Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 29, 2012

J. Gregorio
Google
R. Fielding
Adobe
M. Hadley
MITRE
M. Nottingham
Rackspace
D. Orchard
Salesforce.com
Jan 26, 2012

URI Template
draft-gregorio-uritemplate-08

Abstract

A URI Template is a compact sequence of characters for describing a range of Uniform Resource Identifiers through variable expansion. This specification defines the URI Template syntax and the process for expanding a URI Template into a URI reference, along with guidelines for the use of URI Templates on the Internet.

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 29, 2012.

Table of Contents

1. Introduction

1.1. Overview

A Uniform Resource Identifier (URI) [RFC3986] is often used to identify a specific resource within a common space of similar resources (informally, a "URI space"). For example, personal web spaces are often delegated using a common pattern, such as

```
http://example.com/~fred/
http://example.com/~mark/
```

or a set of dictionary entries might be grouped in a hierarchy by the first letter of the term, as in

```
http://example.com/dictionary/c/cat
http://example.com/dictionary/d/dog
```

or a service interface might be invoked with various user input in a common pattern, as in

```
http://example.com/search?q=cat&lang=en
http://example.com/search?q=chien&lang=fr
```

A URI Template is a compact sequence of characters for describing a range of Uniform Resource Identifiers through variable expansion.

URI Templates provide a mechanism for abstracting a space of resource identifiers such that the variable parts can be easily identified and described. URI templates can have many uses, including discovery of available services, configuring resource mappings, defining computed links, specifying interfaces, and other forms of programmatic interaction with resources. For example, the above resources could be described by the following URI templates:

```
http://example.com/~{username}/
http://example.com/dictionary/{term:1}/{term}
http://example.com/search{?q,lang}
```

We define the following terms:

- expression - The text between '{' and '}', including the enclosing braces, as defined in Section 2.
- expansion - The string result obtained from a template expression after processing it according to its expression type, list of variable names, and value modifiers, as defined in Section 3.
- template processor - A program or library that, given a URI Template and a set of variables with values, transforms the template string into a URI-reference by parsing the template for

expressions and substituting each one with its corresponding expansion.

A URI Template provides both a structural description of a URI space and, when variable values are provided, machine-readable instructions on how to construct a URI corresponding to those values. A URI Template is transformed into a URI-reference by replacing each delimited expression with its value as defined by the expression type and the values of variables named within the expression. The expression types range from simple string expansion to multiple name=value lists. The expansions are based on the URI generic syntax, allowing an implementation to process any URI Template without knowing the scheme-specific requirements of every possible resulting URI.

For example, the following URI Template includes a form-style parameter expression, as indicated by the "?" operator appearing before the variable names.

```
http://www.example.com/foo{?query,number}
```

The expansion process for expressions beginning with the question-mark ("?") operator follows the same pattern as form-style interfaces on the World Wide Web:

```
http://www.example.com/foo{?query,number}
                           \_____________/
                                  |
                                  |
          For each defined variable in [ 'query', 'number' ],
          substitute "?" if it is the first substitution or "&"
          thereafter, followed by the variable name, '=', and the
          variable's value.
```

If the variables have the values

```
query  := "mycelium"
number := 100
```

then the expansion of the above URI Template is

```
http://www.example.com/foo?query=mycelium&number=100
```

Alternatively, if 'query' is undefined, then the expansion would be

```
http://www.example.com/foo?number=100
```

or if both variables are undefined, then it would be

http://www.example.com/foo

A URI Template may be provided in absolute form, as in the examples above, or in relative form. A template is expanded before the resulting reference is resolved from relative to absolute form.

Although the URI syntax is used for the result, the template string is allowed to contain the broader set of characters that can be found in IRI references [RFC3987]. A URI Template is therefore also an IRI template, and the result of template processing can be transformed to an IRI by following the process defined in Section 3.2 of [RFC3987].

1.2. Levels and Expression Types

URI Templates are similar to a macro language with a fixed set of macro definitions: the expression type determines the expansion process. The default expression type is simple string expansion, wherein a single named variable is replaced by its value as a string after pct-encoding any characters not in the set of unreserved URI characters (Section 1.5).

Since most template processors implemented prior to this specification have only implemented the default expression type, we refer to these as Level 1 templates.

Level 1 examples, with variables having values of

```
var   := "value"
hello := "Hello World!"
```

Op	Expression	Expansion	
	Simple string expansion		(Sec 3.2.2)
	{var}	value	
	{hello}	Hello%20World%21	

Level 2 templates add the plus ("+") operator, for expansion of values that are allowed to include reserved URI characters (Section 1.5), and the crosshatch ("#") operator for expansion of fragment identifiers.

Level 2 examples, with variables having values of			
	var := "value"		
	hello := "Hello World!"		
	path := "/foo/bar"		
Op	Expression	Expansion	
+	Reserved string expansion		(Sec 3.2.3)
	{+var}	value	
	{+hello}	Hello%20World!	
	{+path}/here	/foo/bar/here	
	here?ref={+path}	here?ref=/foo/bar	
#	Fragment expansion, crosshatch-prefixed		(Sec 3.2.4)
	X{#var}	X#value	
	X{#hello}	X#Hello%20World!	

Level 3 templates allow multiple variables per expression, each separated by a comma, and add more complex operators for dot-prefixed labels, slash-prefixed path segments, semicolon-prefixed path parameters, and the forms-style construction of a query syntax consisting of name=value pairs that are separated by an ampersand character.

