is an issue outside the scope of the present document. The remainder of this document develops this idea into detail: Section 2 provides terminology; Section 3 discusses the problem statements; Section 4 provides some use cases in which the interface could be used, and Section 5 analyses the current related IETF work.

2 Terminology

AIPI: Application Intelligent Policy Interface, which is used for translating the application requirements to network level requirements.

Software-Defined Networking (SDN): A framework that supports the separation of Control and Forwarding Planes via standardized interfaces.

Network Functions Virtualization (NFV): A network architecture concept that proposes using IT virtualization related technologies to virtualize entire classes of network node functions into building blocks that may be connected, or chained, together to create communication services.

Network Policy: network level policies that can be directly used to configure the network elements

Application Intelligent Policy: Application oriented policies that reflect the QoS of the application service, but cannot be directly configured to network elements

3. Problem Statement

3.1 Gap Between Application Requirements and Network

Traditional network and applications effectively treat each other as black boxes. Network is effectively isolated from the end-systems, which then have little control over how the network handles packets. Likewise, the network has limited visibility on the application logic.

With the increasingly demanding of network opening, technologies like SDN and NFV becomes the future trends of network development. A key objective of this open network is to facilitate and exploit the virtualization of network functions so that they can be provided on a programmable basis for applications and service innovators. It seems to be a good way for OTT or enterprise applications to have some insight into network infrastructure. However, applications may focus more on service logic, service implementation, performance and experience, in stead of network information. Deep network capabilities exposed by the programmable network will be impossible for non-network-owning players to replicate.

Moreover, many different sets of network interfaces for applications are emerging. They provide network functions like path computation, loop avoidance, routing, security and many other tasks through abstracting and normalizing the quirks and eccentricities of different data plane devices. However, these interfaces are all network oriented. OTT applications may not adequately use these interfaces to fulfill their own requirement as they have no idea how their service running in the network really works. For example, OTT applications may not know which one is the most suitable path to choose. Also, there is no common consensus about how those parameters of network (e.g., bandwidth, delay, jitter, etc.) affect the QoE of OTT applications.

3.2 Difficulty for Network adaptation

Most of current interfaces really provides a way for OTT applications to easily apply for network resources or network functions or certain QoS (delay, loss, jitter). Due to the gap between OTT applications requirements and network, OTT applications may abuse these interface for their goods. For example, OTT applications may request more bandwidth than that they really need, or they may apply for some functions that they never use.

There are also some cases that requests from the OTT applications can’t be accepted for network because of extravagant demands, while there do exist some other ways to fulfill the real requirements of

OTT applications. Obviously, current interfaces between OTT applications and network couldn't do network adaptations since network has no idea what OTT applications really want.

3.3 Difficulty for Network Policies Maintenance

Plenty varieties of requirements for service performance and access control breed plenty of complicated network configuration or QoS rules and ACL policies deployments. Usually these configurations and polices are maintained by network administrators who may not understand what the corresponding real service requirements are. With the change of network configuration and policies due to the updates of service requirements, e.g., new service deployment or old service expired, the maintenance of the network becomes more and more difficult especially along with the transfer of network administrators.

4. Use Cases for Application Intelligent Policy Interface

If network has an interface enabling OTT applications to input their own application service requirements and has the ability to translate the application service requirements to network requirement, problems will be solved. AIPI (application intelligent policy interface) is dedicated to cope with such problems. In this document, we mainly discuss several use cases that AIPI which is application friend could be applied.

4.1 OTT Service Deployment

OTT service providers can benefit from using such an application oriented interface so that they don't have to care about the underlay network details of no matter physical or virtual devices and functions. Instead, they just need to focus on what they are good at, which are their applications and services for end users.

Suppose an OTT video provider wants to activate the service which ensure 1000 end user to watch its high definition live video content simultaneously. Traditional way is to request some specific bandwidth resources or CDN services or VPN from ISPs. These kind of requests are usually exaggerated which may result in wasting of resources. When using AIPI, the OTT provider only needs to provide inputs from its perspective. For example, the OTT provider indicate that the video is 720p, and it's a live http streaming, and he wants his end users could watch fluently and the QoE should be excellent. Then ISP will analyse the requirements from the OTT provider, and translate them into network QoS, e.g., bandwidth >5120kpbs, RTT<120ms, jitter<50ms and loss<1%.

When OTT providers have same service deployments in different networks, they can also benefit by using AIPI. Same request can be transmitted by AIPI to different ISP networks instead of figuring out what network resources they need in each different network.

4.2 Enterprise Service Isolation and Interconnection

Enterprise networks are growing in complexity and size with globalization, outsourcing, and wireless expanding the reach of the network beyond its traditional design parameters. IT departments are tasked with ensuring the applications and services run well across both private and public networks. Add to that ongoing application and data center projects such as virtualization and cloud computing all the components that make up that network becomes even more of a challenge. Especially with the frequent personnel transfer of functions, locations or departments, IT departments are facing a huge workload.

AIPI is a way to make IT departments work more efficiently and effectively. Using AIPI, the administrators of IT departments just need to maintain the requirements from their enterprise and leave the automatic configurations and deployments to network service providers, which hence largely reduces the OPEX of IT departments.

4.3 Automatic Adaptation

AIPI is also helpful for network providers to achieve automatic adaptation for OTT service providers. With AIPI, network providers who know network the best will learn the real requirements from OTT service providers, and they certainly have the ability to figure out how to satisfy these requirements with minimum resources and maximum efficiency.

The advantage is also reflected when network problems affecting OTT service SLAs happen. For example, when a node on the specific OTT service path is experiencing congestions, network provider could just choose another path which satisfies the requirements for this OTT service without any communication with this OTT provider.

5 Existing Work

There are some existing efforts in IETF that may be related to this work. In this section, we give a detailed analysis towards them.

Application Enabled Collaborative Networking (AECON): AECON [AECON] is trying to work out ways to enable communication of flow characteristics between applications in end users and the network by

active information exchange. And the information communicated through AECON is network oriented.

Shared Unified Policy Automation (SUPA): The main goal of SUPA [SUPA] is to provide a way for network management applications and for network services to specify share application based policies to the network infrastructure using a simplified view of the network. SUPA is also a network oriented interface and does not really concern OTT applications.

Interface to the Routing System (I2RS): I2RS [I2RS] facilitates real-time or event driven interaction with the routing system through a collection of protocol-based control or management interfaces which allow information, policy and operational parameters to be injected into or retrieved from the routing system of network. It is network oriented as it collects and injects network information.

Application-Layer Traffic Optimization (ALTO): ALTO [ALTO] is considered as a solution to expose abstract topologies and costs to applications in P2P domain, datacenter networks and content distribution networks (CDN). It is also network oriented rather than application oriented.

AIPI tries to specify an application oriented interface which conveys the real application requirements to network and facilitates the translation from real application requirements to network requirements. AIPI is about the communication between Application deployment servers and network. AIPI could further use SUPA to configure or adjust network.

6 IANA Considerations

This draft includes no request to IANA.

7 Security Considerations

TBD.

8 Acknowledgments

The authors would like to thank Hanyu Wei for giving valuable comments and suggestions.

9 References

9.1 Normative References

[SUPA] IETF SUPA (Shared Unified Policy Automation) BoF Charter,

https://github.com/liushucheng/SUPA/wiki/SUPA-Charter

[AECON] IETF AECON (Application Enabled Collaborative Networking) barBoF, http://trac.tools.ietf.org/bof/trac/wiki/BofIETF90#

[I2RS] IETF I2RS (Interface to the Routing System) WG Charter, http://datatracker.ietf.org/wg/i2rs/charter/

[ALTO] IETF ALOT (Application-Layer Traffic Optimization), http://datatracker.ietf.org/wg/alto/charter/

9.2 Informative References

[NFV] http://en.wikipedia.org/wiki/Network_Functions_Virtualization

[SDN] http://tools.ietf.org/html/draft-haleplidis-sdnrg-layer-terminology-06

Authors' Addresses

Rachel Huang
Huawei
101 Software Avenue, Yuhua District
Nanjing 210012
China

EMail: rachel.huang@huawei.com

Haibin Song
Huawei
101 Software Avenue, Yuhua District
Nanjing 210012
China

EMail: songhaibin@huawei.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 27, 2014

P. Saint-Andre

January 23, 2014

The 'acct' URI Scheme
draft-ietf-appsawg-acct-uri-07

Abstract

This document defines the 'acct' Uniform Resource Identifier (URI) scheme as a way to identify a user's account at a service provider, irrespective of the particular protocols that can be used to interact with the account.

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."

This Internet-Draft will expire on July 27, 2014.

Table of Contents

1. Introduction

Existing Uniform Resource Identifier (URI) schemes that enable interaction with, or that identify resources associated with, a user's account at a service provider are tied to particular services or application protocols. Two examples are the 'mailto' scheme (which enables interaction with a user's email account) and the 'http' scheme (which enables retrieval of web files controlled by a user or interaction with interfaces providing information about a user). However, there exists no URI scheme that generically identifies a user's account at a service provider without specifying a particular protocol to use when interacting with the account. This specification fills that gap.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Rationale

During formalization of the WebFinger protocol [RFC7033], much discussion occurred regarding the appropriate URI scheme to include when specifying a user's account as a web link [RFC5988]. Although both the 'mailto' [RFC6068] and 'http' [RFC2616] schemes were proposed, not all service providers offer email services or web interfaces on behalf of user accounts (e.g., a microblogging or instant messaging provider might not offer email services, or an enterprise might not offer HTTP interfaces to information about its employees). Therefore, the participants in the discussion recognized that it would be helpful to define a URI scheme that could be used to generically identify a user's account at a service provider, irrespective of the particular application protocols used to interact

with the account. The result was the 'acct' URI scheme defined in this document.

(Note that a user is not necessarily a human; it could be an automated application such as a bot, a role-based alias, etc. However, an 'acct' URI is always used to identify something that has an account at a service, not the service itself.)

4. Definition

The syntax of the 'acct' URI scheme is defined under Section 7 of this document. Although 'acct' URIs take the form "user@host", the scheme is designed for the purpose of identification instead of interaction (regarding this distinction, see Section 1.2.2 of [RFC3986]). The "Internet resource" identified by an 'acct' URI is a user's account hosted at a service provider, where the service provider is typically associated with a DNS domain name. Thus a particular 'acct' URI is formed by setting the "user" portion to the user's account name at the service provider and by setting the "host" portion to the DNS domain name of the service provider.

Consider the case of a user with an account name of "foobar" on a microblogging service "status.example.net". It is taken as convention that the string "foobar@status.example.net" designates that account. This is expressed as a URI using the 'acct' scheme as "acct:foobar@status.example.net".

A common scenario is for a user to register with a service provider using an identifier (such as an email address) that is associated with some other service provider. For example, a user with the email address "juliet@capulet.example" might register with a commerce website whose domain name is "shoppingsite.example". In order to use her email address as the localpart of the 'acct' URI, the at-sign character (U+0040) needs to be percent-encoded as described in [RFC3986]. Thus the resulting 'acct' URI would be "acct:juliet%40capulet.example@shoppingsite.example".

It is not assumed that an entity will necessarily be able to interact with a user's account using any particular application protocol, such as email; to enable such interaction, an entity would need to use the appropriate URI scheme for such a protocol, such as the 'mailto' scheme. While it might be true that the 'acct' URI minus the scheme name (e.g., "user@example.com" derived from "acct:user@example.com") can be reached via email or some other application protocol, that fact would be purely contingent and dependent upon the deployment practices of the provider.

Because an 'acct' URI enables abstract identification only and not interaction, this specification provides no method for dereferencing an 'acct' URI on its own, e.g., as the value of the 'href' attribute of an HTML anchor element. For example, there is no behavior specified in this document for an 'acct' URI used as follows:

find out more

Any protocol that uses 'acct' URIs is responsible for specifying how an 'acct' URI is employed in the context of that protocol (in particular, how it is dereferenced or resolved; see [RFC3986]). As a concrete example, an "Account Information" application of the WebFinger protocol [RFC7033] might take an 'acct' URI, resolve the host portion to find a WebFinger server, and then pass the 'acct' URI as a parameter in a WebFinger HTTP request for metadata (i.e., web links [RFC5988]) about the resource. For example:

GET /.well-known/webfinger?resource=acct%3Abob%40example.com HTTP/1.1

The service retrieves the metadata associated with the account identified by that URI and then provides that metadata to the requesting entity in an HTTP response.

If an application needs to compare two 'acct' URIs (e.g., for purposes of authentication and authorization), it MUST do so using case normalization and percent-encoding normalization as specified in Sections 6.2.2.1 and 6.2.2.2 of [RFC3986].

5. Security Considerations

Because the 'acct' URI scheme does not directly enable interaction with a user's account at a service provider, direct security concerns are minimized.

However, an 'acct' URI does provide proof of existence of the account; this implies that harvesting published 'acct' URIs could prove useful to spammers and similar attackers, for example if they can use an 'acct' URI to leverage more information about the account (e.g., via WebFinger) or if they can interact with protocol-specific URIs (such as 'mailto' URIs) whose user@host portion is the same as that of the 'acct' URI.

In addition, protocols that make use of 'acct' URIs are responsible for defining security considerations related to such usage, e.g., the risks involved in dereferencing an 'acct' URI, the authentication and authorization methods that could be used to control access to

personal data associated with a user's account at a service, and methods for ensuring the confidentiality of such information.

The use of percent-encoding allows a wider range of characters in account names, but introduces some additional risks. Implementers are advised to disallow percent-encoded characters or sequences that would (1) result in space, null, control, or other characters that are otherwise forbidden, (2) allow unauthorized access to private data, or (3) lead to other security vulnerabilities.

6. Internationalization Considerations

As specified in [RFC3986], the 'acct' URI scheme allows any character from the Unicode repertoire [UNICODE] encoded as UTF-8 [RFC3629] and then percent-encoded into valid ASCII [RFC20]. Before applying any percent-encoding, an application MUST ensure the following about the string that is used as input to the URI-construction process:

- The userpart consists only of Unicode code points that conform to the PRECIS IdentifierClass specified in [I-D.ietf-precis-framework].
- The host consists only of Unicode code points that conform to the rules specified in [RFC5892].
- Internationalized domain name (IDN) labels are encoded as A-labels [RFC5890].

7. IANA Considerations

In accordance with the guidelines and registration procedures for new URI schemes [RFC4395], this section provides the information needed to register the 'acct' URI scheme.

7.1. URI Scheme Name

acct

7.2. Status

permanent

7.3. URI Scheme Syntax

The 'acct' URI syntax is defined here in Augmented Backus-Naur Form (ABNF) [RFC5234], borrowing the 'host', 'pct-encoded', 'sub-delims', 'unreserved' rules from [RFC3986]:

```
acctURI      =  "acct" ":" userpart "@" host
userpart     =  unreserved / sub-delims
                0*( unreserved / pct-encoded / sub-delims )
```

Note that additional rules regarding the strings that are used as input to construction of 'acct' URIs further limit the characters that can be percent-encoded; see the Encoding Considerations as well as Section 6 of RFC XXXX. [Note to RFC Editor: please replace XXXX with the number issued to this document.]

7.4. URI Scheme Semantics

The 'acct' URI scheme identifies accounts hosted at service providers. It is used only for identification, not interaction. A protocol that employs the 'acct' URI scheme is responsible for specifying how an 'acct' URI is dereferenced in the context of that protocol. There is no media type associated with the 'acct' URI scheme.

7.5. Encoding Considerations

See Section 6 of RFC XXXX. [Note to RFC Editor: please replace XXXX with the number issued to this document.]

7.6. Applications/Protocols That Use This URI Scheme Name

At the time of this writing, only the WebFinger protocol uses the 'acct' URI scheme. However, use is not restricted to the WebFinger protocol, and the scheme might be considered for use in other protocols.

7.7. Interoperability Considerations

There are no known interoperability concerns related to use of the 'acct' URI scheme.

7.8. Security Considerations

See Section 5 of RFC XXXX. [Note to RFC Editor: please replace XXXX with the number issued to this document.]

7.9. Contact

Peter Saint-Andre, psaintan@cisco.com

7.10. Author/Change Controller

This scheme is registered under the IETF tree. As such, the IETF maintains change control.

7.11. References

None.

8. References

8.1. Normative References

[I-D.ietf-precis-framework]
Saint-Andre, P. and M. Blanchet, "Precis Framework: Handling Internationalized Strings in Protocols", draft-ietf-precis-framework-13 (work in progress), December 2013.

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

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

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

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, August 2010.

[RFC5892] Faltstrom, P., "The Unicode Code Points and Internationalized Domain Names for Applications (IDNA)", RFC 5892, August 2010.

[UNICODE] The Unicode Consortium, "The Unicode Standard, Version 6.1", 2012, <http://www.unicode.org/versions/Unicode6.1.0/>.

8.2. Informative References

[RFC20] Cerf, V., "ASCII format for network interchange", RFC 20, October 1969.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 35, RFC 4395, February 2006.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

[RFC6068] Duerst, M., Masinter, L., and J. Zawinski, "The 'mailto' URI Scheme", RFC 6068, October 2010.

[RFC7033] Jones, P., Salgueiro, G., Jones, M., and J. Smarr, "WebFinger", RFC 7033, September 2013.

Appendix A. Acknowledgements

The 'acct' URI scheme was originally proposed during work on the WebFinger protocol; special thanks are due to Blaine Cook, Brad Fitzpatrick, and Eran Hammer-Lahav for their early work on the concept (which in turn was partially inspired by work on Extensible Resource Indentifiers at OASIS). The scheme was first formally specified in [RFC7033]; the authors of that specification (Paul Jones, Gonzalo Salgueiro, and Joseph Smarr) are gratefully acknowledged. Thanks are also due to Stephane Bortzmeyer, Melvin Carvalho, Martin Duerst, Graham Klyne, Barry Leiba, Subramanian Moonesamy, Evan Prodromou, James Snell, and various participants in the IETF APPSAWG for their feedback. Meral Shirazipour completed a Gen-ART review. Dave Cridland completed an AppsDir review, and is gratefully acknowledged for providing proposed text that was incorporated into Section 3 and Section 5. IESG comments from Richard Barnes, Adrian Farrel, Stephen Farrell, Barry Leiba, Pete Resnick, and Sean Turner also led to improvements in the specification.

Author's Address

Peter Saint-Andre

Email: ietf@stpeter.im

Individual submission
Internet-Draft
Updates: 7001 (if approved)
Intended status: Standards Track
Expires: April 3, 2015

M. Kucherawy
September 30, 2014

A Property Types Registry for the Authentication-Results Header Field
draft-ietf-appsawg-authres-ptypes-registry-04

Abstract

This document updates RFC7001 by creating a registry for property types in the Authentication-Results header field, used in email authentication work, rather than limiting participants to using the original, small set of fixed values.

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."

This Internet-Draft will expire on April 3, 2015.

Table of Contents

1. Introduction

[RFC7001] defines the email Authentication-Results header field that presents the results of an authentication effort in a machine-readable format. The header field creates a place to collect the output from authentication processes that are disjoint from later processes that might use the output, such as analysis, filtering or sorting mechanisms.

The specification in that document enumerated a small set of types of properties that can be reported using this mechanism. There has emerged a desire to report types of properties about a message through this mechanism. Accordingly, this document updates the specification to allow for additional property types ("ptypes") beyond the original set, and creates a registry where new ones can be listed and their defining documents referenced.

2. Updated 'ptype' Definition

Advanced Backus Naur Form (ABNF) is defined in [RFC5234].

The ABNF in Section 2.2 of [RFC7001] is updated as follows:

```
    ptype = Keyword
          ; indicates whether the property being evaluated was
          ; a parameter to an [SMTP] command, was a value taken
          ; from a message header field, was some property of
          ; the message body, or was some other property evaluated by
          ; the receiving Message Transfer Agent (MTA)
```

The ABNF token "Keyword" is defined in Section 4.1.2 of [RFC5321].

Legal values of "ptype" are as defined in the IANA "Email Authentication Property Types" registry (see Section 3). The initial values are as follows, matching those defined in [RFC7001]:

body: Indicates information that was extracted from the body of the message. This might be an arbitrary string of bytes, a hash of a string of bytes, a Uniform Resource Identifier, or some other content of interest.

header: Indicates information that was extracted from the header of the message. This might be the value of a header field or some portion of a header field.

policy: A local policy mechanism was applied that augments or overrides the result returned by the authentication mechanism. See Section 2.3 of [RFC7001].

smtp: Indicates information that was extracted from an SMTP command that was used to relay the message.

When a consumer of this header field encounters a ptype that it does not understand, it ignores the result reported with that ptype.

3. IANA Considerations

IANA is requested to create the Email Authentication Property Types sub-registry within the existing Email Authentication Parameters registry. Entries in this registry are subject to the Expert Review rules as described in [RFC5226]. Each entry in the registry requires the following values:

- The "ptype" token to be registered, which must fit within the ABNF described in Section 2.
- A brief description of what sort of information this "ptype" is meant to cover.
- An optional reference to the defining document. This is recomended, but not required.

The initial entries in this table are as follows, taken from [RFC7001]:

ptype	Definition	Description
body	RFC7001 Section 2.2	The property being reported was found in the body of the message.
header	RFC7001 Section 2.2	The property being reported was found in a header field of the message.
policy	RFC7001 Section 2.3	The property being reported relates to a locally-defined policy.
smtp	RFC7001 Section 2.2	The property being reported is a parameter to an SMTP command used to relay the message.

For new entries, the Designated Expert needs to assure that the

description provided for the new entry adequately describes the intended use. An example would be helpful to include in the entry's defining document, if any, although entries in the Email Authentication Methods registry or the Email Authentication Result Names registry might also serve as examples of intended use.

4. Security Considerations

It is unknown how legacy code, which expects one of a fixed set of "ptype" tokens, will handle new tokens as they begin to appear. There are typically two options: prevent delivery of the message, or ignore those portions of the field that use unknown "ptype" tokens and allow processing of the message to continue.

The choice comes down to whether the consumer considers it a threat when there are unknown "ptypes" present. The semantics of the report are unknown; the report might be indicating the message is authentic, fraudulent, or that a test failed to complete. The report itself is not actionable because it cannot be understood, and only its presence is certain.

Generally, the advice in this situation is to ignore unknown "ptypes". It is anticipated that a new property type evaluated by earlier handling agents would also result in the filtering of messages by those agents until consumers can be updated to interpret them.

5. Normative References

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, October 2008.

[RFC7001] Kucherawy, M., "Message Header Field for Indicating Message Authentication Status", RFC 7001, September 2013.

Appendix A. Acknowledgements

The author wishes to acknowledge the following for their review and constructive criticism of this update: Dave Crocker, Tim Draegen, Scott Kitterman, Franck Martin.

Author's Address

Murray S. Kucherawy
270 Upland Drive
San Francisco, CA 94127
US

EMail: superuser@gmail.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 23, 2016

M. Nottingham
Akamai
E. Wilde

January 20, 2016

Problem Details for HTTP APIs
draft-ietf-appsawg-http-problem-03

Abstract

This document defines a "problem detail" as a way to carry machine-readable details of errors in a HTTP response, to avoid the need to define new error response formats for HTTP APIs.

Note to Readers

This draft should be discussed on the apps-discuss mailing list [1].

This section is to be removed before publication.

Note to RFC Editor

Please replace all occurrences of "XXXX" with the final RFC number chosen for this draft.

This section is to be removed before publication.

Status of This Memo

Table of Contents

1. Introduction

HTTP [RFC7230] status codes are sometimes not sufficient to convey enough information about an error to be helpful. While humans behind Web browsers can be informed about the nature of the problem with an HTML [W3C.REC-html5-20141028] response body, non-human consumers of so-called "HTTP APIs" are usually not.

This specification defines simple JSON [RFC7159] and XML [W3C.REC-xml-20081126] document formats to suit this purpose. They are designed to be reused by HTTP APIs, which can identify distinct "problem types" specific to their needs.

Thus, API clients can be informed of both the high-level error class (using the status code) and the finer-grained details of the problem (using one of these formats).

For example, consider a response that indicates that the client's account doesn't have enough credit. The 403 Forbidden status code might be deemed most appropriate to use, as it will inform HTTP-generic software (such as client libraries, caches and proxies) of the general semantics of the response.

However, that doesn't give the API client enough information about why the request was forbidden, the applicable account balance, or how to correct the problem. If these details are included in the response body in a machine-readable format, the client can treat it appropriately; for example, triggering a transfer of more credit into the account.

This specification does this by identifying a specific type of problem (e.g., "out of credit") with a URI [RFC3986]; HTTP APIs can do this by nominating new URIs under their control, or by reusing existing ones.

Additionally, problems can contain other information, such as a URI that identifies the specific occurrence of the problem (effectively giving an identifier to the concept "The time Joe didn't have enough credit last Thursday"), which can be useful for support or forensic purposes.

The data model for problem details is a JSON [RFC7159] object; when formatted as a JSON document, it uses the "application/problem+json" media type. Appendix A defines how to express them in an equivalent XML format, which uses the "application/problem+xml" media type.

Note that problem details are (naturally) not the only way to convey the details of a problem in HTTP; if the response is still a representation of a resource, for example, it's often preferable to accommodate describing the relevant details in that application's format. Likewise, in many situations, there is an appropriate HTTP status code that does not require extra detail to be conveyed.

Instead, the aim of this specification is to define common error formats for those applications that need one, so that they aren't required to define their own, or worse, tempted to re-define the semantics of existing HTTP status codes. Even if an application chooses not to use it to convey errors, reviewing its design can help guide the design decisions faced when conveying errors in an existing format.

2. Requirements

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].

3. The Problem Details JSON Object

The canonical model for problem details is a JSON [RFC7159] object.

When serialised as a JSON document, that format is identified with the "application/problem+json" media type.

For example, a HTTP response carrying JSON problem details:

```
HTTP/1.1 403 Forbidden
Content-Type: application/problem+json
Content-Language: en

{
 "type": "https://example.com/probs/out-of-credit",
 "title": "You do not have enough credit.",
 "detail": "Your current balance is 30, but that costs 50.",
 "instance": "/account/12345/msgs/abc",
 "balance": 30,
 "accounts": ["/account/12345",
              "/account/67890"]
}
```

Here, the out-of-credit problem (identified by its type URI) indicates the reason for the 403 in "title", gives a reference for the specific problem occurrence with "instance", gives occurrence-specific details in "detail", and adds two extensions; "balance" conveys the account’s balance, and "accounts" gives links where the account can be topped up.

The ability to convey problem-specific extensions allows more than one problem to be conveyed. For example:

```
HTTP/1.1 400 Bad Request
Content-Type: application/problem+json
Content-Language: en

{
"type": "https://example.net/validation-error",
"title": "Your request parameters didn't validate.",
"invalid-params": [ {
                      "name": "age",
                      "reason": "must be a positive integer"
                    },
                    {
                      "name": "color",
                      "reason": "must be 'green', 'red' or 'blue'"}
                  ]
}
```

Note that this requires each of the sub-problems to be similar enough to use the same HTTP status code. If they do not, the 207 (Multi-Status) [RFC4918] code could be used to encapsulate multiple status messages.

3.1. Problem Details Object Members

A problem details object can have the following members:

- "type" (string) - A URI reference [RFC3986] that identifies the problem type. When dereferenced, it is encouraged to provide human-readable documentation for the problem type (e.g., using HTML [W3C.REC-html5-20141028]). When this member is not present, its value is assumed to be "about:blank".

- "title" (string) - A short, human-readable summary of the problem type. It SHOULD NOT change from occurrence to occurrence of the problem, except for purposes of localisation (e.g., using proactive content negotiation; see [RFC7231], Section 3.4).

- "status" (number) - The HTTP status code ([RFC7231], Section 6) generated by the origin server for this occurrence of the problem.

- "detail" (string) - An human readable explanation specific to this occurrence of the problem.

- "instance" (string) - A URI reference that identifies the specific occurrence of the problem. It may or may not yield further information if dereferenced.

Consumers MUST use the type string as the primary identifier for the problem type; the title string is advisory, and included only for users who are not aware of the semantics of the URI, and don't have the ability to discover them (e.g., offline log analysis). Consumers SHOULD NOT automatically dereference the type URI.

The status member, if present, is only advisory; it conveys the HTTP status code used for the convenience of the consumer. Generators MUST use the same status code in the actual HTTP response, to assure that generic HTTP software that does not understand this format still behaves correctly. See Section 5 for further caveats regarding its use.

Consumers can use the status member to determine what the original status code used by the generator was, in cases where it has been changed (e.g., by an intermediary or cache), and when message bodies are persisted without HTTP information. Generic HTTP software will still use the HTTP status code.

The detail member, if present, ought to focus on helping the client correct the problem, rather than giving debugging information.

Consumers SHOULD NOT parse the detail member for information; extensions are more suitable and less error-prone ways to obtain such information.

Note that both "type" and "instance" accept relative URIs; this means that they must be resolved relative to the document's base URI, as per [RFC3986], Section 5.

3.2. Extension Members

Problem type definitions MAY extend the problem details object with additional members.

For example, our "out of credit" problem above defines two such extensions, "balance" and "accounts" to convey additional, problem-specific information.

Clients consuming problem details MUST ignore any such extensions that they don't recognise; this allows problem types to evolve and include additional information in the future.

Note that because extensions are effectively name spaced by the problem type, it is not possible to define new "standard" members without defining a new media type.

4. Defining New Problem Types

When an HTTP API needs to define a response that indicates an error condition, it might be appropriate to do so by defining a new problem type.

Before doing so, it's important to understand what they are good for, and what's better left to other mechanisms.

Problem details are not a debugging tool for the underlying implementation; rather, they are a way to expose greater detail about the HTTP interface itself. Designers of new problem types need to carefully consider the Security Considerations (Section 5); in particular the risk of exposing attack vectors by exposing implementation internals through error messages.

Likewise, truly generic problems - i.e., conditions that could potentially apply to any resource on the Web - are usually better expressed as plain status codes. For example, a "write access disallowed" problem is probably unnecessary, since a 403 Forbidden status code in response to a PUT request is self-explanatory.

Finally, an application might have a more appropriate way to carry an error in a format that it already defines. Problem details are intended to avoid the necessity of establishing new "fault" or "error" document formats, not to replace existing domain-specific formats.

That said, it is possible to add support for problem details to existing HTTP APIs using HTTP content negotiation (e.g., using the Accept request header to indicate a preference for this format; see [RFC7231], Section 5.3.2).

New problem type definitions MUST document:

1. A type URI (typically, with the "http" or "https" scheme),

2. A title that appropriately describes it (think short), and

3. The HTTP status code for it to be used with.

Problem type definitions MAY specify the use of the Retry-After response header ([RFC7231], Section 7.1.3) in appropriate circumstances.

A problem's type URI SHOULD resolve to HTML [W3C.REC-html5-20141028] documentation that explains how to resolve the problem.

A problem type definition MAY specify additional members on the Problem Details object. For example, an extension might use typed links [RFC5988] to another resource that can be used by machines to resolve the problem.

If such additional members are defined, their names SHOULD start with a letter (ALPHA, as per [RFC5234], Appendix B.1) and SHOULD consist of characters from ALPHA, DIGIT (Ibid.), and "_" (so that it can be serialized in formats other than JSON), and SHOULD be three characters or longer.

4.1. Example

For example, if you are publishing an HTTP API to your online shopping cart, you might need to indicate that the user is out of credit (our example from above), and therefore cannot make the purchase.

If you already have an application-specific format that can accommodate this information, it's probably best to do that. However, if you don't, you might consider using one of the problem details formats; JSON if your API is JSON-based, or XML if it uses that format.

To do so, you might look for an already-defined type URI that suits your purposes. If one is available, you can reuse that URI.

If one isn't available, you could mint and document a new type URI (which ought to be under your control and stable over time), an appropriate title and the HTTP status code that it will be used with, along with what it means and how it should be handled.

In summary: an instance URI will always identify a specific occurrence of a problem. On the other hand, type URIs can be reused if an appropriate description of a problem type is already available someplace else, or they can be created for new problem types.

4.2. Pre-Defined Problem Types

This specification reserves the use of one URI as a problem type:

The "about:blank" URI [RFC6694], when used as a problem type, indicates that the problem has no additional semantics beyond that of the HTTP status code.

When "about:blank" is used, the title SHOULD be the same as the recommended HTTP status phrase for that code (e.g., "Not Found" for

404, and so on), although it MAY be localized to suit client preferences (expressed with the Accept-Language request header).

Please note that according to how the "type" member is defined (Section 3.1), the "about:blank" URI is the default value for that member. Consequently, any problem details object not carrying an explicit "type" member implicitly uses this URI.

5. Security Considerations

When defining a new problem type, the information included must be carefully vetted. Likewise, when actually generating a problem - however it is serialized - the details given must also be scrutinized.

Risks include leaking information that can be exploited to compromise the system, access to the system, or the privacy of users of the system.

Generators providing links to occurrence information are encouraged to avoid making implementation details such as a stack dump available through the HTTP interface, since this can expose sensitive details of the server implementation, its data, and so on.

The "status" member duplicates the information available in the HTTP status code itself, thereby bringing the possibility of disagreement between the two. Their relative precedence is not clear, since a disagreement might indicate that (for example) an intermediary has modified the HTTP status code in transit (e.g., by a proxy or cache).

As such, those defining problem types as well as generators and consumers of problems need to be aware that generic software (such as proxies, load balancers, firewalls, virus scanners) are unlikely to know of or respect the status code conveyed in this member.

6. IANA Considerations

This specification defines two new Internet media types [RFC6838]:

Type name: application

Subtype name: problem+json

Required parameters: None

Optional parameters: None; unrecognised parameters should be ignored

Encoding considerations: Same as [RFC7159]

Security considerations: see Section 5 of this document

Interoperability considerations: None

Published specification: [this document]

Applications that use this media type: HTTP

Fragment identifier considerations: Same as for application/json ([RFC7159])

Additional information:

Deprecated alias names for this type: n/a

Magic number(s): n/a

File extension(s): n/a

Macintosh file type code(s): n/a

Person and email address to contact for further information: Mark Nottingham <mnot@mnot.net>

Intended usage: COMMON

Restrictions on usage: None.

Author: Mark Nottingham <mnot@mnot.net>

Change controller: IESG

Type name: application

Subtype name: problem+xml

Required parameters: None

Optional parameters: None; unrecognised parameters should be ignored

Encoding considerations: Same as [RFC7303]

Security considerations: see Section 5 of this document

Interoperability considerations: None

Published specification: [this document]

Applications that use this media type: HTTP

Fragment identifier considerations: Same as for application/xml (as specified by Section 5 of [RFC7303])

Additional information:

Deprecated alias names for this type: n/a

Magic number(s): n/a

File extension(s): n/a

Macintosh file type code(s): n/a

Person and email address to contact for further information: Mark No ttingham <mnot@mnot.net>

Intended usage: COMMON

Restrictions on usage: None.

Author: Mark Nottingham <mnot@mnot.net>

Change controller: IESG

7. Acknowledgements

The authors would like to thank Jan Algermissen, Mike Amundsen, Subbu Allamaraju, Roy Fielding, Eran Hammer, Sam Johnston, Mike McCall, Julian Reschke, and James Snell for review of this specification.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[W3C.REC-xml-20081126] 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>.

8.2. Informative References

[ISO-19757-2] International Organization for Standardization, "Information Technology --- Document Schema Definition Languages (DSDL) --- Part 2: Grammar-based Validation --- RELAX NG", ISO/IEC 19757-2, 2003.

[RFC4918] Dusseault, L., Ed., "HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)", RFC 4918, DOI 10.17487/RFC4918, June 2007, <http://www.rfc-editor.org/info/rfc4918>.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <http://www.rfc-editor.org/info/rfc5988>.

[RFC6694] Moonesamy, S., Ed., "The "about" URI Scheme", RFC 6694, DOI 10.17487/RFC6694, August 2012, <http://www.rfc-editor.org/info/rfc6694>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <http://www.rfc-editor.org/info/rfc6838>.

[RFC7303] Thompson, H. and C. Lilley, "XML Media Types", RFC 7303, DOI 10.17487/RFC7303, July 2014, <http://www.rfc-editor.org/info/rfc7303>.

[W3C.REC-html5-20141028] Hickson, I., Berjon, R., Faulkner, S., Leithead, T., Navara, E., O'Connor, E., and S. Pfeiffer, "HTML5", World Wide Web Consortium Recommendation REC-html5-20141028, October 2014, <http://www.w3.org/TR/2014/REC-html5-20141028>.

[W3C.REC-rdfa-core-20130822] Adida, B., Birbeck, M., McCarron, S., and I. Herman, "RDFa Core 1.1 - Second Edition", World Wide Web Consortium Recommendation REC-rdfa-core-20130822, August 2013, <http://www.w3.org/TR/2013/REC-rdfa-core-20130822>.

[W3C.REC-xml-stylesheet-20101028] Clark, J., Pieters, S., and H. Thompson, "Associating Style Sheets with XML documents 1.0 (Second Edition)", World Wide Web Consortium Recommendation REC-xml-stylesheet-20101028, October 2010, <http://www.w3.org/TR/2010/REC-xml-stylesheet-20101028>.

Appendix A. HTTP Problems and XML

Some HTTP-based APIs use XML [W3C.REC-xml-20081126] as their primary format convention. Such APIs can express problem details using the format defined in this appendix.

The RELAX NG schema [ISO-19757-2] for the XML format is as follows. Keep in mind that this schema is only meant as documentation, and not as a normative schema that captures all constraints of the XML format. Also, it would be possible to use other XML schema languages to define a similar set of constraints (depending on the features of the chosen schema language).

```
default namespace ns = "urn:ietf:rfc:XXXX"

start = problem

problem =
  element problem {
    (  element  type            { xsd:anyURI }?
     & element  title           { xsd:string }?
     & element  detail          { xsd:string }?
     & element  status          { xsd:positiveInteger }?
     & element  instance        { xsd:anyURI }? ),
    anyNsElement
  }

anyNsElement =
  (  element    ns:*  { anyNsElement | text }
   | attribute  *     { text })*
```

The media type for this format is "application/problem+xml".

Extension arrays and objects are serialized into the XML format by considering an element containing a child or children to represent an object, except for elements that contain only child element(s) named 'i', which are considered arrays. For example, the XML version of the example above appears in XML as follows:

```
HTTP/1.1 403 Forbidden
Content-Type: application/problem+xml
Content-Language: en

<?xml version="1.0" encoding="UTF-8"?>
<problem xmlns="urn:ietf:rfc:XXXX">
  <type>https://example.com/probs/out-of-credit</type>
  <title>You do not have enough credit.</title>
  <detail>Your current balance is 30, but that costs 50.</detail>
  <instance>https://example.net/account/12345/msgs/abc</instance>
  <balance>30</balance>
  <accounts>
    <i>https://example.net/account/12345</i>
    <i>https://example.net/account/67890</i>
  </accounts>
</problem>
```

Note that this format uses an XML Namespace. This is primarily to allow embedding it into other XML-based formats; it does not imply that it can or should be extended with elements or attributes in other namespaces. The RELAX NG schema explicitly only allows elements from the one namespace used in the XML format. Any

extension arrays and objects MUST be serialized into XML markup using only that namespace.

When using the XML format, it is possible to embed an XML Processing Instruction in the XML that instructs clients to transform the XML, using the referenced XSLT code [W3C.REC-xml-stylesheet-20101028]. If this code is transforming the XML into (X)HTML, then it is possible to serve the XML format, and yet have clients capable of performing the transformation display human-friendly (X)HTML that is rendered and displayed at the client. Note that when using this method, it is advisable to use XSLT 1.0, in order to maximize the number of clients capable of executing the XSLT code.

Appendix B. Using Problem Details with Other Formats

In some situations, it can be advantageous to embed Problem Details in formats other than those described here. For example, an API that uses HTML [W3C.REC-html5-20141028] might want to also use HTML for expressing its problem details.

Problem details can be embedded in other formats by either encapsulating one of the existing serializations (JSON or XML) into that format, or by translating the model of a Problem Detail (as specified in Section 3) into the format's conventions.

For example, in HTML, a problem could be embedded by encapsulating JSON in a script tag:

```
<script type="application/problem+json">
  {
   "type": "https://example.com/probs/out-of-credit",
   "title": "You do not have enough credit.",
   "detail": "Your current balance is 30, but that costs 50.",
   "instance": "/account/12345/msgs/abc",
   "balance": 30,
   "accounts": ["/account/12345",
                "/account/67890"]
  }
</script>
```

or by inventing a mapping into RDFa [W3C.REC-rdfa-core-20130822].

This specification does not make specific recommendations regarding embedding Problem Details in other formats; the appropriate way to embed them depends both upon the format in use and application of that format.

Authors' Addresses

Mark Nottingham
Akamai

Email: mnot@mnot.net
URI: http://www.mnot.net/

Erik Wilde

Email: erik.wilde@dret.net
URI: http://dret.net/netdret/

Network Working Group
Internet-Draft
Obsoletes: 3798 (if approved)
Updates: 2046, 3461 (if approved)
Intended status: Standards Track
Expires: June 4, 2017

T. Hansen, Ed.
AT&T Laboratories
A. Melnikov, Ed.
Isode Ltd

December 1, 2016

Message Disposition Notification
draft-ietf-appsawg-mdn-3798bis-16.txt

Abstract

This memo defines a MIME content-type that may be used by a mail user agent (MUA) or electronic mail gateway to report the disposition of a message after it has been successfully delivered to a recipient. This content-type is intended to be machine-processable. Additional message header fields are also defined to permit Message Disposition Notifications (MDNs) to be requested by the sender of a message. The purpose is to extend Internet Mail to support functionality often found in other messaging systems, such as X.400 and the proprietary "LAN-based" systems, and often referred to as "read receipts," "acknowledgements", or "receipt notifications." The intention is to do this while respecting privacy concerns, which have often been expressed when such functions have been discussed in the past.

Because many messages are sent between the Internet and other messaging systems (such as X.400 or the proprietary "LAN-based" systems), the MDN protocol is designed to be useful in a multi-protocol messaging environment. To this end, the protocol described in this memo provides for the carriage of "foreign" addresses, in addition to those normally used in Internet Mail. Additional attributes may also be defined to support "tunneling" of foreign notifications through Internet Mail.

This document obsoletes RFC 3798, moving it to Internet Standard. It also updates RFC 2046 (message/partial Media Type handling) and RFC 3461 (Original-Recipient header field generation requirement).

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/.

Copyright Notice

Table of Contents

1. Introduction

This memo defines a media type [RFC2046] for message disposition notifications (MDNs). An MDN can be used to notify the sender of a message of any of several conditions that may occur after successful delivery, such as display of the message contents, printing of the message, deletion (without display) of the message, or the recipient's refusal to provide MDNs. The "message/disposition-notification" content-type defined herein is intended for use within the framework of the "multipart/report" content type defined in RFC-REPORT [RFC6522].

This memo defines the format of the notifications and the RFC-MSGFMT [RFC5322] header fields used to request them.

This memo is an update to RFC 3798 and is intended to be published at Internet Standard Level.

1.1. Purposes

The MDNs defined in this memo are expected to serve several purposes:

a. Inform human beings of the disposition of messages after successful delivery, in a manner that is largely independent of human language;

b. Allow mail user agents to keep track of the disposition of messages sent, by associating returned MDNs with earlier message transmissions;

c. Convey disposition notification requests and disposition notifications between Internet Mail and "foreign" mail systems via a gateway;

d. Allow "foreign" notifications to be tunneled through a MIME-capable message system and back into the original messaging system that issued the original notification, or even to a third messaging system;

e. Allow language-independent, yet reasonably precise, indications of the disposition of a message to be delivered.

1.2. Requirements

These purposes place the following constraints on the notification protocol:

a. It must be readable by humans, and must be machine-parsable.

b. It must provide enough information to allow message senders (or their user agents) to unambiguously associate an MDN with the message that was sent and the original recipient address for which the MDN was issued (if such information is available), even if the message was forwarded to another recipient address.

c. It must also be able to describe the disposition of a message independent of any particular human language or of the terminology of any particular mail system.

d. The specification must be extensible in order to accommodate future requirements.

1.3. 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 RFC-KEYWORDS [RFC2119].

All syntax descriptions use the ABNF specified by RFC-MSGFMT [RFC5322], in which the lexical tokens (used below) are defined: "CRLF", "FWS", "CFWS", "field-name", "mailbox-list", "msg-id", and

"text". The following lexical tokens are defined in RFC-SMTP [RFC5321]: "atom".

2. Requesting Message Disposition Notifications

Message disposition notifications are requested by including a Disposition-Notification-To header field in the message containing one or more addresses specifying where dispositions should be sent. Further information to be used by the recipient's Mail User Agent (MUA) [RFC5598] in generating the MDN may be provided by also including Original-Recipient and/or Disposition-Notification-Options header fields in the message.

2.1. The Disposition-Notification-To Header

A request for the receiving user agent to issue message disposition notifications is made by placing a Disposition-Notification-To header field into the message. The syntax of the header field is

```
mdn-request-header = "Disposition-Notification-To" ":" mailbox-list CRLF
```

A Disposition-Notification-To header field can appear at most once in a message.

The presence of a Disposition-Notification-To header field in a message is merely a request for an MDN. The recipients' user agents are always free to silently ignore such a request.

An MDN MUST NOT itself have a Disposition-Notification-To header field. An MDN MUST NOT be generated in response to an MDN.

A user agent MUST NOT issue more than one MDN on behalf of each particular recipient. That is, once an MDN has been issued on behalf of a recipient, no further MDNs may be issued on behalf of that recipient by the same user agent, even if another disposition is performed on the message. However, if a message is forwarded, an MDN may have been issued for the recipient doing the forwarding and the recipient of the forwarded message may also cause an MDN to be generated.

It is also possible that if the same message is being accessed by multiple user agents (for example using POP3), then multiple dispositions might be generated for the same recipient. User agents SHOULD leverage support in the underlying message access protocol to prevent multiple MDNs from being generated. In particular, when the user agent is accessing the message using RFC-IMAP [RFC3501], it SHOULD implement the procedures specified in RFC-IMAP-MDN [RFC3503].

While Internet standards normally do not specify the behavior of user interfaces, it is strongly recommended that the user agent obtain the user's consent before sending an MDN. This consent could be obtained for each message through some sort of prompt or dialog box, or globally through the user's setting of a preference. The user might also indicate globally that MDNs are to never be sent. The purpose of obtaining user's consent is to protect user's privacy. The default value should be not to send MDNs.

MDNs MUST NOT be sent automatically if the address in the Disposition-Notification-To header field differs from the address in the Return-Path header field (see RFC-MSGFMT [RFC5322]). In this case, confirmation from the user MUST be obtained, if possible. If obtaining consent is not possible (e.g., because the user is not online at the time or the client is not an interactive email client), then an MDN MUST NOT be sent.

Confirmation from the user MUST be obtained (or no MDN sent) if there is no Return-Path header field in the message, or if there is more than one distinct address in the Disposition-Notification-To header field.

The comparison of the addresses is done using only the addr-spec (local-part "@" domain) portion, excluding any angle brackets, phrase and route. As prescribed by RFC 5322, the comparison is case-sensitive for the local-part and case-insensitive for the domain part. The local-part comparison SHOULD be done after performing local-part canonicalization (i.e. after removing the surrounding double-quote characters, if any, as well as any escaping "\" characters. (See RFC-MSGFMT [RFC5322] for more details.) Implementations MAY treat known domain aliases as equivalent for the purpose of comparison.

Note that use of subaddressing (see [RFC5233]) can result in a failure to match two local-parts and thus result in possible suppression of the MDN. This document doesn't recommend special handling for this case, as the receiving MUA can't reliably know whether or not the sender is using subaddressing.

If the message contains more than one Return-Path header field, the implementation may pick one to use for the comparison, or treat the situation as a failure of the comparison.

The reason for not automatically sending an MDN if the comparison fails or more than one address is specified is to reduce the possibility of mail loops and of MDNs being used for mail bombing.

It's especially important that a message that contains a Disposition-Notification-To header field also contain a Message-ID header field, to permit user agents to automatically correlate MDNs with their original messages.

If the request for message disposition notifications for some recipients and not others is desired, two copies of the message should be sent, one with a Disposition-Notification-To header field and one without. Many of the other header fields of the message (e.g., To, Cc) will be the same in both copies. The recipients in the respective message envelopes determine from whom message disposition notifications are requested and from whom they are not. If desired, the Message-ID header field may be the same in both copies of the message. Note that there are other situations (e.g., Bcc) in which it is necessary to send multiple copies of a message with slightly different header fields. The combination of such situations and the need to request MDNs for a subset of all recipients may result in more than two copies of a message being sent, some with a Disposition-Notification-To header field and some without.

If it is possible to determine that a recipient is a newsgroup, do not include a Disposition-Notification-To header field for that recipient. Similarly, if an existing message is resent or gatewayed to a newsgroup, the agent doing resending/gatewaying SHOULD strip the Disposition-Notification-To header field. See Section 5 for more discussion. Clients that see an otherwise valid Disposition-Notification-To header field in a newsgroup message SHOULD NOT generate an MDN.

2.2. The Disposition-Notification-Options Header

Extensions to this specification may require that information be supplied to the recipient's MUA for additional control over how and what MDNs are generated. The Disposition-Notification-Options header field provides an extensible mechanism for such information. The syntax of this header field is as follows:

```
Disposition-Notification-Options =
          "Disposition-Notification-Options" ":" [FWS]
                         disposition-notification-parameter-list CRLF

disposition-notification-parameter-list =
          disposition-notification-parameter
          *([FWS] ";" [FWS] disposition-notification-parameter)

disposition-notification-parameter = attribute [FWS] "="
          [FWS] importance [FWS] "," [FWS] value *([FWS] "," [FWS] value)

importance = "required" / "optional"

attribute = atom

value = word
```

A Disposition-Notification-Options header field can appear at most once in a message.

An importance of "required" indicates that interpretation of the disposition-notification-parameter is necessary for proper generation of an MDN in response to this request. An importance of "optional" indicates that an MUA that does not understand the meaning of this disposition-notification-parameter MAY generate an MDN in response anyway, ignoring the value of the disposition-notification-parameter.

No disposition-notification-parameter attribute names are defined in this specification. Attribute names may be defined in the future by later revisions or extensions to this specification. disposition-notification-parameter attribute names MUST be registered with the Internet Assigned Numbers Authority (IANA) using "Specification required" registration policy. The "X-" prefix has historically been used to denote unregistered "experimental" protocol elements, that are assumed not to become common use. Deployment experience of this and other protocols have shown that this assumption is often false. This document allows the use of the "X-" prefix primarily to allow the registration of attributes that are already in common use. The prefix has no meaning for new attributes. Its use in substantially new attributes may cause confusion and is therefore discouraged. (See Section 10 for a registration form.)

2.3. The Original-Recipient Header Field

Since electronic mail addresses may be rewritten while the message is in transit, it is useful for the original recipient address to be made available by the delivering Message Transfer Agent (MTA) [RFC5598]. The delivering MTA may be able to obtain this information

from the ORCPT parameter of the SMTP RCPT TO command, as defined in RFC-SMTP [RFC5321] and RFC-DSN-SMTP [RFC3461].

RFC-DSN-SMTP [RFC3461] is amended as follows: If the ORCPT information is available, the delivering MTA SHOULD insert an Original-Recipient header field at the beginning of the message (along with the Return-Path header field). The delivering MTA MAY delete any other Original-Recipient header fields that occur in the message. The syntax of this header field is as follows:

```
original-recipient-header =
          "Original-Recipient" ":" OWS address-type OWS ";" OWS generic-address
OWS

OWS      = [CFWS]
           ; Optional whitespace.
           ; MDN generators SHOULD use "*WSP"
           ; (typically a single space or nothing.
           ; It SHOULD be nothing at the end of a field),
           ; unless an RFC 5322 "comment" is required.
           ;
           ; MDN parsers MUST parse it as "[CFWS]".
```

The address-type and generic-address token are as specified in the description of the Original-Recipient field in Section 3.2.3.

The purpose of carrying the original recipient information and returning it in the MDN is to permit automatic correlation of MDNs with the original message on a per-recipient basis.

2.4. Use with the Message/Partial Media Type

The use of the header fields Disposition-Notification-To, Disposition-Notification-Options, and Original-Recipient with the MIME message/partial content type (RFC-MIME-MEDIA [RFC2046]]) requires further definition.

When a message is segmented into two or more message/partial fragments, the three header fields mentioned in the above paragraph SHOULD be placed in the "inner" or "enclosed" message (using the terms of RFC-MIME-MEDIA [RFC2046]). If these header fields are found in the header fields of any of the fragments, they are ignored.

When the multiple message/partial fragments are reassembled, the following applies. If these header fields occur along with the other header fields of a message/partial fragment message, they pertain to an MDN that will be generated for the fragment. If these header fields occur in the header fields of the "inner" or "enclosed" message (using the terms of RFC-MIME-MEDIA [RFC2046]), they pertain

to an MDN that will be generated for the reassembled message. Section 5.2.2.1 of RFC-MIME-MEDIA [RFC2046]) is amended to specify that, in addition to the header fields specified there, the three header fields described in this specification are to be appended, in order, to the header fields of the reassembled message. Any occurrences of the three header fields defined here in the header fields of the initial enclosing message MUST NOT be copied to the reassembled message.

3. Format of a Message Disposition Notification

A message disposition notification is a MIME message with a top-level content-type of multipart/report (defined in RFC-REPORT [RFC6522]). When multipart/report content is used to transmit an MDN:

a. The report-type parameter of the multipart/report content is "disposition-notification".

b. The first component of the multipart/report contains a human-readable explanation of the MDN, as described in RFC-REPORT [RFC6522].

c. The second component of the multipart/report is of content-type message/disposition-notification, described in Section 3.1 of this document.

d. If the original message or a portion of the message is to be returned to the sender, it appears as the third component of the multipart/report. The decision of whether or not to return the message or part of the message is up to the MUA generating the MDN. However, in the case of encrypted messages requesting MDNs, if the original message or a portion thereof is returned, it MUST be in its original encrypted form.

NOTE: For message disposition notifications gatewayed from foreign systems, the header fields of the original message may not be available. In this case, the third component of the MDN may be omitted, or it may contain "simulated" RFC-MSGFMT [RFC5322] header fields that contain equivalent information. In particular, it is very desirable to preserve the subject and date fields from the original message.

The MDN MUST be addressed (in both the message header field and the transport envelope) to the address(es) from the Disposition-

Notification-To header field from the original message for which the MDN is being generated.

The From header field of the MDN MUST contain the address of the person for whom the message disposition notification is being issued.

The envelope sender address (i.e., SMTP "MAIL FROM") of the MDN MUST be null (<>), specifying that no Delivery Status Notification messages nor other messages indicating successful or unsuccessful delivery are to be sent in response to an MDN.

A message disposition notification MUST NOT itself request an MDN. That is, it MUST NOT contain a Disposition-Notification-To header field.

The Message-ID header field (if present) for an MDN MUST be different from the Message-ID of the message for which the MDN is being issued.

A particular MDN describes the disposition of exactly one message for exactly one recipient. Multiple MDNs may be generated as a result of one message submission, one per recipient. However, due to the circumstances described in Section 2.1, it's possible that some of the recipients for whom MDNs were requested will not generate MDNs.

3.1. The message/disposition-notification Media Type

The message/disposition-notification Media Type is defined as follows:

Type name: message

Subtype name: disposition-notification

Required parameters: none

Optional parameters: none

Encoding considerations: "7bit" encoding is sufficient and MUST be used to maintain readability when viewed by non-MIME mail readers.

Security considerations: discussed in Section 6 of [RFCXXXX].

Interoperability considerations: none

Published specification: [RFCXXXX]

Applications that use this media type: Mail Transfer Agents and email clients that support multipart/report generation and/or parsing.

Fragment identifier considerations: N/A

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): none

File extension(s): .disposition-notification

Macintosh file type code(s): The 'TEXT' type code is suggested as files of this type are typically used for diagnostic purposes and suitable for analysis in a text editor. A uniform type identifier (UTI) of "public.utf8-email-message-header" is suggested. This type conforms to "public.plain-text".

Person & email address to contact for further information: ART Area Mailing List <art@ietf.org>

Intended usage: COMMON

Restrictions on usage: This media type contains textual data in the US-ASCII charset, which is always 7-bit.

Author: See the Authors' Addresses section of [RFCXXXX]

Change controller: IETF

Provisional registration? no

(While the 7bit restriction applies to the message/disposition-notification portion of the multipart/report content, it does not apply to the optional third portion of the multipart/report content.)

The message/disposition-notification report type for use in the multipart/report is "disposition-notification".