Level 3 examples, with variables having values of			
	var := "value"		
	hello := "Hello World!"		
	empty := ""		
	path := "/foo/bar"		
	x := "1024"		
	y := "768"		
Op	Expression	Expansion	
	String expansion with multiple variables		(Sec 3.2.2)
	map?{x,y}	map?1024,768	
	{x,hello,y}	1024,Hello%20World%21,768	

+	Reserved expansion with multiple variables	(Sec 3.2.3)
	{+x,hello,y}	1024,Hello%20World!,768
	{+path,x}/here	/foo/bar,1024/here
#	Fragment expansion with multiple variables	(Sec 3.2.4)
	{#x,hello,y}	#1024,Hello%20World!,768
	{#path,x}/here	#/foo/bar,1024/here
.	Label expansion, dot-prefixed	(Sec 3.2.5)
	X{.var}	X.value
	X{.x,y}	X.1024.768
/	Path segments, slash-prefixed	(Sec 3.2.6)
	{/var}	/value
	{/var,x}/here	/value/1024/here
;	Path-style parameters, semicolon-prefixed	(Sec 3.2.7)
	{;x,y}	;x=1024;y=768
	{;x,y,empty}	;x=1024;y=768;empty
?	Form-style query, ampersand-separated	(Sec 3.2.8)
	{?x,y}	?x=1024&y=768
	{?x,y,empty}	?x=1024&y=768&empty=
&	Form-style query continuation	(Sec 3.2.9)
	?fixed=yes{&x}	?fixed=yes&x=1024
	{&x,y,empty}	&x=1024&y=768&empty=

Finally, Level 4 templates add value modifiers as an optional suffix to each variable name. A prefix modifier (":") indicates that only a limited number of characters from the beginning of the value are used by the expansion (Section 2.4.1). An explode ("*") modifier

indicates that the variable is to be treated as a composite value, consisting of either a list of names or an associative array of (name, value) pairs, that is expanded as if each member were a separate variable (Section 2.4.2).

Level 4 examples, with variables having values of

```
var   := "value"
hello := "Hello World!"
path  := "/foo/bar"
list  := ("red", "green", "blue")
keys  := [("semi",";"),("dot","."),("comma",",")]
```

Op	Expression	Expansion	
	String expansion with value modifiers		(Sec 3.2.2)
	{var:3}	val	
	{var:30}	value	
	{list}	red,green,blue	
	{list*}	red,green,blue	
	{keys}	semi,%3B,dot,.,comma,%2C	
	{keys*}	semi=%3B,dot=.,comma=%2C	
+	Reserved expansion with value modifiers		(Sec 3.2.3)
	{+path:6}/here	/foo/b/here	
	{+list}	red,green,blue	
	{+list*}	red,green,blue	
	{+keys}	semi,;,dot,.,comma,,	
	{+keys*}	semi=;,dot=.,comma=,	
#	Fragment expansion with value modifiers		(Sec 3.2.4)
	{#path:6}/here	#/foo/b/here	
	{#list}	#red,green,blue	
	{#list*}	#red,green,blue	
	{#keys}	#semi,;,dot,.,comma,,	
	{#keys*}	#semi=;,dot=.,comma=,	
.	Label expansion, dot-prefixed		(Sec 3.2.5)
	X{.var:3}	X.val	
	X{.list}	X.red,green,blue	

		X{.list*}	X.red.green.blue
		X{.keys}	X.semi,%3B,dot,.,comma,%2C
		X{.keys*}	X.semi=%3B.dot=..comma=%2C
/	Path segments, slash-prefixed		(Sec 3.2.6)
		{/var:1,var}	/v/value
		{/list}	/red,green,blue
		{/list*}	/red/green/blue
		{/list*,path:4}	/red/green/blue/%2Ffoo
		{/keys}	/semi,%3B,dot,.,comma,%2C
		{/keys*}	/semi=%3B/dot=./comma=%2C
;	Path-style parameters, semicolon-prefixed		(Sec 3.2.7)
		{;hello:5}	;hello=Hello
		{;list}	;list=red,green,blue
		{;list*}	;list=red;list=green;list=blue
		{;keys}	;keys=semi,%3B,dot,.,comma,%2C
		{;keys*}	;semi=%3B;dot=.;comma=%2C
?	Form-style query, ampersand-separated		(Sec 3.2.8)
		{?var:3}	?var=val
		{?list}	?list=red,green,blue
		{?list*}	?list=red&list=green&list=blue
		{?keys}	?keys=semi,%3B,dot,.,comma,%2C
		{?keys*}	?semi=%3B&dot=.&comma=%2C
&	Form-style query continuation		(Sec 3.2.9)
		{&var:3}	&var=val
		{&list}	&list=red,green,blue
		{&list*}	&list=red&list=green&list=blue
		{&keys}	&keys=semi,%3B,dot,.,comma,%2C
		{&keys*}	&semi=%3B&dot=.&comma=%2C

1.3. Design Considerations

Mechanisms similar to URI Templates have been defined within several specifications, including WSDL [WSDL], WADL [WADL] and OpenSearch [OpenSearch]. This specification extends and formally defines the

syntax so that URI Templates can be used consistently across multiple Internet applications and within Internet message fields, while at the same time retaining compatibility with those earlier definitions.

The URI Template syntax has been designed to carefully balance the need for a powerful expansion mechanism with the need for ease of implementation. The syntax is designed to be trivial to parse while at the same time providing enough flexibility to express many common template scenarios. Implementations are able to parse the template and perform the expansions in a single pass.

Templates are simple and readable when used with common examples because the single-character operators match the URI generic syntax delimiters. The operator's associated delimiter (".", ";", "/", "?", "&", and "#") is omitted when none of the listed variables are defined. Likewise, the expansion process for ";" (path-style parameters) will omit the "=" when the variable value is empty, whereas the process for "?" (form-style parameters) will not omit the "=" when the value is empty. Multiple variables and list values have their values joined with "," if there is no predefined joining mechanism for the operator. The "+" and "#" operators will substitute unencoded reserved characters found inside the variable values; the other operators will pct-encode reserved characters found in the variable values prior to expansion.

The most common cases for URI spaces can be described with Level 1 template expressions. If we were only concerned with URI generation, then the template syntax could be limited to just simple variable expansion, since more complex forms could be generated by changing the variable values. However, URI Templates have the additional goal of describing the layout of identifiers in terms of preexisting data values. The template syntax therefore includes operators that reflect how resource identifiers are commonly allocated. Likewise, since prefix substrings are often used to partition large spaces of resources, modifiers on variable values provide a way to specify both the substring and the full value string with a single variable name.

1.4. Limitations

Since a URI Template describes a superset of the identifiers, there is no implication that every possible expansion for each delimited variable expression corresponds to a URI of an existing resource. Our expectation is that an application constructing URIs according to the template will be provided with an appropriate set of values for the variables being substituted, or at least a means of validating user data-entry for those values.