The body of a message/disposition-notification consists of one or more "fields" formatted according to the ABNF of RFC-MSGFMT [RFC5322] header "fields". The syntax of the message/disposition-notification content is as follows:

```
disposition-notification-content = [ reporting-ua-field CRLF ]
          [ mdn-gateway-field CRLF ]
          [ original-recipient-field CRLF ]
          final-recipient-field CRLF
          [ original-message-id-field CRLF ]
          disposition-field CRLF
          *( error-field CRLF )
          *( extension-field CRLF )

extension-field = extension-field-name ":" *([FWS] text)

extension-field-name = field-name
```

Note that the order of the above fields is recommended, but not fixed. Extension fields can appear anywhere.

3.1.1. General conventions for fields

Since these fields are defined according to the rules of RFC-MSGFMT [RFC5322], the same conventions for continuation lines and comments apply. Notification fields may be continued onto multiple lines by beginning each additional line with a SPACE or HTAB. Text that appears in parentheses is considered a comment and not part of the contents of that notification field. Field names are case-insensitive, so the names of notification fields may be spelled in any combination of upper and lower case letters. [RFC5322] comments in notification fields may use the "encoded-word" construct defined in RFC-MIME-HEADER [RFC2047].

3.1.2. "*-type" subfields

Several fields consist of a "-type" subfield, followed by a semi-colon, followed by "*text".
For these fields, the keyword used in the address-type or MTA-type subfield indicates the expected format of the address or MTA-name that follows.

The "-type" subfields are defined as follows:

a. An "address-type" specifies the format of a mailbox address. For example, Internet Mail addresses use the "rfc822" address-type. Other values can appear in this field as specified in the "Address Types" IANA subregistry established by RFC-DSN-FORMAT [RFC3464].

```
address-type = atom

atom = <The version from RFC 5321 (not from RFC 5322) is used in this document.>
```

b. An "MTA-name-type" specifies the format of a mail transfer agent name. For example, for an SMTP server on an Internet host, the MTA name is the domain name of that host, and the "dns" MTA-name-type is used. Other values can appear in this field as specified in the "MTA Name Types" IANA subregistry established by RFC-DSN-FORMAT [RFC3464].

```
mta-name-type = atom
```

Values for address-type and mta-name-type are case-insensitive. Thus, address-type values of "RFC822" and "rfc822" are equivalent.

The Internet Assigned Numbers Authority (IANA) maintains a registry of address-type and mta-name-type values, along with descriptions of the meanings of each, or a reference to one or more specifications that provide such descriptions. (The "rfc822" address-type is defined in RFC-DSN-SMTP [RFC3461].) Registration forms for address-type and mta-name-type appear in RFC-DSN-FORMAT [RFC3464].

3.2. Message/disposition-notification Content Fields

3.2.1. The Reporting-UA field

```
reporting-ua-field = "Reporting-UA" ":" OWS ua-name OWS [ ";" OWS ua-product OWS
 ]

ua-name = *text-no-semi

ua-product = *([FWS] text)

text-no-semi = %d1-9 /          ; "text" characters excluding NUL, CR,
               %d11 / %d12 / %d14-58 / %d60-127  ; LF, or semi-colon
```

The Reporting-UA field is defined as follows:

An MDN describes the disposition of a message after it has been delivered to a recipient. In all cases, the Reporting-UA is the MUA that performed the disposition described in the MDN.

The "Reporting-UA" field contains information about the MUA that generated the MDN, which is often used by servers to help identify the scope of reported interoperability problems, to work around or tailor responses to avoid particular MUA limitations, and for analytics regarding MUA or operating system use. A MUA SHOULD send a "Reporting-UA" field unless specifically configured not to do so.

If the reporting MUA consists of more than one component (e.g., a base program and plug-ins), this may be indicated by including a list of product names.

A reporting MUA SHOULD limit generated product identifiers to what is necessary to identify the product; a sender MUST NOT generate advertising or other nonessential information within the product identifier.

A reporting MUA SHOULD NOT generate a "Reporting-UA" field containing needlessly fine-grained detail and SHOULD limit the addition of subproducts by third parties. Overly long and detailed "Reporting-UA" field values increase the risk of a user being identified against their wishes ("fingerprinting").

Likewise, implementations are encouraged not to use the product tokens of other implementations in order to declare compatibility with them, as this circumvents the purpose of the field. If a MUA masquerades as a different MUA, recipients can assume that the user intentionally desires to see responses tailored for that identified MUA, even if they might not work as well for the actual MUA being used.

Example:

Reporting-UA: Foomail 97.1

3.2.2. The MDN-Gateway field

The MDN-Gateway field indicates the name of the gateway or MTA that translated a foreign (non-Internet) message disposition notification into this MDN. This field MUST appear in any MDN that was translated by a gateway from a foreign system into MDN format, and MUST NOT appear otherwise.

```
mdn-gateway-field = "MDN-Gateway" ":" OWS mta-name-type OWS ";" OWS mta-name OWS

mta-name = *text
```

For gateways into Internet Mail, the MTA-name-type will normally be "dns", and the mta-name will be the Internet domain name of the gateway.

3.2.3. Original-Recipient field

The Original-Recipient field indicates the original recipient address as specified by the sender of the message for which the MDN is being issued. For Internet Mail messages, the value of the Original-Recipient field is obtained from the Original-Recipient header field from the message for which the MDN is being generated. If there is an Original-Recipient header field in the message, or if information about the original recipient is reliably available some other way, then the Original-Recipient field MUST be included. Otherwise, the Original-Recipient field MUST NOT be included. If there is more than one Original-Recipient header field in the message, the MUA may choose the one to use, or act as if no Original-Recipient header field is present.

```
original-recipient-field =
          "Original-Recipient" ":" OWS address-type OWS ";" OWS generic-address
OWS

generic-address = *text
```

The address-type field indicates the type of the original recipient address. If the message originated within the Internet, the address-type field will normally be "rfc822", and the address will be according to the syntax specified in RFC-MSGFMT [RFC5322]. The value "unknown" should be used if the Reporting MUA cannot determine the type of the original recipient address from the message envelope. This address is the same as that provided by the sender and can be used to automatically correlate MDN reports with original messages on a per recipient basis.

3.2.4. Final-Recipient field

The Final-Recipient field indicates the recipient for which the MDN is being issued. This field MUST be present.

The syntax of the field is as follows:

```
final-recipient-field =
          "Final-Recipient" ":" OWS address-type OWS ";" OWS generic-address OWS
```

The generic-address subfield of the Final-Recipient field SHOULD contain the mailbox address of the recipient (which will be the same as the From header field of the MDN) as it was when the MDN was generated by the MUA.

> One example of when this field might not contain the final recipient address of the message is when an alias (e.g. "customer-support@example.com") forwards mail to a specific personal address (e.g. "bob@example.com"). Bob might want to be able to send MDNs, but not give away his personal email address. In this case the Final-Recipient field can be "customer-support@example.com" instead of "bob@example.com".

The Final-Recipient address may differ from the address originally provided by the sender, because it may have been transformed during forwarding and gatewaying into a totally unrecognizable mess. However, in the absence of the optional Original-Recipient field, the Final-Recipient field and any returned content may be the only information available with which to correlate the MDN with a particular message recipient.

The address-type subfield indicates the type of address expected by the reporting MTA in that context. Recipient addresses obtained via SMTP will normally be of address-type "rfc822", but can be other values from the "Address Types" subregistry of the "Delivery Status Notification (DSN) Types" IANA registry.

Since mailbox addresses (including those used in the Internet) may be case sensitive, the case of alphabetic characters in the address MUST be preserved.

3.2.5. Original-Message-ID field

The Original-Message-ID field indicates the message-ID of the message for which the MDN is being issued. It is obtained from the Message-ID header field of the message for which the MDN is issued. This field MUST be present if and only if the original message contained a Message-ID header field. The syntax of the field is as follows:

```
original-message-id-field =
          "Original-Message-ID" ":" msg-id
```

The msg-id token is as specified in RFC-MSGFMT [RFC5322].

3.2.6. Disposition field

The Disposition field indicates the action performed by the Reporting-MUA on behalf of the user. This field MUST be present.

The syntax for the Disposition field is:

```
disposition-field =
          "Disposition" ":" OWS disposition-mode OWS ";"
          OWS disposition-type
          [ OWS "/" OWS disposition-modifier
          *( OWS "," OWS disposition-modifier ) ] OWS

disposition-mode = action-mode OWS "/" OWS sending-mode

action-mode = "manual-action" / "automatic-action"

sending-mode = "MDN-sent-manually" / "MDN-sent-automatically"

disposition-type = "displayed" / "deleted" / "dispatched" /
          "processed"

disposition-modifier = "error" / disposition-modifier-extension

disposition-modifier-extension = atom
```

The disposition-mode, disposition-type, and disposition-modifier values may be spelled in any combination of upper and lower case US-ASCII characters.

3.2.6.1. Disposition modes

Disposition mode consists of 2 parts: action mode and sending mode.

The following action modes are defined:

"manual-action" The disposition described by the disposition type was a result of an explicit instruction by the user rather than some sort of automatically performed action. (This might include the case when the user has manually configured her MUA to automatically respond to valid MDN requests.) Unless prescribed otherwise in a particular mail environment, in order to preserve user's privacy, this MUST be the default for MUAs.

"automatic-action" The disposition described by the disposition type was a result of an automatic action, rather than an explicit instruction by the user for this message. This is typically generated by a Mail Delivery Agent (e.g. MDN generations by Sieve reject action [RFC5429], Fax-over-Email [RFC3249], Voice Messaging System (VPIM) [RFC3801] or upon delivery to a mailing list).

"Manual-action" and "automatic-action" are mutually exclusive. One or the other MUST be specified.

The following sending modes are defined:

"MDN-sent-manually" The user explicitly gave permission for this particular MDN to be sent. Unless prescribed otherwise in a particular mail environment, in order to preserve user's privacy, this MUST be the default for MUAs.

"MDN-sent-automatically" The MDN was sent because the MUA had previously been configured to do so automatically.

"MDN-sent-manually" and "MDN-sent-automatically" are mutually exclusive. One or the other MUST be specified.

3.2.6.2. Disposition types

The following disposition-types are defined:

"displayed" The message has been displayed by the MUA to someone reading the recipient's mailbox. There is no guarantee that the content has been read or understood.

"dispatched" The message has been sent somewhere in some manner (e.g., printed, faxed, forwarded) without necessarily having been previously displayed to the user. The user may or may not see the message later.

"processed" The message has been processed in some manner (i.e., by some sort of rules or server) without being displayed to the user. The user may or may not see the message later, or there may not even be a human user associated with the mailbox.

"deleted" The message has been deleted. The recipient may or may not have seen the message. The recipient might "undelete" the message at a later time and read the message.

3.2.6.3. Disposition modifiers

Only the extension disposition modifiers is defined:

disposition-modifier-extension
Disposition modifiers may be defined in the future by later revisions or extensions to this specification. MDN disposition value names MUST be registered with the Internet Assigned Numbers Authority (IANA) using "Specification required" registration policy. (See Section 10 for a registration form.) MDNs with disposition modifier names not understood by the receiving MUA MAY be silently ignored or placed in the user's mailbox without special interpretation. They MUST NOT cause any error message to be sent to the sender of the MDN.

It is not required that an MUA be able to generate all of the possible values of the Disposition field.

A user agent MUST NOT issue more than one MDN on behalf of each particular recipient. That is, once an MDN has been issued on behalf of a recipient, no further MDNs may be issued on behalf of that recipient, even if another disposition is performed on the message. However, if a message is forwarded, a "dispatched" MDN MAY be issued for the recipient doing the forwarding and the recipient of the forwarded message may also cause an MDN to be generated.

3.2.7. Error Field

The Error field is used to supply additional information in the form of text messages when the "error" disposition modifier appear. The syntax is as follows:

error-field = "Error" ":" *([FWS] text)

Note that syntax of these header fields doesn't include comments, so "encoded-word" construct defined in RFC-MIME-HEADER [RFC2047] can't be used to convey non ASCII text. Application that need to convey non ASCII text in these fields should consider implementing message/global-disposition-notification media type specified in [RFC6533] instead of this specification.

3.3. Extension-fields

Additional MDN fields may be defined in the future by later revisions or extensions to this specification. MDN field names MUST be registered with the Internet Assigned Numbers Authority (IANA) using "Specification required" registration policy. (See Section 10 for a registration form.) MDN Extension-fields may be defined for the following reasons:

a. To allow additional information from foreign disposition reports to be tunneled through Internet MDNs. The names of such MDN fields should begin with an indication of the foreign environment name (e.g., X400-Physical-Forwarding-Address).

b. To allow transmission of diagnostic information that is specific to a particular mail user agent (MUA). The names of such MDN fields should begin with an indication of the MUA implementation that produced the MDN (e.g., Foomail-information).

4. Timeline of events

The following timeline shows when various events in the processing of a message and generation of MDNs take place:

-- User composes message

-- User tells MUA to send message.

-- MUA passes message to Mail Submission Agent (MSA), original recipient information passed along.

-- MSA sends message to next MTA.

-- Final MTA receives message.

-- Final MTA delivers message to recipient's mailbox (possibly generating a Delivery Status Notification (DSN)).

-- (Recipient's) MUA discovers a new message in recipient's mailbox and decides whether an MDN should be generated. If the MUA has information that an MDN has already been generated for this message, no further MDN processing described below is performed. If MUA decides that no MDN can be generated, no further MDN processing described below is performed.

-- MUA performs automatic processing and might generate corresponding MDNs ("dispatched", "processed" or "deleted" disposition type with "automatic-action" and "MDN-sent-automatically" disposition modes). The MUA remembers that an MDN was generated.

-- MUA displays list of messages to user.

-- User selects a message and requests that some action be performed on it.

-- MUA performs requested action; if an automatic MDN has not already been generated, with user's permission, sends an appropriate MDN ("displayed", "dispatched", "processed", or "deleted" disposition type, with "manual-action" and "MDN-sent-manually" or "MDN-sent-automatically" disposition mode). The MUA remembers that an MDN was generated.

-- User possibly performs other actions on message, but no further MDNs are generated.

5. Conformance and Usage Requirements

An MUA or gateway conforms to this specification if it generates MDNs according to the protocol defined in this memo. It is not necessary to be able to generate all of the possible values of the Disposition field.

MUAs and gateways MUST NOT generate the Original-Recipient field of an MDN unless the mail protocols provide the address originally

specified by the sender at the time of submission. Ordinary SMTP does not make that guarantee, but the SMTP extension defined in RFC-DSN-SMTP [RFC3461] permits such information to be carried in the envelope if it is available. The Original-Recipient header field defined in this document provides a way for the MTA to pass the original recipient address to the MUA.

Each sender-specified recipient address may result in more than one MDN. If an MDN is requested for a recipient that is forwarded to multiple recipients of an "alias" (as defined in RFC-DSN-SMTP [RFC3461], section 6.2.7.3), each of the recipients may issue an MDN.

Successful distribution of a message to a mailing list exploder or gateway to Usenet newsgroup SHOULD be considered the final disposition of the message. A mailing list exploder MAY issue an MDN with a disposition type of "processed" and disposition modes of "automatic-action" and "MDN-sent-automatically" indicating that the message has been forwarded to the list. In this case, the request for MDNs is not propagated to the members of the list.

Alternatively (if successful distribution of a message to a mailing list exploder/Usenet newsgroup is not considered the final disposition of the message), the mailing list exploder can issue no MDN and propagate the request for MDNs to all members of the list. The latter behavior is not recommended for any but small, closely knit lists, as it might cause large numbers of MDNs to be generated and may cause confidential subscribers to the list to be revealed. The mailing list exploder can also direct MDNs to itself, correlate them, and produce a report to the original sender of the message.

This specification places no restrictions on the processing of MDNs received by user agents or mailing lists.

6. Security Considerations

The following security considerations apply when using MDNs:

6.1. Forgery

MDNs can be (and are, in practice) forged as easily as ordinary Internet electronic mail. User agents and automatic mail handling facilities (such as mail distribution list exploders) that wish to make automatic use of MDNs should take appropriate precautions to minimize the potential damage from denial-of-service attacks.

Security threats related to forged MDNs include the sending of:

a. A falsified disposition notification when the indicated disposition of the message has not actually occurred,

b. Unsolicited MDNs

Similarly, a forged spam or phishing email message can contain Disposition-Notification-To header field that can trick the recipient to send an MDN. MDN processing should only be invoked once authenticity of email message is verified.

6.2. Privacy

Another dimension of security is privacy. There may be cases in which a message recipient does not wish the disposition of messages addressed to him to be known, or is concerned that the sending of MDNs may reveal other sensitive information (e.g., when the message was read, using which email client, which OS was used). In this situation, it is acceptable for the MUA to silently ignore requests for MDNs.

If the Disposition-Notification-To header field is passed on unmodified when a message is distributed to the subscribers of a mailing list, the subscribers to the list may be revealed to the sender of the original message by the generation of MDNs.

Headers of the original message returned in part 3 of the multipart/report, as well as content of the message/disposition-notification part could reveal confidential information about host names and/or network topology inside a firewall.

Disposition mode (Section 3.2.6.1) can leak information about recipient's MUA configuration, in particular whether MDNs are acknowledged manually or automatically. If this is a concern, MUAs can return "manual-action/MDN-sent-manually" disposition mode in generated MDNs.

In general, any optional MDN field may be omitted if the Reporting MUA site or user determines that inclusion of the field would impose too great a compromise of site confidentiality. The need for such confidentiality must be balanced against the utility of the omitted information in MDNs.

In some cases, someone with access to the message stream may use the MDN request mechanism to monitor the mail reading habits of a target. If the target is known to generate MDN reports, they could add a disposition-notification-to field containing the envelope from

address. This risk can be minimized by not sending MDN's automatically.

6.2.1. Disclosure of Product Information

The "Reporting-UA" field (Section 3.2.1) User-Agent (Section 5.5.3) and header fields often reveal information about the respective sender's software systems. In theory, this can make it easier for an attacker to exploit known security holes; in practice, attackers tend to try all potential holes regardless of the apparent software versions being used. Also note that the "Reporting-UA" field doesn't provide any new information in comparison to the "User-Agent" and/or (undocumented) "X-Mailer" header fields used by many MUAs.

6.2.2. MUA Fingerprinting

The "Reporting-UA" field (Section 3.2.1) might contain enough information to uniquely identify a specific device, usually when combined with other characteristics, particularly if the user agent sends excessive details about the user's system or extensions. Even when the guidance in Section 3.2.1 is followed to avoid fingerprinting, other sources of unique information may still be present, such as the Accept-Language header fields.

6.3. Non-Repudiation

MDNs do not provide non-repudiation with proof of delivery. Within the framework of today's Internet Mail, the MDNs defined in this document provide valuable information to the mail user; however, MDNs cannot be relied upon as a guarantee that a message was or was not seen by the recipient. Even if MDNs are not actively forged, they may be lost in transit. The recipient may bypass the MDN issuing mechanism in some manner.

One possible solution for this purpose can be found in RFC-SEC-SERVICES [RFC2634].

6.4. Mail Bombing

The MDN request mechanism introduces an additional way of mailbombing a mailbox. The MDN request notification provides an address to which MDN's should be sent. It is possible for an attacking agent to send a potentially large set of messages to otherwise unsuspecting third party recipients with a false "disposition-notification-to:" address. Automatic, or simplistic processing of such requests would result in a flood of MDN notifications to the target of the attack. Additionally, as generated MDN notifications can include full content of messages that caused them and thus they can be bigger than such

messages, they can be used for bandwidth amplification attacks. Such an attack could overrun the storage capacity of the targeted mailbox and/or of the mail transport system, and deny service.

For that reason, MDN's SHOULD NOT be sent automatically where the "disposition-notification-to:" address is different from the SMTP "MAIL FROM" address (which is carried in the Return-Path header field). See Section 2.1 for further discussion.

7. Collected ABNF Grammar

NOTE: The following lexical tokens are defined in RFC-MSGFMT [RFC5322]: CRLF, FWS, CFWS, field-name, mailbox-list, msg-id, text, comment, word. The following lexical tokens are defined in RFC-SMTP [RFC5321]: atom. (Note that RFC-MSGFMT [RFC5322] also defines "atom", but the version from RFC-SMTP [RFC5321] is more restrictive and this more restrictive version is used in this document.) "encoded-word" construct defined in RFC-MIME-HEADER [RFC2047] is allowed everywhere where RFC-MSGFMT [RFC5322] "comment" is used, for example in CFWS.

```
OWS          = [CFWS]
                 ; Optional whitespace.
                 ; MDN generators SHOULD use "*WSP"
                 ; (typically a single space or nothing.
                 ; It SHOULD be nothing at the end of a field),
                 ; unless an RFC 5322 "comment" is required.
                 ;
                 ; MDN parsers MUST parse it as "[CFWS]".

Message header fields:
   mdn-request-header =
          "Disposition-Notification-To" ":" mailbox-list CRLF

   Disposition-Notification-Options =
          "Disposition-Notification-Options" ":" [FWS]
                    disposition-notification-parameter-list CRLF

   disposition-notification-parameter-list =
                    disposition-notification-parameter
                    *([FWS] ";" [FWS] disposition-notification-parameter)

   disposition-notification-parameter = attribute [FWS] "=" [FWS]
                    importance [FWS] "," [FWS] value *([FWS] "," [FWS] value)

   importance = "required" / "optional"

   attribute = atom
```

```
   value = word

   original-recipient-header =
          "Original-Recipient" ":" OWS address-type OWS ";" OWS generic-address
OWS CRLF

Report content:
   disposition-notification-content =
          [ reporting-ua-field CRLF ]
          [ mdn-gateway-field CRLF ]
          [ original-recipient-field CRLF ]
          final-recipient-field CRLF
          [ original-message-id-field CRLF ]
          disposition-field CRLF
          *( error-field CRLF )
          *( extension-field CRLF )

   address-type = atom

   mta-name-type = atom

   reporting-ua-field = "Reporting-UA" ":" OWS ua-name OWS [ ";" OWS ua-product
OWS ]

   ua-name = *text-no-semi

   ua-product = *([FWS] text)

   text-no-semi = %d1-9 /        ; "text" characters excluding NUL, CR,
           %d11 / %d12 / %d14-58 / %d60-127      ; LF, or semi-colon

   mdn-gateway-field = "MDN-Gateway" ":" OWS mta-name-type OWS ";" OWS mta-name

   mta-name = *text

   original-recipient-field =
          "Original-Recipient" ":" OWS address-type OWS ";" OWS generic-address
OWS

   generic-address = *text

   final-recipient-field =
          "Final-Recipient" ":" OWS address-type OWS ";" OWS generic-address OWS

   original-message-id-field = "Original-Message-ID" ":" msg-id

   disposition-field =
          "Disposition" ":" OWS disposition-mode OWS ";"
          OWS disposition-type
          [ OWS "/" OWS disposition-modifier
          *( OWS "," OWS disposition-modifier ) ] OWS
```

```
disposition-mode = action-mode OWS "/" OWS sending-mode

action-mode = "manual-action" / "automatic-action"

sending-mode = "MDN-sent-manually" / "MDN-sent-automatically"

disposition-type = "displayed" / "deleted" / "dispatched" /
        "processed"

disposition-modifier = "error" / disposition-modifier-extension

disposition-modifier-extension = atom

error-field = "Error" ":" *([FWS] text)

extension-field = extension-field-name ":" *([FWS] text)

extension-field-name = field-name
```

8. Guidelines for Gatewaying MDNs

NOTE: This section provides non-binding recommendations for the construction of mail gateways that wish to provide semi-transparent disposition notifications between the Internet and another electronic mail system. Specific MDN gateway requirements for a particular pair of mail systems may be defined by other documents.

8.1. Gatewaying from other mail systems to MDNs

A mail gateway may issue an MDN to convey the contents of a "foreign" disposition notification over Internet Mail. When there are appropriate mappings from the foreign notification elements to MDN fields, the information may be transmitted in those MDN fields. Additional information (such as might be needed to tunnel the foreign notification through the Internet) may be defined in extension MDN fields. (Such fields should be given names that identify the foreign mail protocol, e.g., X400-* for X.400 protocol elements).

The gateway must attempt to supply reasonable values for the Reporting-UA, Final-Recipient, and Disposition fields. These will normally be obtained by translating the values from the foreign notification into their Internet-style equivalents. However, some loss of information is to be expected.

The sender-specified recipient address and the original message-id, if present in the foreign notification, should be preserved in the Original-Recipient and Original-Message-ID fields.

The gateway should also attempt to preserve the "final" recipient address from the foreign system. Whenever possible, foreign protocol elements should be encoded as meaningful printable ASCII strings.

For MDNs produced from foreign disposition notifications, the name of the gateway MUST appear in the MDN-Gateway field of the MDN.

8.2. Gatewaying from MDNs to other mail systems

It may be possible to gateway MDNs from the Internet into a foreign mail system. The primary purpose of such gatewaying is to convey disposition information in a form that is usable by the destination system. A secondary purpose is to allow "tunneling" of MDNs through foreign mail systems in case the MDN may be gatewayed back into the Internet.

In general, the recipient of the MDN (i.e., the sender of the original message) will want to know, for each recipient: the closest available approximation to the original recipient address, and the disposition (displayed, printed, etc.).

If possible, the gateway should attempt to preserve the Original-Recipient address and Original-Message-ID (if present) in the resulting foreign disposition report.

If it is possible to tunnel an MDN through the destination environment, the gateway specification may define a means of preserving the MDN information in the disposition reports used by that environment.

8.3. Gatewaying of MDN-requests to other mail systems

By use of the separate disposition-notification-to request header field, this specification offers a richer functionality than most, if not all, other email systems. In most other email systems, the notification recipient is identical to the message sender as indicated in the "from" address. There are two interesting cases when gatewaying into such systems:

1. If the address in the disposition-notification-to header field is identical to the address in the SMTP "MAIL FROM", the expected behavior will result, even if the disposition-notification-to information is lost. Systems should propagate the MDN request.

2. If the address in the disposition-notification-to header field is different from the address in the SMTP "MAIL FROM", gatewaying

into a foreign system without a separate notification address will result in unintended behavior. This is especially important when the message arrives via a mailing list expansion software that may specifically replace the SMTP "MAIL FROM" address with an alternate address. In such cases, the MDN request should not be gatewayed and should be silently dropped. This is consistent with other forms of non-support for MDN.

9. Example

NOTE: This example is provided as illustration only, and is not considered part of the MDN protocol specification. If the example conflicts with the protocol definition above, the example is wrong.

Likewise, the use of *-type subfield names or extension fields in this example is not to be construed as a definition for those type names or extension fields.

This is an MDN issued after a message has been displayed to the user of an Internet Mail user agent.

```
Date: Wed, 20 Sep 1995 00:19:00 (EDT) -0400
From: Joe Recipient <Joe_Recipient@example.com>
Message-Id: <199509200019.12345@example.com>
Subject: Disposition notification
To: Jane Sender <Jane_Sender@example.org>
MIME-Version: 1.0
Content-Type: multipart/report; report-type=disposition-notification;
   boundary="RAA14128.773615765/example.com"

--RAA14128.773615765/example.com

The message sent on 1995 Sep 19 at 13:30:00 (EDT) -0400 to Joe
Recipient <Joe_Recipient@example.com> with subject "First draft of
report" has been displayed.
This is no guarantee that the message has been read or understood.

--RAA14128.773615765/example.com
content-type: message/disposition-notification

Reporting-UA: joes-pc.cs.example.com; Foomail 97.1
Original-Recipient: rfc822;Joe_Recipient@example.com
Final-Recipient: rfc822;Joe_Recipient@example.com
Original-Message-ID: <199509192301.23456@example.org>
Disposition: manual-action/MDN-sent-manually; displayed

--RAA14128.773615765/example.com
content-type: message/rfc822

[original message optionally goes here]

--RAA14128.773615765/example.com--
```

10. IANA Considerations

There are two actions for IANA:

1. IANA is asked to update the registration template for the message/disposition-notification media type to the one in Section 3.1 of this document, and to update the reference for that media type to point to this document instead of to RFC 3798.

2. The registries specified here already exist, and this section is updating their documentation. IANA is asked to change the reference document for the three Message Disposition Notification Parameters registries to point to this document instead of to RFC 3798.

This document specifies three types of parameters that must be registered with the Internet Assigned Numbers Authority (IANA). All of them use [RFC5226] "Specification required" IANA registration policy.

The forms below are for use when registering a new disposition-notification-parameter name for the Disposition-Notification-Options header field, a new disposition modifier name, or a new MDN extension field. Each piece of information required by a registration form may be satisfied either by providing the information on the form itself, or by including a reference to a published, publicly available specification that includes the necessary information. IANA MAY reject registrations because of incomplete registration forms or incomplete specifications.

To register, complete the following applicable form and send it via electronic mail to <IANA@IANA.ORG>.

10.1. Disposition-Notification-Options header field disposition-notification-parameter names

A registration for a Disposition-Notification-Options header field disposition-notification-parameter name MUST include the following information:

a. The proposed disposition-notification-parameter name.

b. The syntax for disposition-notification-parameter values, specified using BNF, ABNF, regular expressions, or other non-ambiguous language.

c. If disposition-notification-parameter values are not composed entirely of graphic characters from the US-ASCII repertoire, a specification for how they are to be encoded as graphic US-ASCII characters in a Disposition-Notification-Options header field.

d. A reference to a permanent and readily available public specification that describes the semantics of the disposition-notification-parameter values.

10.2. Disposition modifier names

A registration for a disposition-modifier name (used in the Disposition field of a message/disposition-notification) MUST include the following information:

a. The proposed disposition-modifier name.

b. A reference to a permanent and readily available public specification that describes the semantics of the disposition modifier.

10.3. MDN extension field names

A registration for an MDN extension-field name MUST include the following information:

a. The proposed extension field name.

b. The syntax for extension values, specified using BNF, ABNF, regular expressions, or other non-ambiguous language.

c. If extension-field values are not composed entirely of graphic characters from the US-ASCII repertoire, a specification for how they are to be encoded as graphic US-ASCII characters in a Disposition-Notification-Options header field.

d. A reference to a permanent and readily available public specification that describes the semantics of the extension field.

11. Acknowledgements

The contributions of Bruce Lilly, Alfred Hoenes, Barry Leiba, Ben Campbell, Pete Resnick, Donald Eastlake and Alissa Cooper are gratefully acknowledged for this revision.

The contributions of Roger Fajman and Greg Vaudreuil to earlier versions of this document are also gratefully acknowledged.

12. References

12.1. Normative References

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, DOI 10.17487/RFC5321, October 2008, <http://www.rfc-editor.org/info/rfc5321>.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <http://www.rfc-editor.org/info/rfc5322>.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <http://www.rfc-editor.org/info/rfc2045>.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <http://www.rfc-editor.org/info/rfc2046>.

[RFC2047] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, DOI 10.17487/RFC2047, November 1996, <http://www.rfc-editor.org/info/rfc2047>.

[RFC6522] Kucherawy, M., Ed., "The Multipart/Report Media Type for the Reporting of Mail System Administrative Messages", STD 73, RFC 6522, DOI 10.17487/RFC6522, January 2012, <http://www.rfc-editor.org/info/rfc6522>.

[RFC3461] Moore, K., "Simple Mail Transfer Protocol (SMTP) Service Extension for Delivery Status Notifications (DSNs)", RFC 3461, DOI 10.17487/RFC3461, January 2003, <http://www.rfc-editor.org/info/rfc3461>.

[RFC3464] Moore, K. and G. Vaudreuil, "An Extensible Message Format for Delivery Status Notifications", RFC 3464, DOI 10.17487/RFC3464, January 2003, <http://www.rfc-editor.org/info/rfc3464>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3503] Melnikov, A., "Message Disposition Notification (MDN) profile for Internet Message Access Protocol (IMAP)", RFC 3503, DOI 10.17487/RFC3503, March 2003, <http://www.rfc-editor.org/info/rfc3503>.

12.2. Informative References

[RFC2634] Hoffman, P., Ed., "Enhanced Security Services for S/MIME", RFC 2634, DOI 10.17487/RFC2634, June 1999, <http://www.rfc-editor.org/info/rfc2634>.

[RFC3249] Cancio, V., Moldovan, M., Tamura, H., and D. Wing, "Implementers Guide for Facsimile Using Internet Mail", RFC 3249, DOI 10.17487/RFC3249, September 2002, <http://www.rfc-editor.org/info/rfc3249>.

[RFC3501] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, DOI 10.17487/RFC3501, March 2003, <http://www.rfc-editor.org/info/rfc3501>.

[RFC3801] Vaudreuil, G. and G. Parsons, "Voice Profile for Internet Mail - version 2 (VPIMv2)", RFC 3801, DOI 10.17487/RFC3801, June 2004, <http://www.rfc-editor.org/info/rfc3801>.

[RFC5233] Murchison, K., "Sieve Email Filtering: Subaddress Extension", RFC 5233, DOI 10.17487/RFC5233, January 2008, <http://www.rfc-editor.org/info/rfc5233>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC5429] Stone, A., Ed., "Sieve Email Filtering: Reject and Extended Reject Extensions", RFC 5429, DOI 10.17487/RFC5429, March 2009, <http://www.rfc-editor.org/info/rfc5429>.

[RFC5598] Crocker, D., "Internet Mail Architecture", RFC 5598, DOI 10.17487/RFC5598, July 2009, <http://www.rfc-editor.org/info/rfc5598>.

[RFC6533] Hansen, T., Ed., Newman, C., and A. Melnikov, "Internationalized Delivery Status and Disposition Notifications", RFC 6533, DOI 10.17487/RFC6533, February 2012, <http://www.rfc-editor.org/info/rfc6533>.

Appendix A. Changes from RFC 3798

Changed IANA registration for different subregistries to "Specification Required" to match what is already used by IANA.

Updated IANA registration template for message/disposition-notification.

"X-" fields no longer reserved for experimental use and can now be registered in compliance with RFC 6648.

Fixed the default MTA-name-type used in "MDN-Gateway" to be "dns".

Strengthen requirements on obtaining user consent in order to protect user privacy.

Removed discussion of using source routes with MDNs, as source route is a deprecated Email feature.

The values of "dispatched" and "processed" were lost from the ABNF for "disposition-type". (Erratum #691)

Because the warning disposition modifier was previously removed, warning-field has also been removed. (Erratum #692)

Because the failed disposition type was previously removed, failure-field has also been removed.

The ABNF for ua-name and ua-product included semi-colon, which could not be distinguished from *text in the production. The ua-name was restricted to not include semi-colon. Semi-colon can still appear in the ua-product.

Removed recommendation to include the MUA DNS host name in the "Reporting-UA" MDN field.

The ABNF did not indicate all places that whitespace was allowable, in particular folding whitespace, although all implementations allow whitespace and folding in the header fields just like any other RFC5322 [RFC5322]-formatted header field. There were also a number of places in the ABNF that inconsistently permitted comments and whitespace in one leg of the production and not another. The ABNF now specifies FWS and CFWS in several places that should have already been specified by the grammar.

Extension-field was defined in the collected grammar but not in the main text.

The comparison of mailboxes in Disposition-Notification-To to the Return-Path addr-spec was clarified.

The use of the grammar production "parameter" was confusing with the RFC2045 [RFC2045] production of the same name, as well as other uses of the same term. These have been clarified.

A clarification was added on the extent of the 7bit nature of MDNs.

Uses of the terms "may" and "might" were clarified.

A clarification was added on the order of the fields in the message/disposition-notification content.

Authors' Addresses

Tony Hansen (editor)
AT&T Laboratories
200 Laurel Ave. South
Middletown, NJ 07748
USA

Email: tony@att.com

Alexey Melnikov (editor)
Isode Ltd
14 Castle Mews
Hampton, Middlesex TW12 2NP
UK

Email: Alexey.Melnikov@isode.com

APPSAWG
Internet-Draft
Obsoletes: 2388 (if approved)
Intended status: Standards Track
Expires: October 12, 2015

L. Masinter
Adobe

April 10, 2015

Returning Values from Forms: multipart/form-data
draft-ietf-appsawg-multipart-form-data-11

Abstract

This specification defines the multipart/form-data Internet Media Type, which can be used by a wide variety of applications and transported by a wide variety of protocols as a way of returning a set of values as the result of a user filling out a form. It obsoletes RFC 2388.

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."

This Internet-Draft will expire on October 12, 2015.

Table of Contents

1. Introduction

In many applications, it is possible for a user to be presented with a form. The user will fill out the form, including information that is typed, generated by user input, or included from files that the user has selected. When the form is filled out, the data from the form is sent from the user to the receiving application.

The definition of "multipart/form-data" is derived from one of those applications, originally set out in [RFC1867] and subsequently incorporated into HTML 3.2 [W3C.REC-html32-19970114], where forms are expressed in HTML, and in which the form data is sent via HTTP or

electronic mail. This representation is widely implemented in numerous web browsers and web servers.

However, "multipart/form-data" is also used for forms that are presented using representations other than HTML (spreadsheets, PDF, etc.), and for transport using means other than electronic mail or HTTP; it is used in distributed applications which do not involve forms at all, or do not have users filling out the form. For this reason, this document defines a general syntax and semantics independent of the application for which it is used, with specific rules for web applications noted in context.

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 BCP 14, RFC 2119 [RFC2119].

2. percent-encoding option

Within this specification, "percent-encoding" (as defined in [RFC3986]) is offered as a possible way of encoding characters in file names that are otherwise disallowed, including non-ASCII characters, spaces, control characters and so forth. The encoding is created replacing each non-ASCII or disallowed character with a sequence, where each byte of the UTF-8 encoding of the character is represented by a percent-sign (%) followed by the (case-insensitive) hexadecimal of that byte.

3. Advice for Forms and Form Processing

The representation and interpretation of forms and the nature of form processing is not specified by this document. However, for forms and form-processing that result in generation of multipart/form-data, some suggestions are included.

In a form, there is generally a sequence of fields, where each field is expected to be supplied with a value, e.g. by a user who fills out the form. Each field has a name. After a form has been filled out, and the form's data is "submitted": the form processing results in a set of values for each field-- the "form data".

In forms that work with multipart/form-data, field names could be arbitrary Unicode strings; however, restricting field names to ASCII will help avoid some interoperability issues (see Section 5.1).

Within a given form, ensuring field names are unique is also helpful. Some fields may have default values or presupplied values in the form itself. Fields with presupplied values might be hidden or invisible;

this allows using generic processing for form data from a variety of actual forms.

4. Definition of multipart/form-data

The media-type "multipart/form-data" follows the model of multipart MIME data streams as specified in [RFC2046] Section 5.1; changes are noted in this document.

A "multipart/form-data" body contains a series of parts, separated by a boundary.

4.1. Boundary parameter of multipart/form-data

As with other multipart types, the parts are delimited with a boundary delimiter, constructed using CRLF, "--", the value of the boundary parameter. The boundary is supplied as a "boundary" parameter to the "multipart/form-data" type. As noted in [RFC2046] Section 5.1, the boundary delimiter MUST NOT appear inside any of the encapsulated parts, and it is often necessary to enclose the boundary parameter values in quotes on the Content-type line.

4.2. Content-Disposition header for each part

Each part MUST contain a "content-disposition" header [RFC2183] and where the disposition type is "form-data". The "content-disposition" header MUST also contain an additional parameter of "name"; the value of the "name" parameter is the original field name from the form (possibly encoded; see Section 5.1). For example, a part might contain a header:

```
Content-Disposition: form-data; name="user"
```

with the body of the part containing the form data of the "user" field.

4.3. filename attribute of content-distribution part header

For form data that represents the content of a file, a name for the file SHOULD be supplied as well, by using a "filename" parameter of the "content-disposition" header. The file name isn't mandatory for cases where the file name isn't available or is meaningless or private; this might result, for example, from selection or drag-and-drop or where the form data content is streamed directly from a device.

If a filename parameter is supplied, the requirements of [RFC2183] Section 2.3 for "receiving MUA" apply to recievers of "multipart/

form-data" as well: Do not use the file name blindly, check and possibly change to match local filesystem conventions if applicable, do not use directory path information that may be present.

In most multipart types, the MIME headers in each part are restricted to US-ASCII; for compatibility with those systems, file names normally visible to users MAY be encoded using the percent-encoding method in Section 2, following how a "file:" URI [I-D.ietf-appsawg-file-scheme] might be encoded.

NOTE: The encoding method described in [RFC5987], which would add a "filename*" paramter to the "Content-Disposition" header, MUST NOT be used.

Some commonly deployed systems use multipart/form-data with file names directly encoded including octets outside the US-ASCII range. The encoding used for the file names is typically UTF-8, although HTML forms will use the charset associated with the form.

4.4. Multiple files for one form field

The form data for a form field might include multiple files.

[RFC2388] suggested that multiple files for a single form field be transmitted using a nested multipart/mixed part. This usage is deprecated.

To match widely deployed implementations, multiple files MUST be sent by supplying each file in a separate part, but all with the same "name" parameter.

Receiving applications intended for wide applicability (e.g. multipart/form-data parsing libraries) SHOULD also support the older method of supplying multiple files.

4.5. Content-Type header for each part

Each part MAY have an (optional) "content-type", which defaults to "text/plain". If the contents of a file are to be sent, the file data SHOULD be labeled with an appropriate media type, if known, or "application/octet-stream".

4.6. The charset parameter for text/plain form data

In the case where the form data is text, the charset parameter for the "text/plain" Content-Type MAY be used to indicate the character encoding used in that part. For example, a form with a text field in

which a user typed "Joe owes <eu>100" where <eu> is the Euro symbol might have form data returned as:

```
--AaB03x
content-disposition: form-data; name="field1"
content-type: text/plain;charset=UTF-8
content-transfer-encoding: quoted-printable

Joe owes =E2=82=AC100.
--AaB03x
```

In practice, many widely deployed implementations do not supply a charset parameter in each part, but, rather, they rely on the notion of a "default charset" for a multipart/form-data instance. Subsequent sections will explain how the default charset is established.

4.7. The _charset_ field for default charset

Some form processing applications (including HTML) have the convention that the value of a form entry with entry name "_charset_" and type "hidden" is automatically set when the form is opened; the value is used as the default charset of text field values (see form-charset in Section 5.1.2). In such cases, the value of the default charset for each text/plain part without a charset parameter is the supplied value. For example:

```
--AaB03x
content-disposition: form-data; name="_charset_"

iso-8859-1
--AaB03x--
content-disposition: form-data; name="field1"

...text encoded in iso-8859-1 ...
AaB03x--
```

4.8. Content-Transfer-Encoding deprecated

Previously, it was recommended that senders use a "Content-Transfer-Encoding" encoding (such as "quoted-printable") for each non-ASCII part of a multipart/form-data body, because that would allow use in transports that only support a "7BIT" encoding. This use is deprecated for use in contexts that support binary data such as HTTP. Senders SHOULD NOT generate any parts with a "Content-Transfer-Encoding" header.

Currently, no deployed implementations that send such bodies have been discovered.

4.9. Other Content- headers

The "multipart/form-data" media type does not support any MIME headers in the parts other than Content-Type, Content-Disposition, and (in limited circumstances) Content-Transfer-Encoding. Other headers MUST NOT be included and MUST be ignored.

5. Operability considerations

5.1. Non-ASCII field names and values

Normally, MIME headers in multipart bodies are required to consist only of 7-bit data in the US-ASCII character set. While [RFC2388] suggested that non-ASCII field names be encoded according to the method in [RFC2047], this practice doesn't seem to have been followed widely.

This specification makes three sets of recommendations for three different states of workflow.

5.1.1. Avoid non-ASCII field names

For broadest interoperability with existing deployed software, those creating forms SHOULD avoid non-ASCII field names. This should not be a burden, because in general the field names are not visible to users. The field names in the underlying need not match what the user sees on the screen.

If non-ASCII field names are unavoidable, form or application creators SHOULD use UTF-8 uniformly. This will minimize interoperability problems.

5.1.2. Interpreting forms and creating form-data

Some applications of this specification will supply a character encoding to be used for interpretation of the multipart/form-data body. In particular, HTML 5 [W3C.REC-html5-20141028] uses:

- The content of a '_charset_' field, if there is one.
- the value of an accept-charset attribute of the <form> element, if there is one,
- the character encoding of the document containing the form, if it is US-ASCII compatible,

o otherwise UTF-8.

Call this value the form-charset. Any text, whether field name, field value, or (text/plain) form data which is uses characters outside the ASCII range MAY be represented directly encoded in the form-charset.

5.1.3. Parsing and interpreting form data

While this specification provides guidance for creation of multipart/form-data, parsers and interpreters should be aware of the variety of implementations. File systems differ as to whether and how they normalize Unicode names, for example. The matching of form elements to form-data parts may rely on a fuzzier match. In particular, some multipart/form-data generators might have followed the previous advice of [RFC2388] and used the [RFC2047] "encoded-word" method of encoding non-ASCII values:

```
encoded-word = "=?" charset "?" encoding "?" encoded-text "?="
```

Others have been known to follow [RFC2231], to send unencoded UTF-8, or even strings encoded in the form-charset.

For this reason, interpreting "multipart/form-data" (even from conforming generators) may require knowing the charset used in form encoding, in cases where the _charset_ field value or a charset parameter of a text/plain Content-Type header is not supplied.

5.2. Ordered fields and duplicated field names

Form processors given forms with a well-defined ordering SHOULD send back results in order (note that there are some forms which do not define a natural order.) Intermediaries MUST NOT reorder the results. Form parts with identical field names MUST NOT be coalesced.

5.3. Interoperability with web applications

Many web applications use the "application/x-url-encoded" method for returning data from forms. This format is quite compact, e.g.:

```
name=Xavier+Xantico&verdict=Yes&colour=Blue&happy=sad&Utf%F6r=Send
```

However, there is no opportunity to label the enclosed data with content type, apply a charset, or use other encoding mechanisms.

Many form-interpreting programs (primarily web browsers) now implement and generate multipart/form-data, but an existing

application might need to optionally support both the application/x-url-encoded format as well.

5.4. Correlating form data with the original form

This specification provides no specific mechanism by which multipart/form-data can be associated with the form that caused it to be transmitted. This separation is intentional; many different forms might be used for transmitting the same data. In practice, applications may supply a specific form processing resource (in HTML, the ACTION attribute in a FORM tag) for each different form. Alternatively, data about the form might be encoded in a "hidden field" (a field which is part of the form but which has a fixed value to be transmitted back to the form-data processor.)

6. IANA Considerations

Please update the Internet Media Type registration of multipart/form-data to point to this document, using the template in Section 8. In addition, please update the registrations of the "name" parameter and the "form-data" value in the "Content Disposition Values and Parameters" registry to both point to this document.

7. Security Considerations

All form processing software should treat user supplied form-data with sensitivity, as it often contains confidential or personally identifying information. There is widespread use of form "auto-fill" features in web browsers; these might be used to trick users to unknowingly send confidential information when completing otherwise innoccuous tasks. Multipart/form-data does not supply any features for checking integrity, ensuring confidentiality, avoiding user confusion, or other security features; those concerns must be addressed by the form-filling and form-data-interpreting applications.

Applications which receive forms and process them must be careful not to supply data back to the requesting form processing site that was not intended to be sent.

It is important when interpreting the filename of the Content-Disposition header to not overwrite files in the recipient's file space inadvertently.

User applications that request form information from users must be careful not to cause a user to send information to the requestor or a third party unwillingly or unwittingly. For example, a form might request 'spam' information to be sent to an unintended third party,

or private information to be sent to someone that the user might not actually intend. While this is primarily an issue for the representation and interpretation of forms themselves (rather than the data representation of the form data), the transportation of private information must be done in a way that does not expose it to unwanted prying.

With the introduction of form-data that can reasonably send back the content of files from a user's file space, the possibility arises that a user might be sent an automated script that fills out a form and then sends one of the user's local files to another address. Thus, additional caution is required when executing automated scripting where form-data might include a user's files.

Files sent via multipart/form-data may contain arbitrary executable content, and precautions against malicious content are necessary.

The considerations of [RFC2183] Sections 2.3 and 5 with respect to the filename parameter of the Content-Disposition header also apply to its usage here.

8. Media type registration for multipart/form-data

This section is the [RFC6838] media type registration.

Type name: multipart

Subtype name: form-data

Required parameters: boundary

Optional parameters: none

Encoding considerations: Common use is BINARY.
In limited use (or transports that restrict the encoding to 7BIT or 8BIT each part is encoded separately using Content-Transfer-Encoding Section 4.8.

Security considerations: See Section 7 of this document.

Interoperability considerations: This document makes several recommendations for interoperability with deployed implementations, including Section 4.8.

Published specification: This document.

Applications that use this media type: Numerous web browsers, servers, and web applications.

Fragment identifier considerations: None: Fragment identifiers are not defined for this type.

Additional information: None: no deprecated alias names, magic numbers, file extensions or Macintosh ssssfile type codes.

Person & email address to contact for further information Author of this document.

Intended Usage: COMMON

Restrictions on usage: none

Author: Author of this document.

Change controller: IETF

Provisional registration: N/A

9. References

9.1. Normative References

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

[RFC2047] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, November 1996.

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

[RFC2183] Troost, R., Dorner, S., and K. Moore, "Communicating Presentation Information in Internet Messages: The Content-Disposition Header Field", RFC 2183, August 1997.

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

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

9.2. Informative References

[I-D.ietf-appsawg-file-scheme]
Kerwin, M., "The file URI Scheme", draft-ietf-appsawg-file-scheme-00 (work in progress), January 2015.

[RFC1867] Nebel, E. and L. Masinter, "Form-based File Upload in HTML", RFC 1867, November 1995.

[RFC2388] Masinter, L., "Returning Values from Forms: multipart/form-data", RFC 2388, August 1998.

[RFC5987] Reschke, J., "Character Set and Language Encoding for Hypertext Transfer Protocol (HTTP) Header Field Parameters", RFC 5987, August 2010.

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

[W3C.REC-html32-19970114]
Raggett, D., "HTML 3.2 Reference Specification", World Wide Web Consortium Recommendation REC-html32-19970114, January 1997, <http://www.w3.org/TR/REC-html32-19970114>.

[W3C.REC-html5-20141028]
Hickson, I., Berjon, R., Faulkner, S., Leithead, T., Navara, E., O'Connor, E., and S. Pfeiffer, "HTML5", World Wide Web Consortium Recommendation REC-html5-20141028, October 2014, <http://www.w3.org/TR/2014/REC-html5-20141028>.

Appendix A. Changes from RFC 2388

The handling of non-ASCII field names changed-- no longer recommending the RFC 2047 method, instead suggesting senders send UTF-8 field names directly, and file names directly in the form-charset.

The handling of multiple files submitted as the result of a single form field (e.g. HTML's <input type=file multiple> element) results in each file having its own top level part with the same name parameter; the method of using a nested "multipart/mixed" from [RFC2388] is no longer recommended for creators, and not required for receivers as there are no known implementations of senders.

The _charset_ convention and use of an explicit form-data charset is documented.

'boundary' is a required parameter in Content-Type.

The relationship of the ordering of fields within a form and the ordering of returned values within multipart/form-data was not defined before, nor was the handling of the case where a form has multiple fields with the same name.

Editorial: Removed obsolete discussion of alternatives in appendix. Update references. Move outline of form processing into Introduction.

Appendix B. Alternatives

There are numerous alternative ways in which form data can be encoded; many are listed in [RFC2388] section 5.2. The multipart/form-data encoding is verbose, especially if there are many fields with short values. In most use cases, this overhead isn't significant.

More problematic are the differences introduced when implementors opted to not follow [RFC2388] when encoding non-ASCII field names (perhaps because "may" should have been "MUST"). As a result, parsers need to be more complex for matching against the possible outputs of various encoding methods.

Author's Address

Larry Masinter
Adobe

Email: masinter@adobe.com
URI: http://larry.masinter.net

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 17, 2015

J. Levine
Taughannock Networks
M. Delany
Apple Inc.

September 13, 2014

A "Null MX" No Service Resource Record for Domains that Accept No Mail
draft-ietf-appsawg-nullmx-10

Abstract

Internet mail determines the address of a receiving server through the DNS, first by looking for an MX record and then by looking for an A/AAAA record as a fallback. Unfortunately this means that the A/AAAA record is taken to be mail server address even when that address does not accept mail. The no service MX RR, informally called null MX, formalizes the existing mechanism by which a domain announces that it accepts no mail, without having to provide a mail server, which permits significant operational efficiencies.

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."

This Internet-Draft will expire on March 17, 2015.

Table of Contents

1. Conventions Used in This Document

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].

The terms RFC5321.MailFrom and RFC5322.From are used as defined in [RFC5598].

2. Introduction

This document defines the No Service MX, informally called null MX, as a simple mechanism by which a domain can indicate that it does not accept email.

SMTP clients have a prescribed sequence for identifying a server that accepts email for a domain. Section 5 of [RFC5321] covers this in detail, but in essence the SMTP client first looks up a DNS MX RR and if that is not found it falls back to looking up a DNS A or AAAA RR. Hence this overloads an email service semantic onto a DNS record with a different primary mission.

If a domain has no MX records, senders will attempt to deliver mail to the hosts at the domain's A or AAAA record's addresses. If there is no SMTP listener at the A/AAAA address, message delivery will be attempted repeatedly for a long period, typically a week, before the sending MTA gives up. This will delay notification to the sender in the case of misdirected mail, and will consume resources at the sender.

This document defines a null MX that will cause all mail delivery attempts to a domain to fail immediately, without requiring domains to create SMTP listeners dedicated to preventing delivery attempts.

3. MX Resource Records Specifying Null MX

To indicate that a domain does not accept email, it advertises a single MX RR (see [RFC1035], section 3.3.9) with an RDATA section consisting of preference number 0, and a zero length label, written in master files as ".", as the exchange domain, to denote that there exists no mail exchanger for a domain. Since "." is not a valid host name, a null MX record can not be confused with an ordinary MX record. The use of "." as a pseudo-host name meaning no service available is modeled on the SRV RR [RFC2782] where it has a similar meaning.

A domain that advertises a null MX MUST NOT advertise any other MX RR.

4. Effects of Null MX

The null MX record has a variety of efficiency and usability benefits.

4.1. SMTP Server Benefits

Mail often has an incorrect address due to user error, where the address was mistranscribed or misunderstood, for example, to alice@www.example.com or alice@example.org or alice@examp1e.com rather than alice@example.com. Null MX allows a mail system to report the delivery failure when the user sends the message, rather than hours or days later.

Senders of abusive mail often use forged undeliverable return addresses. Null MX allows Delivery Status Notifications (DSNs) and other attempted responses to such mail to be disposed of efficiently.

The ability to detect domains that do not accept email offers resource savings to an SMTP client. It will discover on the first sending attempt that an address is not deliverable, avoiding queuing and retries.

When a submission or SMTP relay server rejects an envelope recipient due to a domain's null MX record, it SHOULD use a 556 reply code[code521556] (Requested action not taken: domain does not accept mail) and a 5.1.TBD enhanced status code (Permanent failure: Recipient address has null MX).

A receiving SMTP server that chooses to reject email during the SMTP conversation that presents an undeliverable RFC5321.MailFrom or RFC5322.From domain can be more confident that for other messages a subsequent attempt to send a DSN or other response will reach a recipient SMTP server.

SMTP servers that reject mail because a RFC5321.MailFrom or RFC5322.From domain has a null MX record SHOULD use a 550 reply code (Requested action not taken: mailbox unavailable) and a 5.7.TBD enhanced status code (Permanent failure: Sender address has null MX).

4.2. Sending Mail from Domains that Publish Null MX

Null MX is primarily intended for domains that do not send or receive any mail, but have mail sent to them anyway due to mistakes or malice. Many receiving systems reject mail that has an invalid return address. Return addresses are needed to allow the sender to handle message delivery errors. An invalid return address often signals that the message is spam. Hence mail systems SHOULD NOT publish a null MX record for domains that they use in RFC5321.MailFrom or RFC5322.From addresses. If a server nonetheless does so, it risks having its mail rejected.

Operators of domains that do not send mail can publish SPF -all [RFC7208] policies to make an explicit declaration that the domains send no mail.

Null MX is not intended to be a replacement for the null reverse path described in RFC 5321 section 4.5.5 and does not change the meaning or use of a null reverse path.

5. Security Considerations

Within the DNS, a null MX RR is an ordinary MX record and presents no new security issues. If desired, it can be secured in the same manner as any other DNS record using DNSSEC.