URI Templates are not URIs: they do not identify an abstract or

physical resource, they are not parsed as URIs, and should not be used in places where a URI would be expected unless the template expressions will be expanded by a template processor prior to use. Distinct field, element, or attribute names should be used to differentiate protocol elements that carry a URI Template from those that expect a URI reference.

Some URI Templates can be used in reverse for the purpose of variable matching: comparing the template to a fully formed URI in order to extract the variable parts from that URI and assign them to the named variables. Variable matching only works well if the template expressions are delimited by the beginning or end of the URI or by characters that cannot be part of the expansion, such as reserved characters surrounding a simple string expression. In general, regular expression languages are better suited for variable matching.

1.5. Notational Conventions

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].

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234]. The following ABNF rules are imported from the normative references [RFC5234], [RFC3986], and [RFC3987].

```
ALPHA          =  %x41-5A / %x61-7A   ; A-Z / a-z
DIGIT          =  %x30-39             ; 0-9
HEXDIG         =  DIGIT / "A" / "B" / "C" / "D" / "E" / "F"
                  ; case-insensitive

pct-encoded    =  "%" HEXDIG HEXDIG
unreserved     =  ALPHA / DIGIT / "-" / "." / "_" / "~"
reserved       =  gen-delims / sub-delims
gen-delims     =  ":" / "/" / "?" / "#" / "[" / "]" / "@"
sub-delims     =  "!" / "$" / "&" / "'" / "(" / ")"
               /  "*" / "+" / "," / ";" / "="

ucschar        =  %xA0-D7FF / %xF900-FDCF / %xFDF0-FFEF
               /  %x10000-1FFFD / %x20000-2FFFD / %x30000-3FFFD
               /  %x40000-4FFFD / %x50000-5FFFD / %x60000-6FFFD
               /  %x70000-7FFFD / %x80000-8FFFD / %x90000-9FFFD
               /  %xA0000-AFFFD / %xB0000-BFFFD / %xC0000-CFFFD
               /  %xD0000-DFFFD / %xE1000-EFFFD

iprivate       =  %xE000-F8FF / %xF0000-FFFFD / %x100000-10FFFD
```

1.6. Character Encoding and Unicode Normalization

This specification uses the terms "character", "character encoding scheme", "code point", "coded character set", "glyph", "non-ASCII", "normalization", "protocol element", and "regular expression" as they are defined in [RFC6365].

The ABNF notation defines its terminal values to be non-negative integers (code points) that are a superset of the US-ASCII coded character set [ASCII]. This specification defines terminal values as code points within the Unicode coded character set [UNIV6].

In spite of the syntax and template expansion process being defined in terms of Unicode code points, it should be understood that templates occur in practice as a sequence of characters in whatever form or encoding is suitable for the context in which they occur, whether that be octets embedded in a network protocol element or glyphs painted on the side of a bus. This specification does not mandate any particular character encoding scheme for mapping between URI Template characters and the octets used to store or transmit those characters. When a URI Template appears in a protocol element, the character encoding scheme is defined by that protocol; without such a definition, a URI Template is assumed to be in the same character encoding scheme as the surrounding text. It is only during the process of template expansion that a string of characters in a URI Template is REQUIRED to be processed as a sequence of Unicode code points.

The Unicode Standard [UNIV6] defines various equivalences between sequences of characters for various purposes. Unicode Standard Annex #15 [UTR15] defines various Normalization Forms for these equivalences. The normalization form determines how to consistently encode equivalent strings. In theory, all URI processing implementations, including template processors, should use the same normalization form for generating a URI reference. In practice, they do not. If a value has been provided by the same server as the resource, then it can be assumed that the string is already in the form expected by that server. If a value is provided by a user, such as via a data-entry dialog, then the string SHOULD be normalized as Normalization Form C (NFC: Canonical Decomposition, followed by Canonical Composition) prior to being used in expansions by a template processor.

Likewise, when non-ASCII data that represents readable strings is pct-encoded for use in a URI reference, a template processor MUST first encode the string as UTF-8 [RFC3629] and then pct-encode any octets that are not allowed in a URI reference.

2. Syntax

A URI Template is a string of printable Unicode characters that contains zero or more embedded variable expressions, each expression being delimited by a matching pair of braces ('{', '}').

```
URI-Template  = *( literals / expression )
```

Although templates (and template processor implementations) are described above in terms of four gradual levels, we define the URI-Template syntax in terms of the ABNF for Level 4. A template processor limited to lower level templates MAY exclude the ABNF rules applicable only to higher levels. However, it is RECOMMENDED that all parsers implement the full syntax such that unsupported levels can be properly identified as such to the end user.

2.1. Literals

The characters outside of expressions in a URI Template string are intended to be copied literally to the URI-reference if the character is allowed in a URI (reserved / unreserved / pct-encoded) or, if not allowed, copied to the URI-reference as the sequence of pct-encoded triplets corresponding to that character's encoding in UTF-8 [RFC3629].

```
literals      =  %x21 / %x23-24 / %x26 / %x28-3B / %x3D / %x3F-5B
              /  %x5D / %x5F / %x61-7A / %x7E / ucschar / iprivate
              /  pct-encoded
                   ; any Unicode character except: CTL, SP,
                   ;  DQUOTE, "'", "%" (aside from pct-encoded),
                   ;  "<", ">", "\", "^", "`", "{", "|", "}"
```

2.2. Expressions

Template expressions are the parameterized parts of a URI Template. Each expression contains an optional operator, which defines the expression type and its corresponding expansion process, followed by a comma-separated list of variable specifiers (variable names and optional value modifiers). If no operator is provided, the expression defaults to simple variable expansion of unreserved values.

```
expression    =  "{" [ operator ] variable-list "}"
operator      =  op-level2 / op-level3 / op-reserve
op-level2     =  "+" / "#"
op-level3     =  "." / "/" / ";" / "?" / "&"
op-reserve    =  "=" / "," / "!" / "@" / "|"
```

The operator characters have been chosen to reflect each of their roles as reserved characters in the URI generic syntax. The operators defined in Section 3 of this specification include:

+ Reserved character strings;

Fragment identifiers prefixed by "#";

. Name labels or extensions prefixed by ".";

/ Path segments prefixed by "/";

; Path parameter name or name=value pairs prefixed by ";";

? Query component beginning with "?" and consisting of name=value pairs separated by "&"; and,

& Continuation of query-style &name=value pairs within a literal query component.