6. IANA Considerations

IANA is requested to add the following entries to the "Enumerated Status Codes" sub-registry of the Simple Mail Transfer Protocol (SMTP) Enhanced Status Codes Registry.

```
Code:                X.1.TBD
Sample Text:         Recipient address has null MX
Associated basic status code:  556
Description:         This status code is returned when the associated
                     address is marked as invalid using a null MX.
Reference:           [this document]
Submitter:           [authors of this document]
Change controller: IESG

Code:                X.7.TBD
Sample Text:         Sender address has null MX
Associated basic status code:  550
Description:         This status code is returned when the associated
                     sender address has a null MX, and the SMTP
            receiver is configured to reject mail from such
            sender (e.g. because it could not return a DSN).
Reference:           [this document]
Submitter:           [authors of this document]
Change controller: IESG
```

7. Acknowledgements

We thank Dave Crocker for his diligent and lengthy shepherding of this document, and members of the appsawg working group for their constructive suggestions.

8. References

8.1. Normative References

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

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

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, October 2008.

[code521556] Klensin, J., "SMTP 521 and 556 Reply Codes", internet-draft draft-klensin-smtp-521code, .

8.2. Informative References

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

[RFC5598] Crocker, D., "Internet Mail Architecture", RFC 5598, July 2009.

[RFC7208] Kitterman, S., "Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1", RFC 7208, April 2014.

Appendix A. Change Log

NOTE TO RFC EDITOR: This section may be removed upon publication of this document as an RFC.

A.1. Change to appsawg-nullmx-10

Minor twiddle to clarify reference.

A.2. Change to appsawg-nullmx-09

Change 521 to 556, change reference.

A.3. Change to appsawg-nullmx-08

Fix name of IANA registry.

Yea, even yet more editorial cleanup.

A.4. Change to appsawg-nullmx-07

Add new enhanced status codes and ref for 521 return code.

Even yet more editorial cleanup.

A.5. Change to appsawg-nullmx-06

Even more editorial cleanup.

Mention SRV

you SHOULD NOT put a null MX on domains that send mail

A.6. Change to appsawg-nullmx-05

Fix ID nits, add NULL IANA section. More editorial cleanup.

A.7. Change to appsawg-nullmx-04

Reorganize.

A.8. Change to appsawg-nullmx-03

Editorial nits per Murray.

A.9. Change to appsawg-nullmx-02

Should not publish NULL MX with other MX.

Never say never.

Add 5.1.2 enhanced status code.

Minor editorial changes.

A.10. Change to appsawg-nullmx-1

Editorial improvements per D. Crocker's review.

A.11. Change to appsawg-nullmx-0

Fix typos.

Authors' Addresses

John Levine
Taughannock Networks
PO Box 727
Trumansburg, NY 14886

Phone: +1 831 480 2300
Email: standards@taugh.com
URI: http://jl.ly

Mark Delany
Apple Inc.
1 Infinite Loop
Cupertino, CA 95014

Email: mx0dot@yahoo.com

Applications Area Working Group
Internet-Draft
Intended Status: Informational
Expires: April 19, 2016

S. Leonard
Penango, Inc.

October 17, 2015

The text/markdown Media Type
draft-ietf-appsawg-text-markdown-12

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

Table of Contents

1. Introduction

1.1. This Is Markdown! Or: Markup and Its Discontents

In computer systems, textual data is stored and processed using a continuum of techniques. On the one end is plain text: computer-encoded text that consists only of a sequence of code points from a given standard, with no other formatting or structural information [UNICODE]. (On the other end is binary data, which computer systems store and process with bit-for-bit accuracy.) Many of these standards include control characters that are used as in-band signaling to cause effects other than the addition of a symbol (or grapheme) to the text.

Markup offers an alternative means to encode this signaling information by overloading certain graphic characters (see, e.g., [ISO646]) with additional meanings. Therefore, markup languages allow for annotating a document in a syntactically distinguishable way from the text, while keeping the annotations printable. Markup languages are (reasonably) well-specified and tend to follow (mostly) standardized syntax rules. Examples of formal markup languages include SGML, HTML, XML, and LaTeX. Standardized rules lead to interoperability between markup processors, but impose skill requirements on new users that lead to markup languages becoming less accessible to beginners. These rules also reify "validity": content that does not conform to the rules is treated differently (i.e., is rejected) than content that conforms.

In contrast to formal markup languages, lightweight markup languages use simple syntaxes; they are designed to be easy for humans to enter and understand with basic text editors. Markdown, the subject of this document, began as an /informal/ plain text formatting syntax [MDSYNTAX] and Perl script HTML/XHTML processor [MARKDOWN] targeted at non-technical users using unspecialized tools, such as plain text e-mail clients. [MDSYNTAX] explicitly rejects the notion of validity: there is no such thing as "invalid" Markdown. If the Markdown content does not result in the "right" output (defined as output that the author wants, not output that adheres to some dictated system of rules), the expectation is that the author should continue experimenting by changing the content or the processor to achieve the desired output.

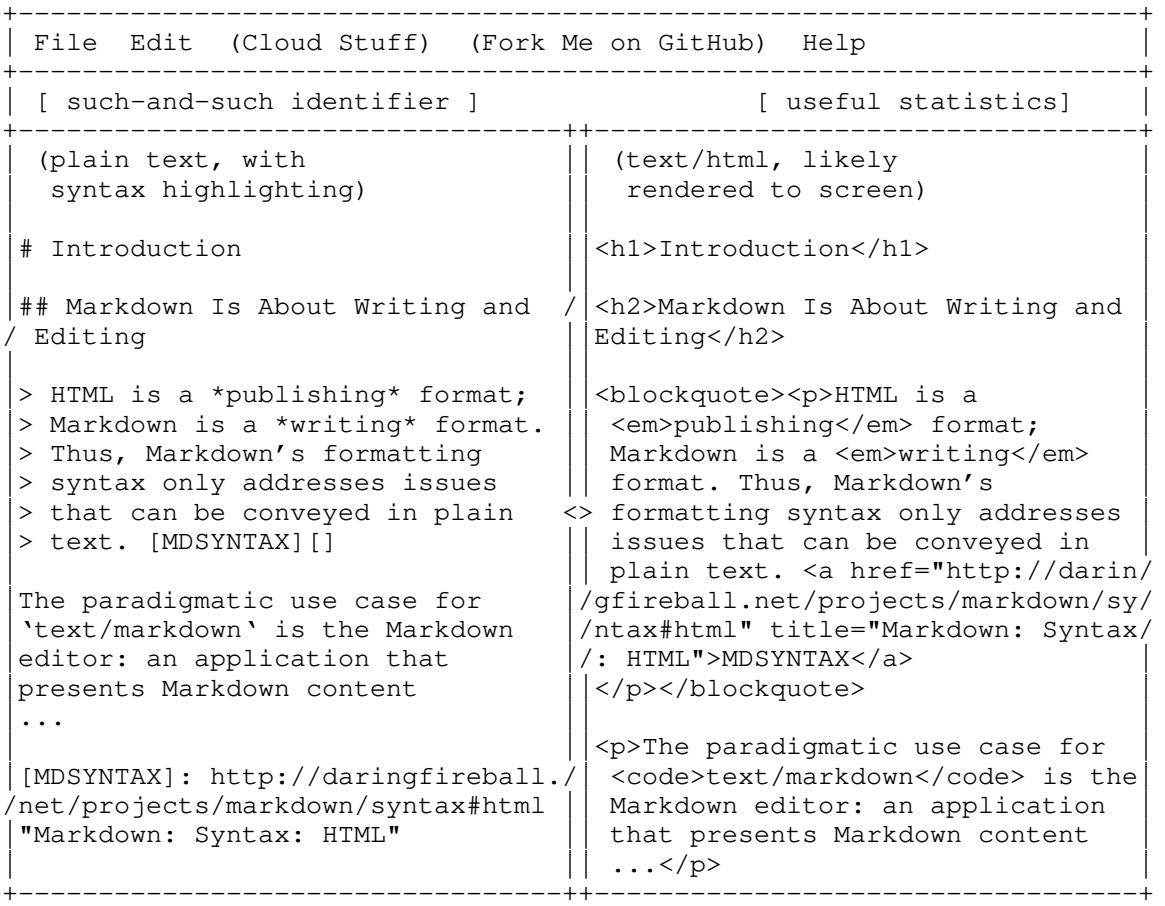
Since its development in 2004 [MARKDOWN], a number of web- and Internet-facing applications have incorporated Markdown into their text entry systems, frequently with custom extensions. Markdown has thus evolved into a kind of Internet meme [INETMEME] as different communities encounter it and adapt the syntax for their specific use cases. Markdown now represents a family of related plain text formatting syntaxes and implementations that, while broadly compatible with humans [HUMANE], are intended to produce different kinds of outputs that push the boundaries of mutual intelligibility between software systems.

To support identifying and conveying Markdown, this document defines a media type and parameters that indicate the Markdown author's intent on how to interpret the content. This registration draws particular inspiration from text/troff [RFC4263], which is a plain text formatting syntax for typesetting based on tools from the 1960s ("RUNOFF") and 1970s ("nroff", et. al.). In that sense, Markdown is a kind of troff for modern computing. A companion document [MDMTGUID] provides additional Markdown background, philosophy, local storage strategies, and variant registrations (including examples).

1.2. Markdown Is About Writing and Editing

> "HTML is a *publishing* format; Markdown is a *writing* format. Thus, Markdown's formatting syntax only addresses issues that can be conveyed in plain text." [MDSYNTAX]

The paradigmatic use case for text/markdown is the Markdown editor: an application that presents Markdown content (which looks like an e-mail or other piece of plain text writing) alongside a published format, so that an author can see results instantaneously and can tweak his or her input in real-time. A significant number of Markdown editors have adopted "split-screen view" (or "live preview") technology that looks like Figure 1:



LEGEND: "/" embedded in a vertical line represents a line-continuation marker, since a line break is not supposed to occur in that content.

Figure 1: Markdown Split-Screen/Live Preview Editor

To get the best results, implementations ought to produce and consume mutually intelligible and identifiable bits of Markdown. That way, users on diverse platforms can collaborate with their tools of choice. Those tools can be desktop-based (MarkdownPad, MultiMarkdown Composer), browser-based (Dillinger, Markable), integrated widgets (Discourse, GitHub), general-purpose editors (emacs, vi), or plain old "Notepad". Additionally, implementations ought to have common ways to identify particular areas of Markdown content when the Markdown becomes appreciably large (e.g., book chapters and Internet-Drafts--not just blog posts). So that users have the option to use Markdown in MIME-capable systems to convey their works in progress, not just their

finished products (for which full-blown markups ranging from text/html to application/pdf are appropriate), implementations ought to label such Markdown content with a common media type: text/markdown. This registration facilitates interoperability between these Markdown editors by conveying the syntax of the particular Markdown variant and the desired output format.

1.3. Definitions

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].

Since Markdown signifies a family of related formats with varying degrees of formal documentation and implementation, this specification uses the term "variant" to identify such formats.

2. 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 because neither [MDSYNTAX] nor many popular implementations at the time of this registration do either. [MDSYNTAX] clearly describes Markdown as a "writing format"; its syntax rules operate on characters (specifically, on punctuation) rather than code points. Many Markdown processors will get along just fine by operating on characters in the US-ASCII repertoire (specifically punctuation), blissfully oblivious to other characters or codes.

Optional parameters:

variant: An optional identifier of the specific Markdown variant that the author intended. The value serves as a "hint" to the recipient, meaning that the recipient MAY interpret the Markdown as that variant, but is under no obligation to do so. When omitted, there is no hint; the interpretation is entirely up to the receiver and context. This identifier is plain US-ASCII and case-insensitive. To promote interoperability, identifiers can be registered in the registry defined in Section 6. If a receiver

does not recognize the variant identifier, the receiver MAY present the identifier to a user to inform him or her of it.

Other parameters MAY be included with the media type. The variant SHOULD define the semantics of such other parameters. Additionally, the variant MAY be registered under another media type; this text/markdown registration does not preclude other registrations.

Encoding considerations:

Markdown content is plain text content; any octet sequence is valid as long as it conforms to the character codes of the charset parameter. See [RFC2046]. Markup characters in [MDSYNTAX] are limited to printable US-ASCII; however, other variants can define markup characters outside this range (including control characters such as NUL and characters encoded in multiple octets).

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, carries 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 processed. 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.

Interoperability considerations:

Markdown variations (some might say "innovations") are designed to be broadly compatible with humans ("humane"), but not necessarily with each other. Therefore, syntax in one Markdown derivative may be ignored or treated differently in another derivative. The overall effect is a general degradation of the output that increases with the quantity of variant-specific Markdown used in

the text. When it is desirable to reflect the author's intent in the output, stick with the variant identified in the variant parameter, i.e., receivers SHOULD only accept Markdown variants that they explicitly know about, and senders SHOULD avoid use of variants that intended recipients are not known to understand.

Published specification: This specification; [MDSYNTAX].

Applications that use this media type:

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

Fragment identifier considerations:

See Section 3.

Additional information:

Magic number(s): None
File extension(s): .md, .markdown
Macintosh file type code(s):
TEXT. A uniform type identifier (UTI) of "net.daringfireball.markdown", which conforms to "public.plain-text", is RECOMMENDED [MDUTI]. See [MDMTGUID] for other considerations.

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? No

Implementations SHOULD record the value of the variant parameter (and other parameters if defined by the variant) along with the Markdown content when the content leaves the domain of Internet media type-capable formats. Strategies for doing so are discussed in [MDMTGUID].

The Content-Disposition header (particularly the preview-type parameter) can be used with Markdown content. See Section 4.

3. Fragment Identifiers

[MARKDOWN] does not define any fragment identifiers, but some variants do, and many types of Markdown processor output (e.g., HTML or PDF) will have well-defined fragment identifiers. Which fragment identifiers are available for a given document are variant-defined.

When encoded in a URI, characters that are outside of the fragment production of [RFC3986] are percent-encoded. The default encoding (character set) of percent-encoded octets in URIs is the same as the Markdown content, which is identified by the charset parameter or by other contextual means. Fragment identifiers SHOULD be considered case-sensitive, which maintains consistency with HTML. Variants MAY override the guidance in this paragraph.

At least the first equals sign "=" SHOULD be percent-encoded to prevent ambiguity as described in the following section.

3.1. Parameters

Similar to application/pdf [RFC3778] and text/plain [RFC5147], this registration permits a parameter syntax for fragment identifiers. The syntax is a parameter name, the equals sign "=" (which MUST NOT be percent-encoded), and a parameter value. To the extent that multiple parameters can appear in a fragment production, the parameters SHALL be separated by the ampersand "&" (which MUST NOT be percent-encoded).

The only parameter defined in this registration is "line", which has the same meaning as [RFC5147] (i.e., counting is zero-based). For example: "#line=10" identifies the eleventh line of Markdown input. Implementers should take heed that different environments and character sets may have a wide range of code sequences to divide lines.

Markdown variants are free to define additional parameters.

4. Content Disposition and preview-type

The Content-Disposition header [RFC2183] conveys presentational information about a MIME entity, using a type and set of parameters. The parameter "preview-type" is defined here for Markdown content.

When present, "preview-type" indicates the Internet media type (and parameters) of the preview output desired from the processor by the author. With reference to the "paradigmatic use case" (i.e., collaborative Markdown editing) in Section 1.3, the preview-type parameter primarily affects the "right-hand" side of a Markdown

editor. There is no default value: when absent, a Markdown user agent can render or display whatever it wants.

The value of this parameter 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, e.g., [RFC2231] and [RFC6532]).

Implementations SHOULD anticipate and support HTML (text/html) and XHTML (application/xhtml+xml) output, to the extent that a syntax targets those markup languages. These types 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, to PDF, to OPML, and even to entire e-books (e.g., [PANDOC]).

The reflexive media type "text/markdown" in this parameter value means that the author does not want to invoke Markdown processing at all: the receiver SHOULD present the Markdown source as-is.

The "preview-type" parameter can be used for other types of content, but the precise semantics are not defined here.

5. Example

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

```
MIME-Version: 1.0
Content-Type: text/markdown; charset=UTF-8; variant=Original
Content-Disposition: attachment; filename=readme.md;
 preview-type="application/xhtml+xml"

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

This is some sample Markdown. [Hooray!][foo]
(Remember that link identifiers 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/loc 'Will Not Work with Markdown.pl-1.0.1'
```

6. IANA Considerations

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

IANA is asked to register "preview-type" in the Content Disposition Parameters subregistry of the Content Disposition Values and Parameters registry, with the following description: "Internet media type (and parameters) of the preview output desired from a processor by the author of the MIME content".

6.1. Markdown Variants

IANA is also asked to establish a registry called "Markdown Variants". While the registry is being created in the context of the text/markdown media type, the registry is intended for broad community use, so protocols and systems that do not rely on Internet media types can still tag Markdown content with a common variant identifier. Each entry in this registry shall consist of basic information about the variant:

Identifier:	unique identifier for the variant
Name:	the commonly known name of the variant
Description:	a prose description of the variant, including how it differs from other variants such as Original
Additional Parameters*:	additional Content-Type parameters
Fragment Identifiers*:	additional fragment identifier syntaxes and semantics
References:	URIs or other references to documentation
Contact Information:	whom to contact (email, URI, phone, address, etc.)
Expiration Date^:	when this provisional registration expires

* (optional)

^ (if provisional)

While the variant parameter is "plain US-ASCII" (see registration template), the Identifier field (and by implication, all registered identifiers) SHALL conform to the ABNF [RFC5234]:

```
ALPHA [*VCHAR (ALPHA / DIGIT)]
```

For style and compatibility reasons, the Identifier field SHOULD conform to the ABNF:

```
ALPHA *( ["-" / "." / "_" / "~"] 1*(ALPHA / DIGIT) )
```

I.e., the identifier MUST start with a letter and MAY contain punctuation in the middle, but not at the end: the last character MUST be alphanumeric. The second production uses the same characters as the "unreserved" rule of [RFC3986], and is designed to be compatible with characters in other identification systems, e.g., filenames. Since the identifier MAY be displayed to a user--particularly in cases where the receiver does not recognize the identifier--the identifier SHOULD be rationally related to the vernacular name of the variant.

The Name, Description, Additional Parameters, Fragment Identifiers, References, and Contact Information fields SHALL be in a Unicode character set (e.g., UTF-8).

The registry includes the registration in Section 6.1.4 (Original Markdown). [MDMTGUID] includes additional registrations.

6.1.1. Reserved Identifiers

The registry has the following identifiers RESERVED, as they have engendered some controversy in the Markdown community. No one is allowed to register them (or any case variations of them). These identifiers are not and cannot be registered:

Standard
Common
Markdown

The registry includes the following text in the note field:

> The variant names Standard, Common, and Markdown are reserved and cannot be registered.

6.1.2. Standard of Review

Registrations are made on a First-Come, First-Served [RFC5226] basis by anyone with a need to interoperate. While documentation is

required, any level of documentation is sufficient; thus, neither Specification Required nor Expert Review are warranted. The checks prescribed by this section can be performed automatically.

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

As a special "escape valve", registrations can be updated with IETF Review [RFC5226]. All fields may be updated except the variant identifier, which is permanent: not even case may be changed.

6.1.3. Provisional Registration

Any registrant may make a provisional registration to reserve a variant identifier. Only the variant identifier and contact information fields are required; the rest are optional. Provisional registrations expire after three months, after which time the variant identifier may be reused. To make a registration permanent, a registrant simply needs to complete a permanent registration with the same identifier as the provisional registration.

6.1.4. Original Markdown

The registry includes this initial variant. A conforming implementation that processes the variant parameter MUST recognize this variant (although the processing behavior is not defined here).

Identifier: Original

Name: Markdown

Description:
Gruber’s original Markdown syntax.

References:
[MARKDOWN]
[MDSYNTAX]

Contact Information:
(individual) John Gruber <http://daringfireball.net/>
<comments@daringfireball.net>

7. Security Considerations

See the Security considerations entry in Section 2.

8. References

8.1. 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>.

[MDUTI] Gruber, J., "Daring Fireball: Uniform Type Identifier for Markdown", August 2011, <http://daringfireball.net/linked/2011/08/05/markdown-uti>.

[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.

[RFC2183] Troost, R., Dorner, S., and K. Moore, Ed., "Communicating Presentation Information in Internet Messages: The Content-Disposition Header Field", RFC 2183, August 1997.

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

[RFC3778] Taft, E., Pravetz, J., Zilles, S., and L. Masinter, "The application/pdf Media Type", RFC 3778, May 2004.

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

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

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

[RFC5234] Crocker, D., Ed., and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC6532] Yang, A., Steele, S., and N. Freed, "Internationalized Email Headers", RFC 6532, February 2012.

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

8.2. Informative References

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

[UNICODE] The Unicode Consortium, "The Unicode Standard, Version 8.0.0", The Unicode Consortium, August 2015.

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

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

[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>.

[MDMTGUID] Leonard, S., "Guidance on Markdown: Design Philosophies, Stability Strategies, and Select Registrations", draft-ietf-appsawg-text-markdown-use-cases-07 (work in progress), September 2015.

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

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

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/

Network Working Group
Internet-Draft
Obsoletes: 4395 (if approved)
Intended status: Best Current Practice
Expires: October 10, 2015

D. Thaler, Ed.
Microsoft
T. Hansen
AT&T Laboratories
T. Hardie
Google
L. Masinter
Adobe

April 8, 2015

Guidelines and Registration Procedures for URI Schemes
draft-ietf-appsawg-uri-scheme-reg-06

Abstract

This document updates the guidelines and recommendations, as well as the IANA registration processes, for the definition of Uniform Resource Identifier (URI) schemes. It obsoletes RFC 4395.

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."

This Internet-Draft will expire on October 10, 2015.

Table of Contents

1. Introduction

The Uniform Resource Identifier (URI) protocol element and generic syntax is defined by [RFC3986]. Each URI begins with a scheme name, as defined by Section 3.1 of RFC 3986, that refers to a specification for identifiers within that scheme. The URI syntax provides a federated and extensible naming system, where each scheme's specification can further restrict the syntax and define the semantics of identifiers using that scheme.

This document obsoletes [RFC4395], which in turn obsoleted [RFC2717] and [RFC2718]. Recent documents have used the term "URI" for all resource identifiers, avoiding the term "URL" and reserving the term "URN" explicitly for those URIs using the "urn" scheme name ([RFC2141]). URN "namespaces" ([RFC3406]) are specific to the "urn" scheme and are not covered explicitly by this specification.

This document provides updated guidelines for the definition of new schemes, for consideration by those who are defining, registering, or evaluating those definitions, as well as a process and mechanism for registering schemes within the IANA URI Schemes registry. There is a single namespace for registered schemes. The intent of the registry is to:

- provide a central point of discovery for established URI scheme names, and easy location of defining documents for schemes;
- discourage multiple separate uses of the same scheme name;
- help those proposing new scheme names to discern established trends and conventions, and avoid names that might be confused with existing ones;
- encourage registration by setting a low barrier for registration.

1.1. URIs and IRIs

As originally defined, URIs only allowed a limited repertoire of characters chosen from US-ASCII. An Internationalized Resource Identifier (IRI), as defined by [RFC3987], extends the URI syntax to allow characters from a much greater repertoire, to accommodate resource identifiers from the world's languages. RFC 3987 [RFC3987] also defined a mapping between URIs and IRIs. IRIs use the same scheme names as URIs. Thus, there is no separate, independent registry or registration process for IRI schemes: the URI Schemes registry is used for both URIs and IRIs. Those who wish to describe resource identifiers that are useful as IRIs should define the corresponding URI syntax, and note that the IRI usage follows the rules and transformations defined in [RFC3987].

2. Terminology

Within this document, the key words MUST, MAY, SHOULD, REQUIRED, RECOMMENDED, and so forth are used within the general meanings established in [RFC2119], as requirements on future registrations.

This document distinguishes between a "scheme specification", being a document defining the syntax and semantics of a scheme, vs. a "scheme

registration request" being the completed template submitted to IANA. The term "scheme definition" refers generically to the syntax and semantics of a scheme, typically documented in a scheme specification.

3. Requirements for Permanent Scheme Definitions

This section gives considerations for new schemes. Meeting these guidelines is REQUIRED for 'permanent' scheme registration. 'Permanent' status is appropriate for, but not limited to, use in standards. For URI schemes defined or normatively referenced by IETF Standards-Track documents, 'permanent' registration status is REQUIRED.

[RFC3986] defines the overall syntax for URIs as:

```
URI = scheme ":" hier-part [ "?" query ] [ "#" fragment ]
```

A scheme definition cannot override the overall syntax for URIs. For example, this means that fragment identifiers cannot be re-used outside the generic syntax restrictions, and fragment identifiers are not scheme-specific. A scheme definition must specify the scheme name and the syntax of the scheme-specific part, which is clarified as follows:

```
URI = scheme ":" scheme-specific-part [ "#" fragment ]

scheme-specific-part = hier-part [ "?" query ]
```

3.1. Demonstrable, New, Long-Lived Utility

In general, the use and deployment of new schemes in the Internet infrastructure can be costly; some parts of URI processing are often scheme-dependent. Introducing a new scheme might require additional software, not only for client software and user agents but also in additional parts of the network infrastructure (gateways, proxies, caches) [W3CWebArch]. Since scheme names share a single, global namespace, it is desirable to avoid contention over use of short, mnemonic scheme names. New schemes ought to have utility to the Internet community beyond that available with already registered schemes. The scheme specification SHOULD discuss the utility of the scheme being registered.

3.2. Syntactic Compatibility

[RFC3986] defines the generic syntax for all URI schemes, along with the syntax of common URI components that are used by many URI schemes to define hierarchical identifiers. [RFC3987] extended this generic syntax to cover IRIs. All scheme specifications MUST define their own URI <scheme-specific-part> syntax. Care must be taken to ensure that all strings matching their scheme-specific syntax will also match the <absolute-URI> grammar described in [RFC3986].

New schemes SHOULD reuse the common URI components of [RFC3986] for the definition of hierarchical naming schemes. If there is a strong reason for a scheme not to use the hierarchical syntax, then the new scheme definition SHOULD follow the syntax of similar previously registered schemes.

Schemes that are not intended for use with relative URIs SHOULD avoid use of the forward slash "/" character, in order to avoid unintended processing such as resolution of "." and ".." (dot-segments).

Schemes SHOULD avoid improper use of "//". The use of double slashes in the first part of a URI is not a stylistic indicator that what follows is a URI: Double slashes are intended for use ONLY when the syntax of the <scheme-specific-part> contains a hierarchical structure. In URIs from such schemes, the use of double slashes indicates that what follows is the top hierarchical element for a naming authority. (Section 3.2 of RFC 3986 has more details.) Schemes that do not contain a conformant hierarchical structure in their <scheme-specific-part> SHOULD NOT use double slashes following the "<scheme>:" string.

New schemes SHOULD clearly define the role of reserved characters (see [RFC3986], Section 2.2) in URIs of the scheme being defined. The syntax of the new scheme should be clear about which of the "reserved" set of characters are used as delimiters within the URIs of the new scheme, and when those characters must be escaped, versus when they can be used without escaping.

3.3. Well-Defined

While URIs might or might not be defined as locators in practice, a scheme definition itself MUST be clear as to how it is expected to function. Schemes that are not intended to be used as locators SHOULD describe how the resource identified can be determined or accessed by software that obtains a URI of that scheme.

For schemes that function as locators, it is important that the mechanism of resource location be clearly defined. This might mean different things depending on the nature of the scheme.

In many cases, new schemes are defined as ways to translate between other namespaces or protocols and the general framework of URIs. For example, the "ftp" scheme translates into the FTP protocol, while the "mid" scheme translates into a Message-ID identifier of an email message. For such schemes, the description of the mapping SHOULD be complete, and in sufficient detail so that the mapping in both directions is clear: how to map from a URI into an identifier or set of protocol actions or name in the target namespace, and how legal values in the base namespace, or legal protocol interactions, are represented in a valid URI. See Section 3.6 for guidelines for encoding strings or sequences of bytes within valid character sequences in a URI. If not all legal values or protocol interactions of the base standard can be represented using the scheme, the definition SHOULD be clear about which subset is allowed, and why.

3.4. Definition of Operations

As part of the definition of how a URI identifies a resource, a scheme definition SHOULD define the applicable set of operations that can be performed on a resource using the URI as its identifier. A model for this is HTTP methods; an HTTP resource can be operated on by GET, POST, PUT, and a number of other methods available through the HTTP protocol. The scheme definition SHOULD describe all well-defined operations on the resource identifier, and what they are supposed to do.

Some schemes don't fit into the "information access" paradigm of URIs. For example, "telnet" provides location information for initiating a bi-directional data stream to a remote host; the only operation defined is to initiate the connection. In any case, the operations appropriate for a scheme SHOULD be documented.