The operator characters equals ("="), comma (","), exclamation ("!"), at-sign ("@"), and pipe ("|") are reserved for future extensions.

The expression syntax specifically excludes use of the dollar ("$") and parentheses ["(" and ")"] characters so that they remain available for use outside the scope of this specification. For example, a macro language might use these characters to apply macro substitution to a string prior to that string being processed as a URI Template.

2.3. Variables

After the operator (if any), each expression contains a list of one or more comma-separated variable specifiers (varspec). The variable names serve multiple purposes: documentation for what kinds of values are expected, identifiers for associating values within a template processor, and the literal string to use for the name in name=value expansions (aside from when exploding an associative array). Variable names are case-sensitive because the name might be expanded within a case-sensitive URI component.

```
variable-list =  varspec *( "," varspec )
varspec       =  varname [ modifier-level4 ]
varname       =  varchar *( ["."] varchar )
varchar       =  ALPHA / DIGIT / "_" / pct-encoded
```

A varname MAY contain one or more pct-encoded triplets. These triplets are considered an essential part of the variable name and

are not decoded during processing. A varname containing pct-encoded characters is not the same variable as a varname with those same characters decoded. Applications that provide URI Templates are expected to be consistent in their use of pct-encoding within variable names.

An expression MAY reference variables that are unknown to the template processor or whose value is set to a special "undefined" value, such as undef or null. Such undefined variables are given special treatment by the expansion process (Section 3.2.1).

A variable value that is a string of length zero is not considered undefined; it has the defined value of an empty string.

In Level 4 templates, a variable may have a composite value in the form of a list of values or an associative array of (name, value) pairs. Such value types are not directly indicated by the template syntax, but do have an impact on the expansion process (Section 3.2.1).

A variable defined as a list value is considered undefined if the list contains zero members. A variable defined as an associative array of (name, value) pairs is considered undefined if the array contains zero members or if all member names in the array are associated with undefined values.

2.4. Value Modifiers

Each of the variables in a Level 4 template expression can have a modifier indicating either that its expansion is limited to a prefix of the variable's value string or that its expansion is exploded as a composite value in the form of a value list or an associative array of (name, value) pairs.

```
modifier-level4 =  prefix / explode
```

2.4.1. Prefix Values

A prefix modifier indicates that the variable expansion is limited to a prefix of the variable's value string. Prefix modifiers are often used to partition an identifier space hierarchically, as is common in reference indices and hash-based storage. It also serves to limit the expanded value to a maximum number of characters. Prefix modifiers are not applicable to variables that have composite values.

```
prefix        =  ":" max-length
max-length    =  %x31-39 0*3DIGIT   ; positive integer < 10000
```

The max-length is a positive integer that refers to a maximum number of characters from the beginning of the variable's value as a Unicode string. Note that this numbering is in characters, not octets, in order to avoid splitting between the octets of a multi-octet encoded character or within a pct-encoded triplet. If the max-length is greater than the length of the variable's value, then the entire value string is used.

For example,

Given the variable assignments

```
var   := "value"
semi  := ";"
```

Example Template	Expansion
{var}	value
{var:20}	value
{var:3}	val
{semi}	%3B
{semi:2}	%3B

2.4.2. Composite Values

An explode ("*") modifier indicates that the variable is to be treated as a composite value consisting of either a list of values or an associative array of (name, value) pairs. Hence, the expansion process is applied to each member of the composite as if it were listed as a separate variable. This kind of variable specification is significantly less self-documenting than non-exploded variables, since there is less correspondence between the variable name and how the URI reference appears after expansion.

```
explode       =  "*"
```

Since URI Templates do not contain an indication of type or schema, the type for an exploded variable is assumed to be determined by context. For example, the processor might be supplied values in a form that differentiates values as strings, lists, or associative arrays. Likewise, the context in which the template is used (script, mark-up language, IDL, etc.) might define rules for associating variable names with types, structures, or schema.

Explode modifiers improve brevity in the URI Template syntax. For example, a resource that provides a geographic map for a given street address might accept a hundred permutations on fields for address input, including partial addresses (e.g., just the city or postal

code). Such a resource could be described as a template with each and every address component listed in order, or with a far more simple template that makes use of an explode modifier, as in

```
/mapper{?address*}
```

along with some context that defines what the variable named "address" can include, such as by reference to some other standard for addressing (e.g., [UPU-S42]). A recipient aware of the schema can then provide appropriate expansions, such as:

```
/mapper?city=Newport%20Beach&state=CA
```

The expansion process for exploded variables is dependent on both the operator being used and whether the composite value is to be treated as a list of values or as an associative array of (name, value) pairs. Structures are processed as if they are an associative array with names corresponding to the fields in the structure definition and "." separators used to indicate name hierarchy in substructures.

If a variable has a composite structure and only some of the fields in that structure have defined values, then only the defined pairs are present in the expansion. This can be useful for templates that consist of a large number of potential query terms.

An explode modifier applied to a list variable causes the expansion to iterate over the list's member values. For path and query parameter expansions, each member value is paired with the variable's name as a (varname, value) pair. This allows path and query parameters to be repeated for multiple values, as in

Given the variable assignments

```
year  := ("1965", "2000", "2012")
dom   := ("example", "com")
```

Example Template	Expansion
find{?year*}	find?year=1965&year=2000&year=2012
www{.dom*}	www.example.com

3. Expansion

The process of URI Template expansion is to scan the template string from beginning to end, copying literal characters and replacing each expression with the result of applying the expression's operator to the value of each variable named in the expression. Each variable's

value MUST be formed prior to template expansion.

The requirements on expansion for each aspect of the URI Template grammar are defined in this section. A non-normative algorithm for the expansion process as a whole is provided in Appendix A.

If a template processor encounters a character sequence outside an expression that does not match the <URI-Template> grammar, then processing of the template SHOULD cease, the URI-reference result SHOULD contain the expanded part of the template followed by the remainder unexpanded, and the location and type of error SHOULD be indicated to the invoking application.

If an error is encountered in an expression, such as an operator or value modifier that the template processor does not recognize or does not yet support, or a character is found that is not allowed by the <expression> grammar, then the unprocessed parts of the expression SHOULD be copied to the result unexpanded, processing of the remainder of the template SHOULD continue, and the location and type of error SHOULD be indicated to the invoking application.