Note: It is perfectly valid to say that "no operation apart from GET is defined for this URI". It is also valid to say that "there's only one operation defined for this URI, and it's not very GET-like". The important point is that what is defined on this scheme is described.

Scheme definitions SHOULD define a "default" operation for when a URI is invoked (or "dereferenced") by an application. For example, a common "default" operation today is to launch an application associated with the scheme name, and let it use the other URI components as inputs to do something. The default invocation, or dereferencing, of a URI SHOULD be "safe" in the sense described by

section 3.4 of [W3CWebArch]; i.e., performing such an invocation should not incur any additional obligations by doing so.

3.5. Context of Use

In general, URIs are used within a broad range of protocols and applications. For example, URIs are commonly used within hypertext documents as references to other resources. In some cases, a scheme is intended for use within a different, specific set of protocols or applications. If so, the scheme definition SHOULD describe the intended use and include references to documentation that define the applications and/or protocols cited. This does not obviate the need for documentation on applications and/or protocols to discuss URI schemes relevant to them.

3.6. Internationalization and Character Encoding

When describing schemes in which (some of) the elements of the URI are actually representations of human-readable text, care should be taken not to introduce unnecessary variety in the ways in which characters are encoded into octets and then into URI characters; see [RFC3987] and Section 2.5 (especially the last paragraph) of [RFC3986] for guidelines. If URIs of a scheme contain any text fields, the scheme definition MUST describe the ways in which characters are encoded and any compatibility issues with IRIs of the scheme.

The scheme specification SHOULD be as restrictive as possible regarding what characters are allowed in the URI, because some characters can create several different security issues (see, for example [RFC4690]).

Percent-encoded character sequences are automatically included by definition for characters given in IRI productions. This means that if you want to restrict the URI percent-encoded forms in some way, you must restrict the Unicode forms that would lead to them. In most cases, it is advisable to define the actual characters allowed in an IRI production, to allow the 'pct-encoded' definition from Section 2.1 of [RFC3986] at the same places, and to add prose that limits percent-escapes to those that can be created by converting valid UTF-8 character sequences to percent-encoding.

3.7. Clear Security and Privacy Considerations

Definitions of schemes MUST be accompanied by a clear analysis of the security and privacy implications for systems that use the scheme; this follows the practice of Security Consideration sections within IANA registrations [RFC5226].

In particular, Section 7 of RFC 3986 [RFC3986] describes general security considerations for URIs, while [RFC3987] gives those for IRIs. The definition of an individual scheme should note which of these apply to the specified scheme, in addition to any more scheme-specific concerns. For example, if the scheme-specific part is privacy sensitive then that should be documented.

3.8. Scheme Name Considerations

Section 3.1 of RFC 3986 defines the syntax of a URI scheme name; this syntax remains the same for IRIs. New scheme registrations MUST follow this syntax, which only allows a limited repertoire of characters (taken from US-ASCII). Although the syntax for the scheme name in URIs is case insensitive, the scheme name itself MUST be registered using lowercase letters.

Scheme names SHOULD be short, but also sufficiently descriptive and distinguished to avoid problems.

Schemes SHOULD NOT use names or other symbols that might cause problems with rights to use the name in IETF specifications and Internet protocols. For example, be careful with trademark and service mark names. (See Section 7.4 of [RFC3978].)

Schemes SHOULD NOT use names that are either very general purpose or associated in the community with some other application or protocol. Schemes also SHOULD NOT use names that are overly general or grandiose in scope (e.g., that allude to their "universal" or "standard" nature.)

A scheme name is not a "protocol." However, like a service name as defined in section 5 of [RFC6335], it often identifies a particular protocol or application. If a scheme name has a one-to-one correspondence with a service name, then the names SHOULD be the same.

Some organizations desire their own namespace for URI scheme names for private use (see Section 6). In doing so, it is important to prevent collisions, and to make it possible to identify the owner of a private use scheme. To accomplish these two goals, such organizations SHOULD use a prefix based on their domain name, expressed in reverse order. For example, a URI scheme name of com.example.mything might be used by the organization that owns the example.com domain name. Care must be taken, however, if the organization later loses the domain name embedded in their scheme names, since domain name registrations are not permanent. To associate the private use scheme name with the original organization,

the private use scheme can be registered using the registration procedure in Section 7.

Furthermore, to prevent collisions with private use scheme names, new scheme names registered MUST NOT contain a "." unless actually constructed from a reversed domain name.

3.9. Interoperability Considerations

If the person or group registering the scheme is aware of any details regarding the scheme that might impact interoperability, identify them. For example: proprietary or uncommon encoding methods; incompatibility with types or versions of any underlying protocol.

4. Guidelines for Provisional URI Scheme Registration

'Provisional' registration can be used for schemes that are not part of any standard, but that are intended for use (or observed to be in use) that is not limited to a private environment within a single organization. 'Provisional' registration can also be used as an intermediate step on the way to 'permanent' registration, e.g., before the scheme specification is finalized as a standard.

For a 'provisional' registration, the following apply:

- The scheme name must meet the syntactic requirements of Section 3.8.
- There must not already be an entry with the same scheme name. In the unfortunate case that there are multiple, different uses of the same scheme name, the Designated Expert can approve a request to modify an existing entry to note the separate use.
- Contact information identifying the person supplying the registration must be included. Previously unregistered schemes discovered in use can be registered by third parties (even if not on behalf of those who created the scheme). In this case, both the registering party and the scheme creator SHOULD be identified.
- If no permanent, citable specification for the scheme definition is included, credible reasons for not providing it SHOULD be given.
- The scheme definition SHOULD include clear security considerations (Section 3.7) or explain why a full security analysis is not available (e.g., in a third-party scheme registration).

- If the scheme definition does not meet the guidelines laid out in Section 3, the differences and reasons SHOULD be noted.

5. Guidelines for Historical URI Scheme Registration

In some circumstances, it is appropriate to note a scheme that was once in use or registered but for whatever reason is no longer in common use or whose use is not recommended. In this case, it is possible for an individual to request that the URI scheme be registered (newly, or as an update to an existing registration) as 'historical'. Any scheme that is no longer in common use MAY be designated as 'historical'; the registration SHOULD contain some indication as to where the scheme was previously defined or documented.

6. Guidelines for Private URI Scheme Use

Unregistered schemes can cause problems if use is not limited to a private environment within a single organization, since the use could leak out beyond the closed environment. Even within a closed environment, other colliding uses of the same scheme name could occur. As such, a unique namespace (see Section 3.8) MUST be used, and it is strongly encouraged to do a 'provisional' registration unless the scheme name is constructed from a domain name as discussed in Section 3.8.

7. URI Scheme Registration Procedure

7.1. General

The IANA policy (using terms defined in [RFC5226]) for 'provisional' registration was formerly Expert Review. The policy for 'permanent' and 'historical' registration continues to be Expert Review, but this document changes the policy for 'provisional' registration to First Come First Served.

The registration procedure is intended to be very lightweight for non-contentious registrations. For the most part, we expect the good sense of submitters and reviewers, guided by these procedures, to achieve an acceptable and useful consensus for the community.

In exceptional cases, where the negotiating parties cannot form a consensus, the final arbiter of any contested registration shall be the IESG.

If standardization is anticipated, the working group or individuals concerned are advised to submit an early 'permanent' registration request, rather than waiting until the standardization process has

run its course. IANA will pass this to the Designated Expert who may recommend 'provisional' registration until the specification is approved as a standard. This will provide opportunity for feedback while specification development and review is still active, and the submitter(s) are still in a position to respond to any issues that might be raised. If and when the specification is approved as a standard, the submitters should submit a request to change the registration status to 'permanent'.

The role of the Designated Expert in the procedure for 'permanent' registrations described here is to ensure that the normal open review process has been properly followed, and to raise possible concerns about wider implications of proposals for the use and deployment of URIs. Nothing in the procedure empowers the Designated Expert to override properly arrived-at IETF or working group consensus.

7.2. Registration Procedures

Someone wishing to register a new scheme MUST:

1. Check the IANA URI Schemes registry to see whether there is already an entry for the desired name. If there is already an entry under the name, choose a different scheme name, or update the existing scheme specification.

2. Prepare a scheme registration request using the template specified in Section 7.4. The scheme registration request can be contained in an Internet Draft, submitted alone, or as part of some other permanently available, stable, protocol specification. The scheme registration request can also be submitted in some other form (as part of another document or as a stand-alone document), but the scheme registration request will be treated as an "IETF Contribution" under the guidelines of [RFC3978].

3. If the registration request is for a 'permanent' registration (or, optionally, for any other registration if desired):

    1. Review the requirements in Section 3.

    2. Send a copy of the scheme registration request or a pointer to the containing document (with specific reference to the section with the scheme registration request) to the mailing list uri-review@ietf.org , requesting review. In addition, request review on other relevant mailing lists as appropriate. For example, general discussion of URI syntactical issues can be discussed on uri@w3.org; schemes for a network protocol can be discussed on a mailing list for

that protocol. Allow a reasonable time for discussion and comments. Four weeks is reasonable for a 'permanent' registration request.

3. Respond to review comments and make revisions to the proposed registration as needed to bring it into line with the guidelines given in this document.

4. Submit the (possibly updated) scheme registration request (or pointer to document containing it) to IANA at iana@iana.org.

Upon receipt of a scheme registration request, the following steps MUST be followed:

1. IANA checks the submission for completeness; if required sections of the scheme registration request are missing or any citations are not correct, IANA will reject the registration request. A registrant can resubmit a corrected request if desired.

2. If the request is for 'provisional' registration and no entry already exists in the current registry for the same name, IANA adds the registration to the registry, under the First Come First Served policy.

3. Otherwise, IANA enters the registration request in the IANA registry, with status marked as "Pending Review", and the remainder of this section applies.

4. IANA requests Expert Review of the registration request against the corresponding guidelines from this document.

5. The Designated Expert will evaluate the request against the criteria of the requested status.

6. In the case of a 'permanent' registration request, the Designated Expert may:

   * Accept the specification of the scheme for 'permanent' registration.

   * Suggest 'provisional' registration instead.

   * Request IETF review and IESG approval; in the meanwhile, suggest 'provisional' registration.

   * Request additional review or discussion, as necessary.

7. If an entry already exists for the same name, the Designated Expert will determine whether the request should be rejected, or whether the existing entry should be modified to note the separate use. This conflict process applies regardless of the requested status or the status of the existing entry.

8. Once Expert Review approves registration for a given status, IANA updates the registration to indicate the approved status. If Expert Review instead rejects the registration, the "Pending Review" request is removed from the registry.

Either based on an explicit request or independently initiated, the Designated Expert or the IESG can request the upgrade of a 'provisional' registration to a 'permanent' one. In such cases, IANA will update the status of the corresponding entry. Typically this would only occur if the use is considered a standard (not necessarily an IETF standard).

7.3. Change Control

Registrations can be updated in the registry by the same mechanism as required for an initial registration. In cases where the original definition of the scheme is contained in an IESG-approved document, update of the specification also requires IESG approval.

'Provisional' registrations can be updated by the original registrant or anyone designated by the original registrant. In addition, the IESG can reassign responsibility for a 'provisional' registration scheme, or can request specific changes to a scheme registration. This will enable changes to be made to schemes where the original registrant is out of contact, or unwilling or unable to make changes.

Transition from 'provisional' to 'permanent' status can be requested and approved in the same manner as a new 'permanent' registration. Transition from 'permanent' to 'historical' status requires IESG approval. Transition from 'provisional' to 'historical' can be requested by anyone authorized to update the 'provisional' registration.

7.4. URI Scheme Registration Template

This template describes the fields that MUST be supplied in a scheme registration request, suitable for adding to the registry:

Scheme name:
  See Section 3.8 for guidelines.

Status:

This reflects the status requested, and must be one of 'permanent', 'provisional', or 'historical'.

Applications/protocols that use this scheme name:
See Section 3.5.

Contact:
Person (including contact information) to contact for further information.

Change controller:
Organization or person (often the author), including contact information, authorized to change this.

References:
Include full citations for all referenced documents. Scheme registration requests for 'provisional' registration can be included in an Internet Draft; when the documents expire or are approved for publication as an RFC, the registration will be updated. A scheme specification is only required for 'permanent' registration.

The previous version of this specification required the following additional fields in a scheme registration request. These fields are no longer part of the template. The answers instead belong in the scheme specification.

Scheme syntax:
See Section 3.2 for guidelines.

Scheme semantics:
See Section 3.3 and Section 3.4 for guidelines.

Encoding considerations:
See Section 3.3 and Section 3.6 for guidelines.

Interoperability considerations:
See Section 3.9 for guidelines.

Security considerations:
See Section 3.7 for guidelines.

8. The "example" URI Scheme

There is a need for a scheme name that can be used for examples in documentation without fear of conflicts with current or future actual schemes. The scheme "example" is hereby registered as a 'permanent' scheme for that purpose.

The "example" scheme is specified as follows:

Scheme syntax: The entire range of allowable syntax specified in [RFC3986] is allowed for "example" URIs. Similarly, the entire range of allowable syntax specified in [RFC3987] is allowed for "example" IRIs. For example, <example:foo>, <example:/foo>, and <example://foo> are all valid.

Scheme semantics: URIs in the "example" scheme are to be used for documentation purposes only. The use of "example" URIs must not be used as locators, identify any resources, or specify any particular set of operations.

Encoding considerations: See Section 2.5 of [RFC3986] for guidelines.

Interoperability considerations: None.

Security considerations: None.

8.1. "Example" URI Scheme Registration Request

Scheme name: example

Status: permanent

Applications/protocols that use this scheme name: An "example" URI is to be used for documentation purposes only. It MUST NOT be used for any protocol.

Contact: N/A

Change controller: IETF

References: Section 8 of this RFC XXXX.
RFC Editor Note: Replace XXXX with this RFC's reference.

9. IANA Considerations

Previously, the former "URL Scheme" registry was replaced by the "Uniform Resource Identifier (URI) Schemes" registry. The process was based on [RFC5226] "Expert Review" with an initial (optional) mailing list review.

The updated template has an additional field for the status of the scheme, and the procedures for entering new name schemes have been augmented. Section 7 establishes the process for new scheme registration.

IANA is requested to do the following:

- Update the URI Schemes registry to point to this document.
- Combine the "Permanent URI Schemes", "Provisional URI Schemes", and "Historical URI Schemes" sub-registries into a single common registry with an additional "Status" column containing the status (Permanent, Provisional, Historical, or Pending Review), and an additional "Notes" column which is normally empty, but may contain notes approved by the Designated Expert.
- Add the "example" URI scheme to the registry (see the template in Section 8.1 for registration).

10. Security Considerations

All registered values are expected to contain clear security considerations as discussed in Section 3.7. However, information concerning possible security vulnerabilities of a protocol might change over time. Consequently, claims as to the security properties of a registered scheme might change as well. As new vulnerabilities are discovered, information about such vulnerabilities might need to be attached to existing documentation, so that users are not misled as to the true security properties of a registered scheme.

11. Acknowledgements

Thanks to Mark Nottingham and Graham Klyne and other members of the apps-discuss@ietf.org mailing list for their comments on this document.

Many thanks to Patrik Faltstrom, Paul Hoffmann, Ira McDonald, Roy Fielding, Stu Weibel, Tony Hammond, Charles Lindsey, Mark Baker, and other members of the uri@w3.org mailing list for their comments on earlier versions.

Parts of this document are based on [RFC2717], [RFC2718] and [RFC3864]. Some of the ideas about use of URIs were taken from the "Architecture of the World Wide Web" [W3CWebArch].

12. References

12.1. Normative References

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

[RFC2141] Moats, R., "URN Syntax", RFC 2141, May 1997.

[RFC3978] Bradner, S., "IETF Rights in Contributions", RFC 3978, March 2005.

[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", BCP 26, RFC 5226, May 2008.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, August 2011.

12.2. Informative References

[RFC2717] Petke, R. and I. King, "Registration Procedures for URL Scheme Names", BCP 35, RFC 2717, November 1999.

[RFC2718] Masinter, L., Alvestrand, H., Zigmond, D., and R. Petke, "Guidelines for new URL Schemes", RFC 2718, November 1999.

[RFC3406] Daigle, L., van Gulik, D., Iannella, R., and P. Faltstrom, "Uniform Resource Names (URN) Namespace Definition Mechanisms", BCP 66, RFC 3406, October 2002.

[RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, September 2004.

[RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, January 2005.

[RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 35, RFC 4395, February 2006.

[RFC4690] Klensin, J., Faltstrom, P., Karp, C., and IAB, "Review and Recommendations for Internationalized Domain Names (IDNs)", RFC 4690, September 2006.

[W3CWebArch] W3C Technical Architecture Group, "Architecture of the World Wide Web, Volume One", December 2004, <http://www.w3.org/TR/webarch/>.

Appendix A. Changes Since RFC 4395

1. Combined the Historical, Permanent, and Provisional URI Schemes registries into one registry with a status column. This is done to make it easier to prevent duplicates and see existing conventions.

2. Added a Notes column in the registry for notes approved by the Designated Expert.

3. Moved the following fields out of the scheme registration request template and into the requirements for a scheme specification: Scheme syntax, Scheme semantics, Encoding considerations, Interoperability considerations, and Security considerations.

4. Simplified the process for 'provisional' registration significantly: changed from Expert Review to First Come First Served, and clarified that mailing list review is not required.

5. Updated process for handling of scheme name conflicts, so that adding a note can be approved by the Designated Expert rather than the IESG.

6. Clarified that a "URI scheme name" and an "IRI scheme name" are the same thing and thus use the same IANA registry.

7. Clarified that a registration request falls under the "IETF Contribution" rules, but the scheme's specification need not.

8. Added the "example:" URI scheme.

9. Added text about when to use 'provisional' registration.

10. Updated convention for Private use schemes to use "." (instead of "-") between domain name labels, to reduce chance of collision, and recommended use of a reverse domain name prefix to allow identifying the owning organization.

11. Recommended that scheme definitions define a "default" operation for when a URI is invoked.

12. Recommended that a scheme name be the same as the service name, when there exists a 1:1 correspondence.

13. Elaborated on when a 'provisional' request should be upgraded to Permanent.

14. Clarified that since, per RFC 3986, fragment identifiers are not scheme-specific and hence are not part of the scheme definition.

15. Added text about getting early review and registration for schemes intended to become standards.

16. Added text about privacy as an example of what should be covered in security considerations in a scheme definition.

17. Clarified the role of a Designated Expert in reviewing Standards-Track registrations.

Authors' Addresses

Dave Thaler (editor)
Microsoft
One Microsoft Way
Redmond, WA 98052
US

Phone: +1 425 703 8835
Email: dthaler@microsoft.com

Tony Hansen
AT&T Laboratories
200 Laurel Ave.
Middletown, NJ 07748
USA

Email: tony+urireg@maillennium.att.com

Ted Hardie
Google

Phone: +1 408 628 5864
Email: ted.ietf@gmail.com

Larry Masinter
Adobe
345 Park Ave.
San Jose, CA 95110
US

Phone: +1 408 536 3024
Email: masinter@adobe.com
URI: http://larry.masinter.net

Network Working Group
Internet-Draft
Updates: 7001 (if approved)
Intended status: Standards Track
Expires: June 22, 2015

F. Martin, Ed.
LinkedIn

December 19, 2014

Authentication-Results Registration for TLS
draft-martin-authentication-results-tls-03

Abstract

This memo updates the registry of properties in Authentication-Results: message header fields to allow relaying of the results of an email sent using STARTTLS [RFC3207] or not.

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."

This Internet-Draft will expire on June 22, 2015.

Copyright Notice

Table of Contents

1. Introduction

STARTTLS [RFC3207] defines how to send an email over an SMTP [RFC5321] encrypted session between two mail servers.

This memo thus registers an additional reporting property allowing a TLS result to be relayed as an annotation in a message header.

1.1. Requirements Language

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 RFC 2119 [RFC2119].

1.2. Discussion

STARTTLS [RFC3207] defines how to send an email over an encrypted session between two mail servers, Message Transfer Agent (MTA), using the TLS [RFC5246] protocol.

Most of these exchanges are opportunistic, meaning a best effort is done to establish an encrypted message exchange regardless of the strength of the cipher or the validity of the certificates used. However, the results of this negotiation should be recorded in the message via the Authentication-Results header [RFC7001] to indicate to other message processing algorithms, including Messaging User Agents (MUA), how securely this message was transmitted from the MTA client to the MTA server.

The concept of authentication here is related to the presentation of a certificate which is verified valid by a set of trusted Certificate

Authorithies (CA), via DANE [RFC6698] or by local policy. This does not indicate that any string in the certificate is related to any string in the email.

The usage and usefulness of the Authentication-Results header is discussed in [RFC7001].

2. Definitions

This memo adds to the "Email Authentication Methods" registry, created by IANA upon publication of [RFC7001], the following:

- The method "tls"; and
- Associated with that method, the properties (reporting items) "cert.client", "cert.server", "cert.verif", "tls.v", "key.ciphersuite", "key.fingerprint", "key.length" and "key.strength".

2.1. results meaning

The "tls" method can have the following results:

none: the message was sent in clear.

pass: the message was sent encrypted and the client certificate was verified valid either using a trusted CA, via DANE [RFC6698] or via a local policy and host identity was verified.

selfsigned: the message was sent encrypted but the client certificate is self signed.

invalidhost: the message was sent encrypted and the client certificate is verified valid but the host identity is invalid.

fail: the message was sent encrypted but the client certificate is not valid. It is advised to use comments to indicate the nature of the problem like certifcate expired, not linked to a trusted CA,...

temperror: the message was sent encrypted and the server was not able to verify the client certificate at this time. This may indicate for instance that the server could not fetch the CRL.

permerror: the message was sent encrypted and the client certificate was not verified by the MTA server. MTA should always attempt to verify the client certificate.

2.2. properties

cert.client: the subject of the X.509 certificate used by the client to initiate TLS.

cert.server: the subject of the X.509 certificate used by the server to initiate TLS (optional).

cert.clientalt: the subject alternative name of the X.509 certificate used by the client to initiate TLS (optional).

cert.serveralt: the subject alternative name of the X.509 certificate used by the server to initiate TLS (optional).

cert.clientissuer: the issuer of the X.509 certificate used by the client to initiate TLS (optional).

cert.serverissuer: the issuer of the X.509 certificate used by the server to initiate TLS (optional).

cert.verif: the type of certification performed: CA, DANE [RFC6698], LOCAL (optional).

tls.v: the protocol version used to encrypt SSL2.0, SSL3.0, TLS1.0, TLS1.1,... (optional)

key.ciphersuite: the description of the TLS cipher suite used as defined in the IANA cipher suite registry.

key.fingerprint: the fingerprint of the key used (optional).

key.length: the length in bits of the key used (optional).

key.strength: as many SMTP TLS are opportunistic in nature this property is an arbitrary value set by the MTA server to indicate the strength of the encryption (optional).

While ciphers strength vary overtime, and key length in bits does not indicate a comparable strength between various cyphers, it may be difficult for all the processors of the authentication-results header to redo the analysis based on the cipher used and all to arrive to the same conclusion. It seems, therefore, best if the receiving MTA does that analysis and communicate it to the other layers. This is the purpose of the key.strength. For instance This value could be used by the MUA to indicate to the end user some quality of the encryption channel.

3. IANA Considerations

Per [IANA], the following items have been added to the "Email Authentication Methods" registry:

Method	Defined	ptype	property	value
tls	RFC 3207	cert	client	subject of client certificate section 4.1.2.6 of RFC 5280
tls	RFC 3207	cert	server	subject of server certificate section 4.1.2.6 of RFC 5280
tls	RFC 3207	cert	clientalt	alternate subject of client certificate section 4.2.1.6 of RFC 5280
tls	RFC 3207	cert	serveralt	alternate subject of server certificate section 4.2.1.6 of RFC 5280
tls	RFC 3207	cert	clientissuer	issuer of client certificate section 4.1.2.4 of RFC 5280
tls	RFC 3207	cert	serverissuer	issuer of server certificate section 4.1.2.4 of RFC 5280
tls	RFC 3207	cert	verif	CA DANE LOCAL

tls	RFC 3207	tls	v	protocol version description from RFC 5246
tls	RFC 3207	key	ciphersuite	IANA cipher suite registry description from RFC 5246
tls	RFC 3207	key	fingerprint	key fingerprint from RFC 5246
tls	RFC 3207	key	length	in bits
tls	RFC 3207	key	strength	low medium high

Also, the following items have been added to the "Email Authentication Result Names" registry:

Code	Existing/New	Defined In	Method	Meaning
none	existing	RFC 7001	tls (added)	this memo
pass	existing	RFC 7001	tls (added)	this memo
selfsigned	existing	RFC 7001	tls (added)	this memo
invalidhost	existing	RFC 7001	tls (added)	this memo
fail	existing	RFC 7001	tls (added)	this memo
temperror	existing	RFC 7001	tls (added)	this memo
permerror	existing	RFC 7001	tls (added)	this memo

4. Security Considerations

This memo creates a mechanism for relaying STARTTLS [RFC3207] results using the structure already defined by [RFC7001]. The Security Considerations sections of those documents should be consulted.

By this mechanism, some identifiers of the client certificates gets to live pass the receiving MTA. This is a change in the sender expectation on where the client certificate is used

5. References

5.1. Normative References

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

[RFC3207] Hoffman, P., "SMTP Service Extension for Secure SMTP over Transport Layer Security", RFC 3207, February 2002.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, October 2008.

[RFC6698] Hoffman, P. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", RFC 6698, August 2012.

[RFC7001] Kucherawy, M., "Message Header Field for Indicating Message Authentication Status", RFC 7001, September 2013.

5.2. Informative References

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

Appendix A. Authentication-Results Examples

This section presents an example of the use of this new header field to indicate TLS results.

A.1. TLS Results

A message that went over a successful TLS session:

```
Authentication-Results: mail-router.example.net;
  dkim=pass (good signature) header.d=newyork.example.com
    header.b=oINEO8hg;
  tls=pass (verified, expires 20140505)
    cert.verif=CA
    cert.client="CN=smtp.example.com,O=ACME,L=ToonTown,
      ST=CA,C=US"
    cert.clientalt="DNS:smtp.example.com, DNS:newyork.example.com"
    cert.clientissuer="C=US, O=AcmeCert Inc, CN=AcmeCert CA"
    key.ciphersuite=TLS_RSA_WITH_RC4_128_SHA
    tls.v=TLS1.0
    key.fingerprint="68:B3:29:DA:98:93:E3:40:99:C7:D8:
      AD:5C:B9:C9:40"
    key.length=128
    key.strength=MEDIUM;
Received: from newyork.example.com
  (newyork.example.com [192.0.2.250])
  by mail-router.example.net (8.11.6/8.11.6)
  for <recipient@example.net>
  with ESMTPS id i7PK0sH7021929;
  Fri, Feb 15 2002 17:19:22 -0800
DKIM-Signature: v=1; a=rsa-sha256; s=rashani;
  d=newyork.example.com;
  t=1188964191; c=relaxed/simple;
  h=From:Date:To:VBR-Info:Message-Id:Subject;
  bh=sEu28nfs9fuZGD/pSr7ANysbY3jtdaQ3Xv9xPQtS0m7=;
   b=oINEO8hgn/gnunsg ... 9n9ODSNFSDij3=
From: sender@newyork.example.com
Date: Fri, Feb 15 2002 16:54:30 -0800
To: meetings@example.net
Message-Id: <12345.abc@newyork.example.com>
Subject: here's a sample
```

Author's Address

Franck Martin (editor)
LinkedIn
Mountain View, CA
US

Email: fmartin@linkedin.com

Network Working Group
Internet-Draft
Intended Status: Informational
Expires: April 26, 2015

S. Leonard
Penango, Inc.

October 23, 2014

The Windows Bitmap Media Type
draft-seantek-image-bmp-01

Abstract

This document registers the image/bmp media type for use with the Windows Bitmap format (BMP), also known as Device-Independent Bitmap (DIB). Originally designed for Microsoft Windows 2.0 and OS/2, these bitmaps contain a single raster graphic in a variety of compressed or uncompressed formats.