If an error occurs, the result returned might not be a valid URI reference; it will be an incompletely expanded template string that is only intended for diagnostic use.

3.1. Literal Expansion

If the literal character is allowed anywhere in the URI syntax (unreserved / reserved / pct-encoded), then it is copied directly to the result string. Otherwise, the pct-encoded equivalent of the literal character is copied to the result string by first encoding the character as its sequence of octets in UTF-8 and then encoding each such octet as a pct-encoded triplet.

3.2. Expression Expansion

Each expression is indicated by an opening brace ("{") character and continues until the next closing brace ("}"). Expressions cannot be nested.

An expression is expanded by determining its expression type and then following that type's expansion process for each comma-separated varspec in the expression. Level 1 templates are limited to the default operator (simple string value expansion) and a single variable per expression. Level 2 templates are limited to a single varspec per expression.

The expression type is determined by looking at the first character

after the opening brace. If the character is an operator, then remember the expression type associated with that operator for later expansion decisions and skip to the next character for the variable-list. If the first character is not an operator, then the expression type is simple string expansion and the first character is the beginning of the variable-list.

The examples in the subsections below use the following definitions for variable values:

```
count := ("one", "two", "three")
dom   := ("example", "com")
dub   := "me/too"
hello := "Hello World!"
half  := "50%"
var   := "value"
who   := "fred"
base  := "http://example.com/home/"
path  := "/foo/bar"
list  := ("red", "green", "blue")
keys  := [("semi",";"),("dot","."),("comma",",")]
v     := "6"
x     := "1024"
y     := "768"
empty := ""
empty_keys  := []
undef := null
```

3.2.1. Variable Expansion

A variable that is undefined (Section 2.3) has no value and is ignored by the expansion process. If all of the variables in an expression are undefined, then the expression's expansion is the empty string.

Variable expansion of a defined, non-empty value results in a substring of allowed URI characters. As described in Section 1.6, the expansion process is defined in terms of Unicode code points in order to ensure that non-ASCII characters are consistently pct-encoded in the resulting URI reference. One way for a template processor to obtain a consistent expansion is to transcode the value string to UTF-8 (if it is not already in UTF-8) and then transform each octet that is not in the allowed set into the corresponding pct-encoded triplet. Another is to map directly from the value's native character encoding to the set of allowed URI characters, with any remaining disallowed characters mapping to the sequence of pct-encoded triplets that correspond to the octet(s) of that character

when encoded as UTF-8 [RFC3629].

The allowed set for a given expansion depends on the expression type: reserved ("+") and fragment ("#") expansions allow the set of characters in the union of (unreserved / reserved / pct-encoded) to be passed through without pct-encoding, whereas all other expression types allow only unreserved characters to be passed through without pct-encoding. Note that the percent character ("%") is only allowed as part of a pct-encoded triplet and only for reserved/fragment expansion: in all other cases, a value character of "%" MUST be pct-encoded as "%25" by variable expansion.

If a variable appears more than once in an expression or within multiple expressions of a URI Template, the value of that variable MUST remain static throughout the expansion process (i.e., the variable must have the same value for the purpose of calculating each expansion). However, if reserved characters or pct-encoded triplets occur in the value, they will be pct-encoded by some expression types and not by others.

For a variable that is a simple string value, expansion consists of appending the encoded value to the result string. An explode modifier has no effect. A prefix modifier limits the expansion to the first max-length characters of the decoded value. If the value contains multi-octet or pct-encoded characters, care must be taken to avoid splitting the value in mid-character: count each Unicode code point as one character.

For a variable that is an associative array, expansion depends on both the expression type and the presence of an explode modifier. If there is no explode modifier, expansion consists of appending a comma-separated concatenation of each (name, value) pair that has a defined value. If there is an explode modifier, expansion consists of appending each pair that has a defined value as either "name=value" or, if the value is the empty string and the expression type does not indicate form-style parameters (i.e., not a "?" or "&" type), simply "name". Both name and value strings are encoded in the same way as simple string values. A separator string is appended between defined pairs according to the expression type, as defined by the following table:

Type	Separator	
	","	(default)
+	","	
#	","	
.	"."	
/	"/"	
;	";"	
?	"&"	
&	"&"	

For a variable that is a list of values, expansion depends on both the expression type and the presence of an explode modifier. If there is no explode modifier, the expansion consists of a comma-separated concatenation of the defined member string values. If there is an explode modifier and the expression type expands named parameters (";", "?", or "&"), then the list is expanded as if it were an associative array in which each member value is paired with the list's varname. Otherwise, the value will be expanded as if it were a list of separate variable values, each value separated by the expression type's associated separator as defined by the table above.

Example Template	Expansion
{count}	one,two,three
{count*}	one,two,three
{/count}	/one,two,three
{/count*}	/one/two/three
{;count}	;count=one,two,three
{;count*}	;count=one;count=two;count=three
{?count}	?count=one,two,three
{?count*}	?count=one&count=two&count=three
{&count*}	&count=one&count=two&count=three

3.2.2. Simple String Expansion: {var}

Simple string expansion is the default expression type when no operator is given.

For each defined variable in the variable-list, perform variable expansion, as defined in Section 3.2.1, with the allowed characters being those in the unreserved set. If more than one variable has a defined value, append a comma (",") to the result string as a separator between variable expansions.

Example Template	Expansion
{var}	value
{hello}	Hello%20World%21
{half}	50%25
O{empty}X	OX
O{undef}X	OX
{x,y}	1024,768
{x,hello,y}	1024,Hello%20World%21,768
?{x,empty}	?1024,
?{x,undef}	?1024
?{undef,y}	?768
{var:3}	val
{var:30}	value
{list}	red,green,blue
{list*}	red,green,blue
{keys}	semi,%3B,dot,.,comma,%2C
{keys*}	semi=%3B,dot=.,comma=%2C

3.2.3. Reserved expansion: {+var}

Reserved expansion, as indicated by the plus ("+") operator for Level 2 and above templates, is identical to simple string expansion except that the substituted values may also contain pct-encoded triplets and characters in the reserved set.