1. Introduction

Long before the invention of Portable Network Graphics (PNG), Microsoft Corporation and IBM Corporation needed to record images in a format that their applications and operating systems could easily render on low-end machines (Intel 80286). The resulting "BMP" format contains a single raster graphic with basic header fields that can be easily mapped (or "blitted") to locations in memory. As computing moved from 16-bit to 32-bit, BMP evolved to contain 32-bit structures. As the 90s wore on, the venerable BMP got boosts with support for additional color spaces, color profiles, and compression formats. The same basic format can be used to convey 2-bit black-and-white bitmaps with a 1-bit alpha mask from the ’80s, and full-color Ultra HD images on leading-edge displays. BMP is a building block of other formats, including Windows Metafiles, Windows Icons, and Windows Cursors.

Many implementations of BMP were created because of Windows’ commercial success in the 1990s. Usage of the format for interchange has [[probably?]] declined since the advent of PNG (for lossless raster graphics) and JPEG (for lossy raster graphics); however, a large body of free and commercially available BMP artwork exists. Since Windows Icons are a building block of "favicon.ico" Web technology, an implementer would almost certainly need to support this format for basic interoperability.

Microsoft publicly documented the BMP format as early as the 1992 Windows 3.1 SDK (in the Windows Metafile documentation). Since 2007 Microsoft has released the format specification [MS-WMF], which includes most components of the Windows Bitmap format, under its Open Specification Promise [MS-OSP]. See Section 2.2.2.9 of [MS-WMF] (DeviceIndependentBitmap Object). BMP data begins with a BITMAPFILEHEADER and is followed by one of the bitmap headers (BITMAPINFOHEADER, BITMAPV4HEADER, or BITMAPV5HEADER), optional color table data, bitmap data, and optional profile data, in that order [BMPSTOR].

[[TODO: Maybe it is worth considering registering Windows cursors, animated cursors, and (re-registering) icons.]]

The key word "SHOULD" in this document is to be interpreted as described in [RFC2119].

3. Windows Bitmap Media Type Registration Application

Type name: image

Subtype name: bmp

Required parameters: None.

Optional parameters: None.

Encoding considerations: Binary.

Security considerations:

Bitmaps have a mostly unremarkable security history.

Because BMP data can encapsulate JPEG or PNG data (BI_JPEG, BI_PNG values of the Compression enumeration in Section 2.1.1.7 of [MS-WMF]), the security considerations of JPEG and PNG processing may also apply to BMP.

Interoperability considerations:

Uncompressed Windows Bitmaps can be rather large. If there is a need to compress an image, modern applications SHOULD consider emitting JPEG or PNG data instead of embedding them in BMP payloads.

Published specification: [MS-WMF] and [BMPSTOR].

Applications that use this media type:

Office productivity applications; clip art applications; desktop publishing applications; Web browsers; graphics processing applications.

Fragment identifier considerations: None.

Additional information:

Magic number(s): 42 4D ("BM"), meaning "bitmap". The next field (BITMAPFILEHEADER bfSize) is a little-endian DWORD indicating the size of the bitmap content in bytes.
File extension(s): .bmp, .dib
Macintosh file type code(s):
"BMP ", "BMPf", or "BMPp". Apple has promulgated a uniform type identifier (UTI) of "com.microsoft.bmp".

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? No

3. IANA Considerations

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

4. Security Considerations

See the registration template for security considerations.

6. References

6.1. Normative References

[BMPSTOR] Microsoft Corporation, "Bitmap Storage", MSDN ID dd183391, 2014, <http://msdn.microsoft.com/library/dd183391>.

[MS-WMF] Microsoft Corporation, "Windows Metafile Format", [MS-WMF], v20140502 (Rev 11.1), May 2014, <http://msdn.microsoft.com/library/cc250370>.

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

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

6.2. Informative References

[MS-OSP] Microsoft Corporation, "Open Specification Promise", February 2007, <http://www.microsoft.com/interop/osp/default.mspx>.

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/

Network Working Group
Internet-Draft
Intended Status: Informational
Expires: April 20, 2015

S. Leonard
Penango, Inc.

October 17, 2014

The Windows Metafile and Enhanced Metafile Media Types
draft-seantek-image-wmf-emf-00

Abstract

This document registers the image/wmf and image/emf media types for use with Windows Metafile and Enhanced Metafile formats. Originally designed for Microsoft Windows 3.0, these image files are intended to be portable between applications and devices, and may contain both vector and raster graphics.

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

Long before the invention of Scalable Vector Graphics, Microsoft Corporation recognized the value of recording images in a format that its applications and operating systems could easily render irrespective of the output device. With the release of Windows 3.0, Microsoft released its Windows Metafile (WMF) format, which can contain vector and raster graphics in one package. As a binary format that needed to work on 16-bit machines, WMF is comprised of a sequence of record structures. Each record contains drawing commands, object definitions, and configuration settings. When a metafile is processed, the image can be rendered on a display, output to a printer or plotter, stored in memory, or saved to some persistent storage. Reflecting the relationship to the Windows Graphics Device Interface (GDI) API, WMF metafiles are "played" to a playback device context in the same manner that PostScript content is treated as an executable program that results in the output of the original document.

As Microsoft's first 32-bit operating system, Windows NT 3.1 introduced an overhaul to the Windows API ("Win32") and the in-memory formats upon which those APIs relied. The Enhanced Metafile (EMF) format was created at this time, using 32-bit values instead of WMF's 16-bit values. In Windows XP, Microsoft extended EMF with "EMF+", adding support for Windows GDI+.

Many implementations of WMF and EMF were created because of Windows' commercial success in the 1990s. A large body of free and commercially available clip art and other artwork exists in this format. Furthermore, WMF content appears non-normatively in certain standards (e.g., [OOXML]); the usage is common enough that an implementer would almost certainly need to support it for basic interoperability.

Microsoft publicly documented the WMF format as early as the 1992 Windows 3.1 SDK. Since 2007 Microsoft has released the format specifications [MS-WMF] and [MS-EMF] under its Open Specification Promise [MS-OSP].

The key word "SHOULD" in this document is to be interpreted as described in [RFC2119].

2. Windows Metafile Media Type Registration Application

Type name: image

Subtype name: wmf

Required parameters: None.

Optional parameters:

DEFAULT_CHARSET: The character set intended when the CharacterSet Enumeration (see [MS-WMF]) specifies DEFAULT_CHARSET. The value of this parameter is a charset name defined in accordance to the procedures laid out in [RFC2978]. When this parameter is not specified, DEFAULT_CHARSET has the following meaning in [MS-WMF]: "a character set based on the current system locale; for example, when the system locale is United States English, the default character set is ANSI_CHARSET" (which is Windows-1252, more-or-less). I.e., when not specified, the default character set is system-dependent. As this optional parameter is novel, content producers embedding text SHOULD use EMF instead of WMF (or if absolutely necessary, SHOULD embed EMF within WMF).

Encoding considerations: Binary.

Security considerations:

The Windows Metafile format's security history is punctuated in 2005-2006 with the disclosure of the Metafile Image Code Execution vulnerability, codenamed MICE. MICE won the 2007 Pwnie Award for "Mass 0wnage" and "Breaking the Internet" [PWNIES07]. The official Microsoft security bulletin [MICE] describes that the flaw occurs because Windows Metafiles can set the SETABORTPROC value of the MetafileEscapes enumeration (accessible via the META_ESCAPE record), allowing for arbitrary code execution.

Windows Metafiles can contain Enhanced Metafiles using the META_ESCAPE_ENHANCED_METAFILE record; thus, the security considerations of EMF apply to WMF.

Windows Metafiles are historically very buggy. As the original intent was to replicate Windows GDI calls, flaws in GDI, or in a display or printer driver implementing the back-end to GDI, could be exploitable. WMF implementations not backed by Windows GDI have different risks: namely, while a malicious WMF author may not consider the non-Windows GDI implementation as a primary target, WMF has many "corner case" records for which an implementation's processing may not have received the same level of scrutiny as the Windows implementation. "Fuzzing" the implementation is appropriate.

Interoperability considerations:

Windows Metafile is the original 16-bit metafile format; it was

released in 1990 at what some computer historians might consider the "zenith" of the desktop publishing revolution. Accordingly, there is a large body of free and commercially available clip art that is still in use, either independently or embedded in productivity documents (word processing documents, desktop publishing documents, slideshows and presentations, and spreadsheets and workbooks). For example, references to WMF content appear (non-normatively) in Office Open XML [OOXML]. To say that support for this format is necessary for interoperability would not be an understatement.

Accommodations for comments or arbitrary data storage in Windows Metafiles are virtually non-existent. However, Windows Metafiles can contain Enhanced Metafiles using the META_ESCAPE_ENHANCED_METAFILE record; an implementation SHOULD be able to handle both types. Windows Metafiles can store and output text strings (see META_TEXTOUT and META_EXTTEXTOUT records), but the encodings of the strings may be ambiguous. Unicode encodings are not possible without the DEFAULT_CHARSET parameter defined in this registration.

The previously unregistered type "image/x-wmf" is also in wide use. Accordingly, it is registered as a deprecated alias. See Appendix A and Section 4.2.9 of [RFC6838].

Published specification: [MS-WMF].

Applications that use this media type:

Office productivity applications; clip art applications; desktop publishing applications; some Web browsers (e.g., Internet Explorer).

Fragment identifier considerations: None.

Additional information:

Deprecated alias names for this type: image/x-wmf
Magic number(s): D7 CD C6 9A (little-endian DWORD 0x9AC6CDD7)
File extension(s): .wmf
Macintosh file type code(s):
????. A uniform type identifier (UTI) of "????" is RECOMMENDED.

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? No

3. Enhanced Metafile Media Type Registration Application

Type name: image

Subtype name: emf

Required parameters: None.

Optional parameters: None.

Encoding considerations: Binary.

Security considerations:

Enhanced Metafiles are not afflicted with [MICE]. There has been no public disclosure of vulnerabilities specific to EMF or EMF+ to date. Nonetheless:

Enhanced Metafiles can contain Encapsulated PostScript (EPS) data; thus the security considerations of PostScript processing may also apply to EMF.

As the original intent was to replicate Windows GDI calls, flaws in GDI, or in a display or printer driver implementing the back-end to GDI, could be exploitable with maliciously crafted EMF content. EMF implementations not backed by Windows GDI have different risks: namely, while a malicious EMF author may not consider the non-Windows GDI implementation as a primary target, EMF has many "corner case" records for which an implementation's processing may not have received the same level of scrutiny as the Windows implementation. "Fuzzing" the implementation is appropriate. It is also possible that EMF+ data is "safe" while EMF data contains an exploit (or vice-versa); the EMF+-aware implementation (such as an application designed for GDI+ on Windows XP or above) would skip the "unsafe" data while another implementation would fall prey to the exploit.

Interoperability considerations:

Enhanced Metafile is the 32-bit metafile format; it was released in 1992 along with Windows NT 3.1. There is a large body of free and commercially available clip art that is still in use, either

independently or embedded in productivity documents (word processing documents, desktop publishing documents, slideshows and presentations, and spreadsheets and workbooks). To say that support for this format is necessary for interoperability would not be an understatement.

Enhanced Metafiles have extensive accommodations for comments and arbitrary data storage. Enhanced Metafiles can store and output text strings. Mercifully, the encodings of these strings are well-defined. Record examples include EMR_EXTTEXTOUTA (US-ASCII), EMR_EXTTEXTOUTW (UTF16-LE), EMR_POLYTEXTOUTA (US-ASCII), EMR_POLYTEXTOUTW (UTF16-LE), and EMR_SMALLTEXTOUT (UTF16-LE or the low-order 8 bits of UTF16-LE--effectively ISO-8859-1--depending on ETO_SMALL_CHARS).

Enhanced Metafiles can contain Encapsulated PostScript (EPS) data in the EpsData object [MS-EMF]. The FormatSignature EPS_SIGNATURE (0x46535045, in little-endian) is used instead of ENHMETA_SIGNAUTRE (0x464D4520, in little-endian) in such a case.

Windows XP introduced the GDI+ API, along with EMF+ [MS-EMF+]. EMF+ is actually an embedded format in which GDI+ commands are stored as EMF comment records (EMR_COMMENT_EMFPLUS record type). Content containing EMF+ data can be identified as "EMF+ Only" (only EMF+; the EMF records are not sufficient to reconstitute the drawing) or "EMF+ Dual" (both EMF records alone or EMF+ records alone, when played back, are sufficient to reconstitute the drawing) [MS-EMF+]. Support for EMF+ records may not be as extensive as support for the original EMF records.

The previously unregistered type "image/x-emf" is also in wide use. Accordingly, it is registered as a deprecated alias. See Appendix A and Section 4.2.9 of [RFC6838].

Published specification: [MS-EMF] and [MS-EMF+].

Applications that use this media type:

Office productivity applications; clip art applications; desktop publishing applications; some Web browsers (e.g., Internet Explorer).

Fragment identifier considerations: None.

Additional information:

Deprecated alias names for this type: image/x-emf
Magic number(s): 01 00 00 00 (little-endian DWORD 0x00000001),

corresponding to the EMR_HEADER Type field.
The next field (EMR_HEADER Size) should be
at least 88 (little-endian DWORD 0x00000050).
File extension(s): .emf
(for both EMF and EMF+ content)
Macintosh file type code(s):
????. A uniform type identifier (UTI) of "????" is RECOMMENDED.

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? No

4. IANA Considerations

IANA is asked to register the media types image/wmf and image/emf in the Standards tree using the applications provided in Sections 2 and 3 of this document.

5. Security Considerations

See the image/wmf and image/emf registration templates for their respective security considerations.

6. References

6.1. Normative References

[MS-WMF] Microsoft Corporation, "Windows Metafile Format", [MS-WMF], v20140502 (Rev 11.1), May 2014, <http://msdn.microsoft.com/library/cc250370>.

[MS-EMF] Microsoft Corporation, "Enhanced Metafile Format", [MS-EMF], v20140502 (Rev 10.0), May 2014, <http://msdn.microsoft.com/library/cc230514>.

[MS-EMF+] Microsoft Corporation, "Enhanced Metafile Format Plus Extensions", [MS-EMFPLUS], v20140502 (Rev 13.0), May 2014, <http://msdn.microsoft.com/library/cc230724>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate

Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2978] Freed, N. and J. Postel, "IANA Charset Registration Procedures", BCP 19, RFC 2978, October 2000.

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

6.2. Informative References

[MICE] Microsoft Corporation, "Vulnerability in Graphics Rendering Engine Could Allow Remote Code Execution (912919)", MS06-001, V1.0, January 2006, <https://technet.microsoft.com/library/security/ms06-001>.

[MS-OSP] Microsoft Corporation, "Open Specification Promise", February 2007, <http://www.microsoft.com/interop/osp/default.mspx>.

[OOXML] Ecma International, "Office Open XML File Formats", Standard ECMA-376, Fourth Edition, ISO/IEC 29500, December 2012, <http://www.ecma-international.org/publications/standards/Ecma-376.htm>.

[PWNIES07] Pwnie Awards LLC, "Pwnie Awards 2007", 2007, <http://pwnies.com/archive/2007/winners/>.

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/

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

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

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).

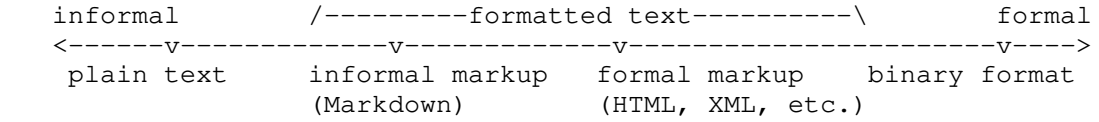


Figure 1: Degrees of Formality in Data Storage Formats for Text

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/

Network Working Group
Internet-Draft
Intended Status: Informational
Expires: May 19, 2018

S. Leonard
Penango, Inc.

November 15, 2017

The text/nfo Media Type
draft-seantek-text-nfo-05

Abstract

This document registers the text/nfo media type for use with release iNFOrmation. While compatible with text/plain, ".NFO" files and content have distinguishing characteristics from typical plain text because they are meant to be output to IBM PC-compatible system consoles that support certain "ANSI" escape sequences.

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

1. iNFOrmation

Packagers of files or other bundled content commonly include a common human-readable manifest that describes their packages. While an obvious solution is to include a README, in an archive such as a ZIP file, READMEs are generally written for software applications and provide late-breaking instructions on how to configure and install the software, along with known bugs and changelogs. (Plain) text READMEs are also generally limited to printable US-ASCII characters.

Starting from circa 1990, packagers of various types of content settled upon the Release iNFOrmation format (NFO, commonly pronounced "EN-foe" or "info") to describe their releases. An NFO file serves similar purposes to a README, but with several nuanced differences. NFOs usually contain release information about the media, rather than about software per-se. NFOs credit the releasers or packagers. Much like the Received: Internet Message header [RFC5322], intermediates ("couriers") can also insert NFOs.

Most distinctly, NFOs have come to contain elaborate ASCII or ANSI artwork that is remarkable in its own right in the pantheon of the postmodern computing culture. Many NFOs have been authored with the intent of displaying them on a terminal display with monospaced, inverted text (black background, gray or off-white foreground); some NFOs even include escape sequences to generate animations or color. The widely accepted encoding for NFOs is "OEM Code Page 437", the character set of the original IBM PC and MS-DOS.

When served in the same manner as plain text (text/plain), a lot of the elaborate artwork in NFOs is lost, garbled, or misaligned on display. As NFOs are still in considerable use, the goal of this registration is to rectify these interchange problems and reclaim this piece of living computer history.

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. Release iNFOrmation Media Type Registration Application

Type name: text

Subtype name: nfo

Required parameters:

charset: Per Section 4.2.1 of [RFC6838], charset is REQUIRED. Unlike most other text types, the default value is the character set of

the original IBM PC and MS-DOS, called OEM Code Page 437, and named "oem437". Implementations MUST support OEM Code Page 437. Unfortunately, the simple application of the IANA registered character set "IBM437" (aka "cp437") [RFC1345] will miss some important characters, so conformant implementations MUST support OEM Code Page 437 as specified in Section 3. NFOs authored for more modern computing environments are known to use ISO-8859-1, ISO-8859-15 (including support for the Euro sign), or UTF-8; however, for maximum interoperability, these or any other character sets MUST be declared by the sender. When absent, a receiver MAY guess, but SHOULD heavily bias the outcome towards OEM Code Page 437 unless UTF-8 encoding is patently obvious. A RECOMMENDED detection algorithm is provided in Appendix A.

Optional parameters:

baud: A natural number (integer greater than 0) indicating the gross bit rate ("symbol rate") at which the NFO is supposed to be rendered to screen. This optional parameter provides a nostalgic effect from the days of dialup modems and fixed-speed serial lines. It also controls the animation rate, to the extent that the NFO employs optional escape sequences. While the term "bps" might be more accurate, this parameter is meant to be interpreted the way that an end user would experience the real-world conditions that a dialup modem would provide on the eve of Y2K. (The term "baud" is also used by a couple of popular modern viewers of this format.) For example, a conforming implementation could implement "57600" as if the data were being downloaded using a V.92 modem, replete with random stalls due to retransmission attempts on account of noise on the line.

Encoding considerations:

Text with 8-bit code points; all 8-bit combinations (including NUL) are possible.

Security considerations:

It’s just text; this format provides no facilities for confidentiality or integrity. The ANSI escape sequence "CSI 5 m" could, however, blink you to death. As only a subset of ANSI escape sequences MUST be interpreted; interpreting a greater range than the subset prescribed in this registration may introduce other security issues, such as transmitting operating system commands.

Some code points in oem437 have been used ambiguously in practice, so implementations SHOULD NOT assume that the mapping between this charset and Unicode is bijective. When displayed, codes 00, 20, and

FF MAY appear to be similar, i.e., as a blank space.

Interoperability considerations:

NFOs are plain text but look best when read in a terminal view or with a dedicated NFO viewer that can emulate terminal features. As a result, they SHOULD be treated differently than text/plain files. The reference environment for NFO viewers to emulate is an IBM PC-compatible machine running MS-DOS 6.22 with the ANSI.SYS MS-DOS device driver loaded, where the NFO is displayed as if it were output to the terminal using the "TYPE" command.

Published specification: [[Note to RFC Editor: Insert number here.]]

Applications that use this media type:

NFO viewers; text editors; terminals.

Fragment identifier considerations:

Same as text/plain [RFC5147].

Additional information:

Deprecated alias names for this type: text/x-nfo
File extension(s): .nfo
Macintosh file type code(s):
TEXT. A uniform type identifier (UTI) of "public.nfo", which conforms to "public.plain-text", is RECOMMENDED.

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? No

3. OEM Code Page 437

"OEM Code Page 437" refers to the character set of the original IBM PC and MS-DOS. The code page actually represents two related things: the set of 256 graphemes stored in video read-only memory (ROM) that are accessed with a single 8-bit code, and an 8-bit encoding for text

content that displays the graphemes or causes other behavior as defined by the code, the operating system, and the loaded device drivers. NFO is encoded with the aforementioned 8-bit encoding, which means that not all 256 graphemes are directly available for use.

For example: the sequence 0D 0A (CR LF) identifies a new line; the code 1A (SUB) is the MS-DOS end-of-file marker. The code 0D cannot be used directly to express the grapheme U+266A EIGHTH NOTE; the code 0A cannot be used directly to express the grapheme U+25D9 INVERSE WHITE CIRCLE; the code 1A cannot be used to express U+2191 RIGHTWARDS ARROW.

The registration for IBM437 [RFC1345] is used as a basis for this specification, which only elaborates upon the differences. Suggested mappings to Unicode characters are included; however, the mapping is not bijective. Octets are in hexadecimal. The symbols below next to the octets match [RFC1345], although the actual character has the meaning described here rather than the [RFC1345] meaning.

3.1. Low-Order Codes (00-7F)

The codes in the 20-7E range are the same as in US-ASCII and IBM437.

01-06, 0B, 0C, 0E-19, and 1C-1F are displayed as their corresponding ROM graphemes.

00 NUL is displayed (and treated) as a space. Depending on the output environment, an implementation MAY map this code to U+0000 NULL, or U+0020 SPACE.

07 BEL MAY cause an audible bell sound (beep) to be emitted. Actually emitting a sound is not required for conformance. However, implementations that progressively render the output MUST pause for this code as if a sound were emitted.

08 BS causes the prior character to be erased: the prior grapheme is displayed and treated as a regular or non-breaking space (SP or NBSP), depending on whether the prior character would have been breaking or non-breaking.

09 HT causes horizontal tabbing, which for purposes of conformance, SHOULD produce the equivalent spaces so that the subsequent text is aligned on the next 8-character boundary.

0A LF causes a new line to be created and the text insertion point ("cursor") to be moved to the beginning of that line.

0D CR causes the text insertion point ("cursor") to be moved to the

beginning of the current line. Subsequent text will overwrite the characters on the current line, until the cursor moves somewhere else. (0A creates and moves the cursor to a new line; therefore, 0A in the middle of overwriting the current line will not insert or erase any characters that might otherwise be on that line.)

1A SUB is the MS-DOS end-of-file (EOF) marker; it ends the display. Codes after 1A MUST NOT be displayed. 1A can be used to delimit metadata from the main NFO content, although this practice is rarely used for NFOs. A well-known metadata format in this technology area is SAUCE (Standard Architecture for Universal Comment Extensions) [SAUCE], which implementations MAY support. A SAUCE record can specify a different code page. An implementation that supports SAUCE SHOULD support following the code page directive in the SAUCE record when the MIME entity's charset is oem437.

1B ESC may be the start of an ANSI ESC sequence. If no valid ESC sequence is recognized, output the corresponding ROM grapheme (U+2190 LEFTWARDS ARROW) and continue normal processing with the next code.

7F DEL is displayed as the corresponding ROM grapheme (U+2302 HOUSE).

3.2. High-Order Codes (80-FF)

The codes in the 80-AF range are a selection of Latin characters; they are the same as in IBM437. A conformant implementation MUST NOT treat these codes as C1 control characters.

The codes in the B0-DF range are box drawing and block characters; they are the same as in IBM437.

The codes in the E0-FF range are for mathematical symbols, which are the same as in IBM437, with the following exceptions. The preferred Unicode mapping in Microsoft's OEM Code Page 437 documentation is designated with [OEMCP437]:

E1 b* can be either U+03B2 GREEK SMALL LETTER BETA, or U+00DF LATIN SMALL LETTER SHARP S (German Eszett) [OEMCP437]. The two were undistinguishable at low resolution on the original IBM hardware. Newer grapheme sets, including those of the IBM EGA and VGA graphics cards, display this code as the Eszett. Unfortunately only context can determine the proper character to use.

E3 p* can be U+03C0 GREEK SMALL LETTER PI [OEMCP437], U+03A0 GREEK

CAPITAL LETTER PI, or U+220F N-ARY PRODUCT, depending on the particular grapheme used.

E4 S* can be either U+03A3 GREEK CAPITAL LETTER SIGMA [OEMCP437] or U+2211 N-ARY SUMMATION.

E6 m* can be either U+00B5 MICRO SIGN [OEMCP437] or U+03BC GREEK SMALL LETTER MU.

EA W* can be either U+2126 OHM SIGN or U+03A9 GREEK CAPITAL LETTER OMEGA [OEMCP437].

EB d* is U+03B4 GREEK SMALL LETTER DELTA [OEMCP437]. However, it can be used as a surrogate for U+00F0 LATIN SMALL LETTER ETH (Icelandic, Faroese, Old English, IPA) or U+2202 PARTIAL DIFFERENTIAL.

ED /0 is U+03C6 GREEK SMALL LETTER PHI [OEMCP437], but in MS-DOS was mainly used as U+2205 EMPTY SET. Other possible meanings include U+03D5 GREEK PHI SYMBOL (used as a technical symbol, with a stroked glyph) (to name angles), U+2300 DIAMETER SIGN, or U+00F8 SMALL LETTER O WITH STROKE (as a surrogate).

EE e* is U+03B5 GREEK SMALL LETTER EPSILON [OEMCP437] or U+2208 ELEMENT OF.

FF NS is NBSP, also known as U+00A0 NO-BREAK SPACE. The ROM grapheme is the same as SP (SPACE), i.e., it is blank.

3.3. ANSI Escape Sequences

To support NFO content containing colors and other goodies, an NFO viewer MUST support a subset of "ANSI" escape sequences. (The required sequences are not directly related to ANSI, but rather to [ANSI.SYS].)

[ANSI.SYS] supports cursor positioning, erasing, Set Graphics Mode (SGR), mode switching, and keyboard remapping. Of these functions, a conforming implementation MUST support the Set Graphics Mode (SGR) escape sequence. An implementation MUST support setting foreground colors (30-37) and background colors (40-47), which are also in [ISO6429]. An implementation MUST support all of the [ANSI.SYS] text attributes (0, 1, 4, (5 and/or 6), 7, and 8). Text attribute 5 is "Blink: Slow" (less than 150 per minute); text attribute 6 is "Blink: Fast" (more than 150 per minute). While [ANSI.SYS] does not document attribute 6, that was the behavior of the actual ANSI.SYS. An implementation SHOULD reproduce similar functionality.

The other [ANSI.SYS] escape sequences are OPTIONAL. An implementation MAY support standard or vendor-specific escape sequences. For a list of standard sequences, see, e.g., [ISO6429] and [ISO8613].

3.4. Accessing Hidden Grapheme Codes

There is no obvious way to encode the graphemes that are inaccessible at the values 07, 08, 09, 0A, 0D, 1A, and 1B. This specification provides a technique to access these graphemes in the context of OEM Code Page 437. This technique is RECOMMENDED, but not required.

Although MS-DOS and ANSI.SYS did not conform to [ISO2022], that standard defines escape sequences to switch to other character sets. Unicode contains appropriate code points for all of the inaccessible graphemes (characters). Accordingly, the escape sequence:

```
   ESC % G
```

switches the code to UTF-8 (with unspecified implementation level) [REG196]. While in UTF-8, the escape sequence:

```
   ESC % @
```

reverts the code back to the original [ISO2022]. Normally the code would be [ISO2022], but given the starting context of OEM Code Page 437, the code returns to OEM Code Page 437. The codes are as follows:

```
ROM grapheme number
|  IBM437 symbol
|  |   Unicode code point
|  |   |       Unicode name: UTF-8 encoding
|  |   |       |
07 BEL U+2022 BULLET: E2 80 A2

08 BS  U+25D8 INVERSE BULLET: E2 97 98

09 HT  U+25CB WHITE CIRCLE: E2 97 8B

0A LF  U+25D9 INVERSE WHITE CIRCLE: E2 97 99

0D CR  U+266A EIGHTH NOTE: E2 99 AA

1A SUB U+2192 RIGHTWARDS ARROW: E2 86 92

1B ESC U+2190 LEFTWARDS ARROW: E2 86 90
```

3.5. UTF-8/Unicode Processing

When NFO content is encoded in UTF-8 or another Unicode encoding [UTF], the C0 and C1 code points may be present. These codes MUST be treated as control codes, not graphemes. They have the same behavior as specified for the special low-order codes described in Section

3.1. For example, 1A ends the display, and 09 emits spaces sufficient for 8-column tabbing. 1B is ALWAYS treated as the start of an ESC sequence; if the sequence is not recognized, 1B does NOT revert to outputting a LEFTWARDS ARROW grapheme. Instead, nothing is displayed. For LEFTWARDS ARROW, encode U+2190 instead.

The C1 control code 9B (CSI: Control Sequence Introducer) (Unicode code point U+009B) MUST be recognized as such; it is equivalent to 1B 5B (ESC [).

3.6. Grapheme Reference

The following figure is a reference of all 256 graphemes in the IBM PC ROM. The figure is a MIME (base64)-encoded PNG image.

```
MIME-Version: 1.0
Content-Type: image/png
Content-Disposition: attachment; filename="Codepage-437.png"
Content-Transfer-Encoding: base64

iVBORw0KGgoAAAANSUhEUgAAASwAAACMCAMAAADxyGQdAAAAGXRFWHRTb2Z0d2FyZQBBZG9i
ZSBJbWFnZVJlYWR5ccllPAAAAAZQTFRFqKioAAAAmKDP8QAACYZJREFUeNrsXYt24zoIhP//
6XvO7daVmBlAfrRJq+xuk20SWxojBMPD5vvRftiGYIO1wdpgbbA2WJ3D/Xtkv9lg3QyWLY3y
u66G8emOL+4DKz4f5yDfmo8an4YR2ud/4kjki9vAss+nXCLUCArJsmNen6hNc/fh53yIGerj
3z+kyCin6/EcWPbxYxrKcT49lJ5keQQLJCvI3tfrQeQjWG4oV7g+lsAaJd/6kjUvoxksEL4B
mmFqg+igZI1vD3+nc9FlSC9nZxlqUbMBpYtgDc8fP4xJxPHjQOd4ikoHwApiFLQZ01l+RmeR
k8PAjGwdZBn+P4SvA+K4GVggR3ZcqeGAcfGiZI1HLiTLw0rsgcWmA8ti2pIysP5NNSLCJMth
o+uBxZ+jZFU66wBrVF2JqlqQrLgl9XdD9s3kuqlliGrdp43Kgs6Su+EsPuOBiJUySTOT5/lb
n0oV1XTXdBgwWjAdoqyhZM0XIei+hp01avYgWVSdt7fD6TirRqk1rJjadDBjYjiANb6VGDha
Z3WX4Vu7O1en8WPuznakN1gbrNcGy/42V2gpMNIs/pswmpw3gkWWE+M/zjj5rjkWj26aRRsE
bUj0Ok7umLm7g7TR7H2BtQjQ3GPXgE87ezrAjwn3iX6rXhnohhVgHXzF7KraaDeagcWNInZG
2sfrxY5IHFI6YwGx8Ba5z9ACK9r+UfadsW4IFljnK0KlpoQifwks7T/OE6p19SA+zGEB7xAv
zxKxOxOBiZcXuPjIRYYxt0hytRJPgAXaO9IDDKzAXpjwV1EcAawY0DDABsUnk7xCXZDVcA4s
uWFeliycKl3DKRcllU4lWWY41gIsxVfgpWqBxXSWRC1yOAVpcw6sTpSMjnBBsvh+3IgUdASf
6XK+oOBy0bcYWIBwuRteWIaHHh0VkkQrW3MSLGbuxchF4Oe55QkMvqUbtxDc5WX4gFPzrr7R
dqR/jqLZYO3HBmsdlw2WMi6bdhY6lWSDhuSXSOPEDX4IvaLP5CEUbGANqN/MobBg6tAXMbLJ
6SkzDRaGnanTjlQXMVGF404syq8Mgfh24zdjBHsFrGisGnPjedIX+QxJNHM0kscXM9W1ANZX
os4JsMbEAAFWnIV50+nMFBnxGNCfmUTM8eReghVphJjV1AGLe58KLCR+HdMBbBksHC4jbefF
Pidnucg2wNSG0Y/6yl6TZ5eZLQ2wSK4DPttZsEg2Ykey4mx7YPmQM2JMYxD9wMmJCFbGHaLL
b5clKwOrChAQhkaweaChK1rBPSUkQPHzxRde2TJYrsFKFPwCWPPXm2BZteGyMY9akBCKDsrP
FhV8QhAlpoN7ORNhcHTBsjZYLP1bU4lMsXfAQoXaNUqHS2Ni3FNSIISLpFEaB0ZzGGMkBTOa
nab2glGqSK93c3dehhh6D7Ds5848RZC207wpmmfBsg3bBuuZZYgZ7Q1Va8m3wktjBvMPKxET
LrVxQmslEMoQrfNCmA34IiE1XSciZOUqWG7d8b8PWAqgRvFEYjHPBvzMDcCyI27aWDkYKOPh
N/IzmHuUE6SiJi8Dy6qINHVDnCY9kApCkjkJHMvkNlnFiwJqKpmnYkrxgJVWnvUMr91xRtFA
ITdLp+F5ftEVi0lYPbAwbbEGi0RSOkWuXNeyZIkaLOarfwdYnE+hYDG+VKZdviJYlPZbB0vX
1PFac1NRm4fA4rNZBotkWS6DxasYCZtkUme1s9qKgvJxX8O9z/qsU6jcZW8hWPyk4lzuccfM
+Czab8OSOM+jjnSWsLluELXDU99ql902itcH6/JxbhtHVgN1Biuovr+Bxtt81h8m/5oRvBcE
K/brwcVvueepJi0ahvAys3rQoiyMWkzMdDBqMeAUsS+HzAUgJhShfUxazMSqMCsYJuGI6VOw

Leonard Exp. May 19, 2018 [Page 9]
```

8HNVohDK3ky0jhjjhq6oqayHkKW1K0K2aM1FZy/TmWYyUwagDjHKsfsSsV5pHjxtPXHXdrii
qngvptMWSLIMEd8saotgwVoxzXA5rKfCBEdEpkUn6BWHxMDUyk8zzajkoKvhQzJHEyxa9ZIR
sdkFLHyxJNdhaOgTE96KRM02WJhcR0oZK7B0TbZJypQ22yqZSTpBg5y31vwvgBVUPiE7gs7S
CTLpOSmfo/2WsgFYzCX0p8Ca9wSZqMR3w5Ng6druHliephy5PwCW44URXae8tLOKEYhkvFt1
1hSnuB8siJv0wYKIA6Wx1Av33m7IX7ApjQyZjXmnhQ7FEepuGFhqPu7/Umdd9sZeuTfP+ZEF
+9fupDB+G1bRQ/ntFM2tMrD5rKfBEkkm94/sI7H76HWq+6lkTn/tiJO3ihK6hBRix8Z9ras6
ek1Wjn8ff74azL4CWA2sUKC6Z6nN1LeSrBZWrApjESyb6SD96ZeWrNX9lJiDq4zS2+osFqU1
Uo0ythZfs2qokxQmOQx2QbKAoeBUtEEZ2lAngm/lkpXUpUVqgeSTYSS8T9HQq36XZLFazjQ/
sidZfbC0rccpcFH5R4Z/4FFLFvcRUWoSyXJOvamuiqS5UwcspaOSjnKKun5GslBGkvLyrrN0
FqysnRlr2hkyaZDev6CzfiNYB0Hsbm5lydrrSpabGakTjCRRvfcX7IVqc3W7nfUoWN/Nljxt
Z70/WNbvlbgl6xtZhz8F1otL1lIfys4oBZFd/GbFy8tbrhf9Pg3a+sN/LHxmg7XB2mBtsP4W
WEf41mLDI4sNsqYPhwf9DJtSqGkU90QlX39Nyarp9YbBsiZZ8pZrj0kWf3TeYg24pPh0JCt5
aPLR3KuvpjNFsBY4+I6saRXTorobctTRYh2jlHrzG6wN1gZrg7XB2mBtsDZYPwLWGz2+J8Xz
14NVBJvGdhnr1fcm67IuhCmY87Uwp/rY2fosnVRsOtQES5UbdeHDe3tXhAp2Pa+lR7vvV8Aq
V3MNFuk/zj5N2lDojjkaK19q3NTgOloxX2fNxzVYhr07PBkRtvAtiybiPuSkeobGusUt8FIc
mynCpL1WA6wxBZL0AuJCw2pcst4vE6039SIy0rgJ1rV3Ou+E6SyChf3VJFjuuWTxNa7Ampqs
uZMWRKGQsFyGtWRZU+dVwrcAFtYuXpCsFKxYppgvwzpfL0VCphs+Ahaa0vTmiUlGs7otaApW
vhs6y3tsLcNrYFkOVriHi+fqI6kRL1Q33kK0vfWvmfLZTVqv7oZzDaAAi93/2sUtBufqgdS4
Mx7JaZd33GMJXzLRf7zo7psH8N6+4QZrg7XB+muPDdYGa4O1wdpg/dLHfwIMAORIgm4Mk35I
AAAAAElFTkSuQmCC

Figure 1: Code Page 437 Grapheme Reference

3.7. Charset Registration Template

To: ietf-charsets@iana.org
Subject: Registration of new charset oem437

Charset name: oem437

Charset aliases: None.

Suitability for use in MIME text: Suitable.

Published specification(s): This specification; [OEMCP437].

ISO 10646 equivalency table:

> This table is taken from the IBM437 registration in [RFC1345], with modifications based on actual implementations of [OEMCP437], as discussed in this document. Character mnemonic symbols generally map to the Unicode code points listed in Section 3 of [RFC1345], with the following exceptions. The symbol suffix $ (for example, HT$) means that the Unicode code point mapping is essentially correct, but an implementation might need to perform additional or special processing as discussed in this document, depending on the output environment.

The symbol $$ means that this code point has special considerations as discussed in this document, so no single, definitive Unicode code point mapping can be given. Finally, three characters have no corresponding mnemonic symbols in Section 3 of [RFC1345], so symbols are defined here:

```
  $>   25ba   BLACK RIGHT-POINTING POINTER
  $<   25c4   BLACK LEFT-POINTING POINTER
  $B   21a8   UP DOWN ARROW WITH BASE
```

```
NU$ 0u 0U cH- cD- cC cS BL$ BS$ HT$ LF$ Ml Fm CR$ M2 SU
$> $< UD !*2 PI SE SR $B -! -v $$ EC$ -L <> UT Dt
SP !  "  Nb DO %  &  ’  (  )  *  +  ,  -  .  /
0  1  2  3  4  5  6  7  8  9  :  ;  <  =  >  ?
At A  B  C  D  E  F  G  H  I  J  K  L  M  N  O
P  Q  R  S  T  U  V  W  X  Y  Z  <( // )> ’> _
’! a  b  c  d  e  f  g  h  i  j  k  l  m  n  o
p  q  r  s  t  u  v  w  x  y  z  (! !! !) ’? Eh
C, u: e’ a> a: a! aa c, e> e: e! i: i> i! A: AA
E’ ae AE o> o: o! u> u! y: O: U: Ct Pd Ye Pt Fl
a’ i’ o’ u’ n? N? -a -o ?I NI NO 12 14 !I << >>
.S :S ?S vv vl vL Vl Dl dL VL VV LD UL Ul uL dl
ur uh dh vr hh vh vR Vr UR DR UH DH VR HH VH uH
Uh dH Dh Ur uR dR Dr Vh vH ul dr FB LB lB RB TB
a* $$ G* $$ $$ s* $$ t* F* H* $$ $$ 00 $$ $$ (U
=3 +- >= =< Iu Il -: ?2 Ob .M Sb RT nS 2S fS NS$
```

Additional information:

See this document for details on how to handle particular codes that correspond both to graphemes in the IBM PC ROM, and to control characters.

Person & email address to contact for further information:

Sean Leonard <dev+ietf@seantek.com>

Intended usage: COMMON

4. Example

The following example is a RELEASE.NFO file as an e-mail attachment, with base64 encoding. Note that the character set is (correctly) assumed to be OEM Code Page 437.

```
MIME-Version: 1.0
Content-Type: text/nfo
Content-Disposition: attachment; filename="RELEASE.NFO"
Content-Transfer-Encoding: base64

TODO/PutInBase64EncodedContentHere==
```

5. IANA Considerations

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

IANA is asked to register the charset oem437 in the Character Sets registry using the application provided in Section 3 of this document.

6. Security Considerations

It’s just text; this format provides no facilities for confidentiality or integrity. The ANSI escape sequence "CSI 5 m" could, however, blink you to death. As only a subset of ANSI escape sequences MUST be interpreted; interpreting a greater range than the subset prescribed in this registration may introduce other security issues, such as transmitting operating system commands.

Some code points in oem437 have been used ambiguously in practice, so implementations SHOULD NOT assume that the mapping between this charset and Unicode is bijective. When displayed, codes 00, 20, and FF MAY appear to be similar, i.e., as a blank space.

7. References

7.1. Normative References

[ANSI.SYS] Microsoft Corporation, "ANSI.SYS", MSDN ID cc722862, 1994, <http://technet.microsoft.com/library/cc722862>.

[OEMCP437] Microsoft Corporation, "OEM 437", MSDN ID cc305156, 2014, <http://msdn.microsoft.com/goglobal/cc305156>.

[RFC1345] Simonsen, K., "Character Mnemonics and Character Sets", RFC 1345, June 1992.

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

[RFC5147] Wilde, E. and M. Duerst, "URI Fragment Identifiers for the

text/plain Media Type", RFC 5147, April 2008.

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

[UTF] The Unicode Consortium, "The Unicode Standard, Version 8.0.0", Chapter 3: "Conformance", The Unicode Consortium, August 2015.

7.2. Informative References

[ISO2022] International Organization for Standardization, "Character Code Structure and Extension Techniques, 6th edition", ISO Standard 2022, ECMA-35, December 1994.

[ISO6429] International Organization for Standardization, "Information Technology - Control Functions for Coded Character Sets, 3rd edition", ISO Standard 6429, December 1992.

[ISO8613] International Organization for Standardization, "Information Technology - Open Document Architecture (ODA) and Interchange Format: Character Content Architectures", ISO Standard 8613-6, ITU-T T.416, March 1993.

[REG196] International Organization for Standardization, "International Register of Coded Character Sets: UTF-8 without implementation level", Sec. 2.8.1, Reg. 196, April 1996, <http://kikaku.itscj.ipsj.or.jp/ISO-IR/196.pdf>.

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

[SAUCE] O. "Tasmaniac" Reubens / ACiD, "SAUCE--Standard Architecture for Universal Comment Extensions", 00.5, November 2013, <http://www.acid.org/info/sauce/sauce.htm>.

Appendix A. IBM Code Page 437 vs. UTF-8 Detection Algorithm

In cases of ambiguity, the following algorithm SHOULD be used to detect UTF-8 encoded data in text/nfo content:

If the octets EF BB BF are present at the beginning => UTF-8.

Considering all octets in the content:

```
     If no octets are greater than 7F => oem437.
     If any octets are F5 - FF, C0, or C1 => oem437.
     If any UTF-8 encodings are "ill-formed" => oem437.
     If any UTF-8 encodings represent illegal code points
       (e.g., surrogate code points) => oem437.

     Ragged line tests:

       If display characters decoded with oem437
         result in identical line widths => oem437.
       If display characters decoded with UTF-8
         result in identical line widths => UTF-8.

   Finally:
   => UTF-8 or oem437; prefer oem437.
```

Author's Address

Sean Leonard
Penango, Inc.
5900 Wilshire Blvd
Ste 2600
Los Angeles, CA 90036
USA

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 19, 2015

D. Singer
Apple, Inc.
A. Begen
Cisco

October 16, 2014

URLs and HTTP Response Forms for Multicast
draft-singer-appsawg-mcast-url-00

Abstract

This document motivates the need for defining a URL for multicast delivery and provides a proposal for this purpose.

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."

This Internet-Draft will expire on April 19, 2015.

Copyright Notice

Table of Contents

1. Introduction

Various multicast file-delivery protocols are defined by the IETF and 3GPP (notably File Delivery over Unidirectional Transport (FLUTE) and FCAST). However, they are hard to adopt into other services, partly because they do not follow conventions on how these transports are addressed and what information they deliver. Notably:

1. Much of the Web (Internet) assumes that if a file can be used, it can be referred to by a URL that contains enough information to start to try retrieving it. This is not true for files available over multicast.

2. When a URL form is used, it can be annotated with the information on what it refers to (e.g., a MIME type, a codecs parameter for that MIME type, and so on). If we have no URL, we cannot annotate it.

3. HTTP header responses are widely used to signal the unavailability of an expected resource (404) or that a resource has moved (re-direct), or that there are other choices to retrieve the indicated resource, or to deliver portions of a resource (byte-ranges). Though FCAST uses the metainformation format of HTTP, it misses the status line, so neither unavailability nor re-direct can be signaled. You cannot re-direct from multicast to HTTP, for example.

4. 'Soft' information such as multicast group addresses, transport session identifiers, and so on, are hard-coded into the descriptions (e.g., Session Description Protocol (SDP) files [RFC4566]). In general, the Web/Internet avoids hard-coding such values, preferring to use lookup (e.g., DNS for addresses); lookups can be re-factored as boundaries are crossed.

Traditionally multicasts have been addressed by requiring the client to acquire some pre-knowledge (e.g., an SDP file) by some means out of band. Thus, we require that every protocol that might use multicast be adapted. This is error-prone, limiting, and time-consuming. Instead, if an operating system can have a URL handler for multicast URLs that deliver file objects, with an interface that 'emulates' the interface to HTTP, many (not all) things would 'just work'. Perhaps the most notable is that we might re-direct from HTTP to multicast when the server detects that there is a better way to get the resource (perhaps, at this time and for this client).

The places where URLs occur, and where it would be advantageous to be able to state "this file is available on multicast", are legion. Obvious examples include anything linked into HTML (a Web page or email), especially media (video, audio, images); in HTTP itself where re-direction supplies a URL, and HTTP adaptive streaming systems where many clients could be fetching the same set of content segments. For many of these, operating system support with the same API as HTTP would suffice. Even in the HTTP adaptive streaming case, where it is true that the streaming engine needs to know it is using multicast (as this would make substantial changes to bandwidth estimation, etc.), simplifying the markup and the protocol identification to a URL is a plus. FCAST is closer to HTTP operation than FLUTE; files 'just arrive' and there is no concept of the 'set' as represented by the file delivery table in FLUTE. We therefore focus on FCAST in this document.

2. Glossary of Terms

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Use Cases

Here are two example use cases.

1. The classic stadium. A sports franchise wants everyone in the stadium to be able to watch a few selected camera streams. They multicast the streams over a tuned WiFi system.

2. A network operator (either Internet service or 3G/4G) wants to enable people to see a video mosaic of the top channels, and click through to get to a channel fast.

The simple solutions are:

1. Provide a QR code that embeds a multicast URL linking to the manifest file for the video content at the entrance, in printed material, on posters, etc. When people tune to that multicast URL with their phones, they get the manifest, and it refers to streams that are also multicast. The act of tuning into the session starts the client caching everything that arrives.

2. The mosaic is a multicast URL, and the segments of each program are also multicast but with short cache-times, and using the same URL label as the unicast address (i.e. ,an HTTP URL). When the user clicks on a program, they fetch the manifest (or perhaps the manifests are also multicast and pre-filled into the cache) and they already have the current segment cached, so startup is effectively instant. As they proceed, there is a good chance the multicast has delivered every segment they need, just in time.

4. Required Information

Currently, tune-in to a multicast involves getting hold of a 'head' file that gives a variety of information. The possible information can be roughly separated into different classes:

1. Information about alternatives that could be supplied as part of the higher-level protocol (e.g., different representations in HTTP adaptive streaming and HTML5 source elements)

2. Information (IP addresses and the like) that is needed to 'bootstrap' the multicast reception

3. Information about where/how the reception is possible (e.g., protocol parameters, time-ranges, and so on)

4. Information that could be acquired later, in-band, such as feedback addresses, the availability of alternatives and unicast repair servers, and so on (or indeed, a fuller description of the multicast itself)

For the sake of simplicity, we propose that we only include (2) and (3) in the URL form.

Ideally, there is something about the multicast itself that allows the client system to assess fairly rapidly whether it is working (the multicast join succeeded, packets are arriving, etc.) and if that fails, the URL handler can give a suitable error indication (maybe an existing one, maybe new).

5. Suggested URL Form

5.1. Introduction

Both FLUTE and FCAST rely on Asynchronous Layered Coding (ALC) [RFC5775] / Layered Coding Transport (LCT) [RFC5651], which in turn has the concept of channels to handle congestion and rate control. We presume the existence of a base channel and indicate how to acquire that.

In an FCAST session, files are identified by URI labels. We suggest that we identify a reserved URN form to indicate 'this is a complete SDP file describing all the sessions'. This allows bootstrapping from the base channel to all of the channels in a session.

FLUTE [RFC6726] is specific about the parameters needed to acquire an ALC/LCT session, and since FCAST [RFC6968] also relies on ALC/LCT, the same analysis applies.

5.2. Information on FLUTE (RFC 6726)

To start receiving a file delivery session, the receiver needs to know transport parameters associated with the session. Interpreting these parameters and starting the reception therefore represents the entry point from which thereafter the receiver operation falls into the scope of this specification. According to [RFC5775], the transport parameters of an ALC/LCT session that the receiver needs to know are:

- The source IP address.
- The number of channels in the session.
- The destination IP address and port number for each channel in the session.
- The transport session identifier (TSI) of the session.
- An indication that the session is a FLUTE session. The need to demultiplex objects upon reception is implicit in any use of FLUTE, and this fulfills the ALC requirement of an indication of whether or not a session carries packets for more than one object (all FLUTE sessions carry packets for more than one object).

5.3. URL Form Requirements

We have at least the following requirements:

- The URLs must be valid according to the RFCs and recent work at the W3C
- The absolute form must exist (obviously)
- Relative URLs must also work
- We should avoid the fragment (#) and query suffices (?) even though, in the latter case, there is no server that the URL is sent to.
- We should permit the URL to self-declare its validity period (and thus enable rapid time-outs when it is requested outside this period)
- Ideally, we also allow it to indicate its 'geographic' (operator network) availability scope

5.4. Base URL Form

This suggests a URL form in three parts:

- A prefix giving the URL scheme and basic parameters
- A mid-part giving the temporal and geographic scope
- A suffix that is the label of the desired file

Where the prefix is roughly like

fcast://destination:port/source:TSI

with

destination: An explicit multicast address (x.y.z.w) or (better) a name that resolves to one (or more) IP multicast address(es) for the base channel.

port: Port number for the base channel.

source: An explicit IP address (x.y.z.w) or (better) a name that resolves to the source address.

TSI: The transport session identifier for the session.

The mid-part has optional terms that are each formatted as /key:value. The keys and their values are:

start: the absolute start-time of availability

end: likewise, the end time

network: an identification of the network(s) on which the multicast is made available (for the indicated time-span, if any)

The start and end times are each optional and if present are expressed exactly as in SDP, i.e. as the decimal representation of Network Time Protocol (NTP) time values in seconds since 1900. If the URL agent determines it is operating outside this time range, a suitable error SHOULD be returned immediately. If either the start or end time are absent, then the multicast starts (or stops) at an indefinite time.

The network attribute takes a list of domain names, joined by the plus sign; if the URL handler is confident that the machine is not on any of the networks, a suitable error SHOULD be returned immediately, as it knows the multicast reception will not succeed.

The suffix starts with the special key /label: and is followed by the label of the desired file. (We retain at least the forward slashes in the path in the clear so that relative URLs work, but perhaps some characters and maybe some instances of slash should be escaped.)

Example:

fcast://232.0.0.1:5620/broadcast.example.com:527353/start:35776638264/network:media.example.com/label:http://news.example.com/stuff.mp4

given such a URL, the terminal can (try to) tune into the FCAST session, and retrieve the indicated file.

6. FCAST Metainformation field

In FCAST the metainformation field carries anything that an HTTP metainformation field can carry, but not the status line. This means it is not possible to indicate "this file might be expected here, but it is not here any more" (404) or "this file has moved" (301 or 307) or even that there are multiple choices on where to get this resource (300 'choices'). The most useful, perhaps, is the ability to indicate "you might have expected to get this over this multicast, but it's not here, but over there (re-direct)" perhaps even re-directing back to HTTP, or to another multicast session.

We therefore suggest we define a new form of the FCAST metainformation that also includes a status line formatted exactly as the HTTP status line, but with the HTTP-version replaced by FCAST-version:

Status-Line = FCAST-Version SP Status-Code SP Reason-Phrase CRLF

7. HTTP Status Codes and Their Applicability to FCAST

Here are the status codes available in HTTP 1.1, and a brief statement of whether they could be applicable to FCAST:

7.1. Useful

- 200 OK: Usual status code when the object is supplied, or when just the metainformation is supplied
- 203 Non-Authoritative Information
- 206 Partial Content: Useful to indicate that byte-ranges of the resource are supplied separately
- 300 Multiple Choices: Useful to indicate that there are also other places to get the content
- 301 Moved Permanently: The resource might be expected here, but has moved (re-direct)
- 302 Found
- 303 See Other

- 307 Temporary Redirect: The resource might be expected here, but has moved (re-direct)
- 404 Not Found: The resource might be expected here, but it is no longer available
- 410 Gone

7.2. Unlikely but Possible

- 100 Continue: Unlikely to be of use
- 101 Switching Protocols: Maybe useful
- 502 Bad Gateway: The multicast was being fed by a gateway that failed
- 503 Service Unavailable

7.3. Probably Inapplicable

- 201 Created
- 202 Accepted
- 204 No Content
- 205 Reset Content
- 304 Not Modified
- 305 Use Proxy
- 400 Bad Request
- 401 Unauthorized
- 402 Payment Required
- 403 Forbidden
- 405 Method Not Allowed
- 406 Not Acceptable
- 407 Proxy Authentication Required
- 408 Request Time-out

- 409 Conflict
- 411 Length Required
- 412 Precondition Failed
- 413 Request Entity Too Large
- 414 Request-URI Too Large
- 415 Unsupported Media Type
- 416 Requested range not satisfiable
- 417 Expectation Failed
- 500 Internal Server Error
- 501 Not Implemented
- 504 Gateway Time-out
- 505 HTTP Version not supported

8. Operation of the URL Handler

When the client-side URL handler gets the first URL for a given session, it would 'tune in that session' and (with luck) start receiving files and metainformation. On the receipt of 'special files' (e.g., an SDP) it can expand its knowledge of the session. Other files not corresponding to the immediate request in hand should be cached, observing the cache control headers. When the indicated file (or at least the requested byte-range of the indicated file) is available, it is returned. If a 404, 410, or 3xx response is received for the indicated file, then an appropriate error is returned, as indeed it is if the URL specifies that the multicast is only available over a given time range, and the request is not or cannot be satisfied in that time range.

9. Security Considerations

TBC.

10. IANA Considerations

This section contains the registration information for the "fcast" URI scheme (in accordance with Section 5.4 of [RFC4395]).

Editor's note: The registration template will be provided in a later revision.

11. References

11.1. Normative References

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

[RFC6968] Roca, V. and B. Adamson, "FCAST: Object Delivery for the Asynchronous Layered Coding (ALC) and NACK-Oriented Reliable Multicast (NORM) Protocols", RFC 6968, July 2013.

[RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 35, RFC 4395, February 2006.

11.2. Informative References

[RFC5775] Luby, M., Watson, M., and L. Vicisano, "Asynchronous Layered Coding (ALC) Protocol Instantiation", RFC 5775, April 2010.

[RFC5651] Luby, M., Watson, M., and L. Vicisano, "Layered Coding Transport (LCT) Building Block", RFC 5651, October 2009.

[RFC6726] Paila, T., Walsh, R., Luby, M., Roca, V., and R. Lehtonen, "FLUTE - File Delivery over Unidirectional Transport", RFC 6726, November 2012.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.

Authors' Addresses

Dave Singer
Apple, Inc.
1 Infinite Loop
Cupertino 95014
USA

EMail: singer@apple.com

Ali Begen
Cisco
181 Bay Street
Toronto, ON M5J 2T3
Canada

EMail: abegen@cisco.com