For each defined variable in the variable-list, perform variable expansion, as defined in Section 3.2.1, with the allowed characters being those in the set (unreserved / reserved / pct-encoded). If more than one variable has a defined value, append a comma (",") to the result string as a separator between variable expansions.

Example Template	Expansion
{+var}	value
{+hello}	Hello%20World!
{+half}	50%25
{base}index	http%3A%2F%2Fexample.com%2Fhome%2Findex
{+base}index	http://example.com/home/index
O{+empty}X	OX
O{+undef}X	OX
{+path}/here	/foo/bar/here
here?ref={+path}	here?ref=/foo/bar
up{+path}{var}/here	up/foo/barvalue/here
{+x,hello,y}	1024,Hello%20World!,768
{+path,x}/here	/foo/bar,1024/here
{+path:6}/here	/foo/b/here
{+list}	red,green,blue
{+list*}	red,green,blue
{+keys}	semi,;,dot,.,comma,,
{+keys*}	semi=;,dot=.,comma=,

3.2.4. Fragment expansion: {#var}

Fragment expansion, as indicated by the crosshatch ("#") operator for Level 2 and above templates, is identical to reserved expansion except that a crosshatch character (fragment delimiter) is appended first to the result string if any of the variables are defined.

Example Template	Expansion
{#var}	#value
{#hello}	#Hello%20World!
{#half}	#50%25
foo{#empty}	foo#
foo{#undef}	foo
{#x,hello,y}	#1024,Hello%20World!,768
{#path,x}/here	#/foo/bar,1024/here
{#path:6}/here	#/foo/b/here
{#list}	#red,green,blue
{#list*}	#red,green,blue
{#keys}	#semi,;,dot,.,comma,,
{#keys*}	#semi=;,dot=.,comma=,

3.2.5. Label expansion with dot-prefix: {.var}

Label expansion, as indicated by the dot (".") operator for Level 3 and above templates, is useful for describing URI spaces with varying domain names or path selectors (e.g., filename extensions).

For each defined variable in the variable-list, append "." to the result string and then perform variable expansion, as defined in Section 3.2.1, with the allowed characters being those in the unreserved set.

Since "." is in the unreserved set, a value that contains a "." has the effect of adding multiple labels.

Example Template	Expansion
{.who}	.fred
{.who,who}	.fred.fred
{.half,who}	.50%25.fred
www{.dom*}	www.example.com
X{.var}	X.value
X{.empty}	X.
X{.undef}	X
X{.var:3}	X.val
X{.list}	X.red,green,blue
X{.list*}	X.red.green.blue
X{.keys}	X.semi,%3B,dot,.,comma,%2C
X{.keys*}	X.semi=%3B.dot=..comma=%2C
X{.empty_keys}	X
X{.empty_keys*}	X

3.2.6. Path segment expansion: {/var}

Path segment expansion, as indicated by the slash ("/") operator in Level 3 and above templates, is useful for describing URI path hierarchies.

For each defined variable in the variable-list, append "/" to the result string and then perform variable expansion, as defined in Section 3.2.1, with the allowed characters being those in the unreserved set.

Note that the expansion process for path segment expansion is identical to that of label expansion aside from the substitution of "/" instead of ".". However, unlike ".", a "/" is a reserved character and will be pct-encoded if found in a value.

Example Template	Expansion
{/who}	/fred
{/who,who}	/fred/fred
{/half,who}	/50%25/fred
{/who,dub}	/fred/me%2Ftoo
{/var}	/value
{/var,empty}	/value/
{/var,undef}	/value
{/var,x}/here	/value/1024/here
{/var:1,var}	/v/value
{/list}	/red,green,blue
{/list*}	/red/green/blue
{/list*,path:4}	/red/green/blue/%2Ffoo
{/keys}	/semi,%3B,dot,.,comma,%2C
{/keys*}	/semi=%3B/dot=./comma=%2C

3.2.7. Path-style parameter expansion: {;var}

Path-style parameter expansion, as indicated by the semicolon (";") operator in Level 3 and above templates, is useful for describing URI path parameters, such as "path;property" or "path;name=value".

For each defined variable in the variable-list:
- append ";" to the result string;
- if the variable has a simple string value or no explode modifier is given, then:
  - append the variable name (encoded as if it were a literal string) to the result string;
  - if the variable's value is not empty, append "=" to the result string;
- perform variable expansion, as defined in Section 3.2.1, with the allowed characters being those in the unreserved set.

Example Template	Expansion
{;who}	;who=fred
{;half}	;half=50%25
{;empty}	;empty
{;v,empty,who}	;v=6;empty;who=fred
{;v,bar,who}	;v=6;who=fred
{;x,y}	;x=1024;y=768
{;x,y,empty}	;x=1024;y=768;empty
{;x,y,undef}	;x=1024;y=768
{;hello:5}	;hello=Hello
{;list}	;list=red,green,blue
{;list*}	;list=red;list=green;list=blue
{;keys}	;keys=semi,%3B,dot,.,comma,%2C

{;keys*}	;semi=%3B;dot=.;comma=%2C

3.2.8. Form-style query expansion: {?var}

Form-style query expansion, as indicated by the question-mark ("?") operator in Level 3 and above templates, is useful for describing an entire optional query component.

For each defined variable in the variable-list:
- append "?" to the result string if this is the first defined value or append "&" thereafter;
- if the variable has a simple string value or no explode modifier is given, append the variable name (encoded as if it were a literal string) and an equals character ("=") to the result string; and,
- perform variable expansion, as defined in Section 3.2.1, with the allowed characters being those in the unreserved set.

Example Template	Expansion
{?who}	?who=fred
{?half}	?half=50%25
{?x,y}	?x=1024&y=768
{?x,y,empty}	?x=1024&y=768&empty=
{?x,y,undef}	?x=1024&y=768
{?var:3}	?var=val
{?list}	?list=red,green,blue
{?list*}	?list=red&list=green&list=blue
{?keys}	?keys=semi,%3B,dot,.,comma,%2C
{?keys*}	?semi=%3B&dot=.&comma=%2C

3.2.9. Form-style query continuation: {&var}

Form-style query continuation, as indicated by the ampersand ("&") operator in Level 3 and above templates, is useful for describing optional &name=value pairs in a template that already contains a literal query component with fixed parameters.

For each defined variable in the variable-list:
- append "&" to the result string;
- if the variable has a simple string value or no explode modifier is given, append the variable name (encoded as if it were a literal string) and an equals character ("=") to the result string; and,
- perform variable expansion, as defined in Section 3.2.1, with the allowed characters being those in the unreserved set.

Example Template	Expansion
{&who}	&who=fred
{&half}	&half=50%25
?fixed=yes{&x}	?fixed=yes&x=1024
{&x,y,empty}	&x=1024&y=768&empty=
{&x,y,undef}	&x=1024&y=768
{&var:3}	&var=val
{&list}	&list=red,green,blue
{&list*}	&list=red&list=green&list=blue
{&keys}	&keys=semi,%3B,dot,.,comma,%2C
{&keys*}	&semi=%3B&dot=.&comma=%2C

4. Security Considerations

A URI Template does not contain active or executable content. However, it might be possible to craft unanticipated URIs if an attacker is given control over the template or over the variable values within an expression that allows reserved characters in the expansion. In either case, the security considerations are largely determined by who provides the template, who provides the values to use for variables within the template, in what execution context the expansion occurs (client or server), and where the resulting URIs are used.

This specification does not limit where URI Templates might be used. Current implementations exist within server-side development frameworks and within client-side javascript for computed links or forms.

Within frameworks, templates usually act as guides for where data might occur within later (request-time) URIs in client requests. Hence, the security concerns are not in the templates themselves, but rather in how the server extracts and processes the user-provided data within a normal Web request.

Within client-side implementations, a URI template has many of the same properties as HTML forms, except limited to URI characters and possibly included in HTTP header field values instead of just message body content. Care ought to be taken to ensure that potentially dangerous URI reference strings, such as those beginning with "javascript:", do not appear in the expansion unless both the template and the values are provided by a trusted source.

Other security considerations are the same as those for URIs, as described in section 7 of [RFC3986].

5. IANA Considerations

No IANA actions are required by this document.

6. Acknowledgments

The following people made contributions to this specification: Mike Burrows, Michaeljohn Clement, DeWitt Clinton, John Cowan, Stephen Farrell, Robbie Gates, Vijay K. Gurbani, Peter Johanson, Murray S. Kucherawy, James H. Manger, Tom Petch, Marc Portier, Pete Resnick, James Snell, and Jiankang Yao.

7. References

7.1. Normative References

[ASCII] American National Standards Institute, "Coded Character Set - 7-bit American Standard Code for Information Interchange", ANSI X3.4, 1986.

[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.

[RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, January 2005.

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

[RFC6365] Hoffman, P. and J. Klensin, "Terminology Used in Internationalization in the IETF", BCP 166, RFC 6365, September 2011.

[UNIV6] The Unicode Consortium, "The Unicode Standard, Version 6.0.0", ISBN 978-1-936213-01-6, October 2011, <http://www.unicode.org/versions/Unicode6.0.0/>.

[UTR15] Davis, M. and M. Duerst, "Unicode Normalization Forms", Unicode Standard Annex # 15, April 2003,

<http://www.unicode.org/unicode/reports/tr15/
tr15-23.html>.

7.2. Informative References

[OpenSearch]
Clinton, D., "OpenSearch 1.1", Draft 5, December 2011,
<http://www.opensearch.org/Specifications/OpenSearch>.

[UPU-S42] Universal Postal Union, "International Postal Address Components and Templates", UPU S42-1, November 2002, <http ://www.upu.int/en/activities/addressing/standards.html>.

[WADL] Hadley, M., "Web Application Description Language", World Wide Web Consortium Member Submission SUBM-wadl-20090831, August 2009,
<http://www.w3.org/Submission/2009/SUBM-wadl-20090831/>.

[WSDL] Weerawarana, S., Moreau, J., Ryman, A., and R. Chinnici, "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language", World Wide Web Consortium Recommendation REC-wsdl20-20070626, June 2007,
<http://www.w3.org/TR/2007/REC-wsdl20-20070626>.

Appendix A. Implementation Hints

The normative sections on expansion describe each operator with a separate expansion process for the sake of descriptive clarity. In actual implementations, we expect the expressions to be processed left-to-right using a common algorithm that has only minor variations in process per operator. This non-normative appendix describes one such algorithm.

Initialize an empty result string and its non-error state.

Scan the template and copy literals to the result string (as in Section 3.1) until an expression is indicated by a "{", an error is indicated by the presence of a non-literals character other than "{", or the template ends. When it ends, return the result string and its current error or non-error state.

- If an expression is found, scan the template to the next "}" and extract the characters in between the braces.
- If the template ends before a "}", then append the "{" and extracted characters to the result string and return with an error status indicating the expression is malformed.

Examine the first character of the extracted expression for an

operator.

- If the expression ended (i.e., is "{}"), an operator is found that is unknown or unimplemented, or the character is not in the varchar set (Section 2.3), then append "{", the extracted expression, and "}" to the result string, remember that the result is in an error state, and then go back to scan the remainder of the template.
- If a known and implemented operator is found, store the operator and skip to the next character to begin the varspec-list.
- Otherwise, store the operator as NUL (simple string expansion).

Use the following value table to determine the processing behavior by expression type operator. The entry for "first" is the string to append to the result first if any of the expression's variables are defined. The entry for "sep" is the separator to append to the result before any second (or subsequent) defined variable expansion. The entry for "named" is a boolean for whether or not the expansion includes the variable or key name when no explode modifier is given. The entry for "ifemp" is a string to append to the name if its corresponding value is empty. The entry for "allow" indicates what characters to allow unencoded within the value expansion: (U) means any character not in the unreserved set will be encoded; (U+R) means any character not in the union of (unreserved / reserved / pct-encoding) will be encoded; and, for both cases, each disallowed character is first encoded as its sequence of octets in UTF-8 and then each such octet is encoded as a pct-encoded triplet.

	NUL	+	.	/	;	?	&	#
first	""	""	"."	"/"	";"	"?"	"&"	"#"
sep	","	","	"."	"/"	";"	"&"	"&"	","
named	false	false	false	false	true	true	true	false
ifemp	""	""	""	""	""	"="	"="	""
allow	U	U+R	U	U	U	U	U	U+R

With the above table in mind, process the variable-list as follows:

For each varspec, extract a variable name and optional modifier from the expression by scanning the variable-list until a character not in the varname set is found or the end of the expression is reached.

- If it is the end of the expression and the varname is empty, go back to scan the remainder of the template.
- If it is not the end of the expression and the last character found indicates a modifier ("*" or ":"), remember that modifier. If it is an explode ("*"), scan the next character. If it is a prefix (":"), continue scanning the next one to four characters

for the max-length represented as a decimal integer and then, if it is still not the end of the expression, scan the next character.

- If it is not the end of the expression and the last character found is not a comma (","), append "{", the stored operator (if any), the scanned varname and modifier, the remaining expression, and "}" to the result string, remember that the result is in an error state, and then go back to scan the remainder of the template.

Lookup the value for the scanned variable name, and then

- If the varname is unknown or corresponds to a variable with an undefined value (Section 2.3), then skip to the next varspec.
- If this is the first defined variable for this expression, append the first string for this expression type to the result string and remember that it has been done. Otherwise, append the sep string to the result string.
- If this variable's value is a string, then
  - if named is true, append the varname to the result string using the same encoding process as for literals, and
    - if the value is empty, append the ifemp string to the result string and skip to the next varspec;
    - otherwise, append "=" to the result string.
  - if a prefix modifier is present and the prefix length is less than the value string length in number of Unicode characters, append that number of characters from the beginning of the value string to the result string, after pct-encoding any characters that are not in the allow set, while taking care not to split multi-octet or pct-encoded triplet characters that represent a single Unicode code point;
  - otherwise, append the value to the result string after pct-encoding any characters that are not in the allow set.
- else if no explode modifier is given, then
  - if named is true, append the varname to the result string using the same encoding process as for literals, and
    - if the value is empty, append the ifemp string to the result string and skip to the next varspec;
    - otherwise, append "=" to the result string; and
  - if this variable's value is a list, append each defined list member to the result string, after pct-encoding any characters that are not in the allow set, with a comma (",") appended to the result between each defined list member;
  - if this variable's value is an associative array or any other form of paired (name, value) structure, append each pair with a defined value to the result string as "name,value", after pct-encoding any characters that are not in the allow set, with a comma (",") appended to the result between each defined pair.

- else if an explode modifier is given, then
  - if named is true, then for each defined list member or array (name, value) pair with a defined value, do:
    - if this is not the first defined member/value, append the sep string to the result string;
    - if this is a list, append the varname to the result string using the same encoding process as for literals;
    - if this is a pair, append the name to the result string using the same encoding process as for literals;
    - if the member/value is empty, append the ifemp string to the result string; otherwise, append "=" and the member/value to the result string, after pct-encoding any member/value characters that are not in the allow set.
  - else if named is false, then
    - if this is a list, append each defined list member to the result string, after pct-encoding any characters that are not in the allow set, with the sep string appended to the result between each defined list member.
    - if this is an array of (name, value) pairs, append each pair with a defined value to the result string as "name=value", after pct-encoding any characters that are not in the allow set, with the sep string appended to the result between each defined pair.

When the variable-list for this expression is exhausted, go back to scan the remainder of the template.

Authors' Addresses

Joe Gregorio
Google

Email: joe@bitworking.org
URI: http://bitworking.org/

Roy T. Fielding
Adobe Systems Incorporated

Email: fielding@gbiv.com
URI: http://roy.gbiv.com/

Marc Hadley
The MITRE Corporation

Email: mhadley@mitre.org
URI: http://mitre.org/

Mark Nottingham
Rackspace

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

David Orchard
Salesforce.com

Email: orchard@pacificspirit.com
URI: http://www.pacificspirit.com/

Applications Area WG (APPSAWG) S. Moonesamy, Ed.
Internet-Draft
Intended Status: Informational
Expires: December 9, 2012 June 7, 2012

The "about" URI Scheme
draft-ietf-appsawg-about-uri-scheme-07

Abstract

This document describes the "about" URI scheme, which is widely used by web browsers and some other applications to designate access to their internal resources, such as settings, application information, hidden built-in functionality, and so on.

Table of Contents

1. Introduction

This document describes the "about" Uniform Resource Identifier (URI) scheme. The "about" URI scheme is currently widely used by Web browsers to designate access to their internal resources such as settings, application information, so called "Easter eggs" (i.e. hidden feature or joke in an application).

2. URI Scheme Specification

2.1. URI Scheme Syntax

The "about" URI syntactically conforms to the <about-uri> rule below, expressed using Augmented Backus-Naur Form (ABNF) [RFC5234]:

```
about-uri = "about:" about-token [ about-query ] [ about-fragment ]
about-token = *pchar
about-query = "?" query
about-fragment = "#" fragment
pchar     = <as specified in RFC 3986, Appendix A>
query     = <as specified in RFC 3986, Appendix A>
fragment  = <as specified in RFC 3986, Appendix A>
```

2.2. URI Scheme Semantics

The resource which a particular "about" URI references is denoted by <about-token> part of the URI. It is not a hierarchical element for a naming authority. The <about-query> specifies additional information about its handling and/or the information that should be returned by the resource which the URI references.

It is impossible to specify a binding between all the possible tokens and the semantics of "about" URIs that would contain such tokens. Therefore the resource referenced by the URI is generally considered as specific to a Web browser implementation.

2.2.1. Well-known "about" URIs

Some <about-token>s have been reserved as the behavior when the resource is referenced is well-known (Well-known tokens).

A well-known "about" URI is a URI that has a well-known token as its <about-token> part. It is recommended that such URIs be handled in accordance with the specification referenced in the Well-known Tokens registry (see Section 5.2).

Well-known "about" URIs are intended to be registered when there is a need to codify the behavior of particular <about-token>.

2.3. Encoding Considerations

"about" URIs are subject to encoding rules defined in RFC 3986 [RFC3986].

3. "about:blank"

This document defines one well-known token: "blank". The "about:blank" URI refers to a resource represented in the browser by a blank page.

4. Security Considerations

Security considerations for URIs are discussed in Section 7 of RFC 3986 [RFC3986]. However, most of those provisions do not apply to the "about" URI scheme as they are mainly scoped to schemes used in the Internet.

"about" URIs can sometimes refer to sensitive information, such as user passwords stored in a cache, or parameters that, if changed, could affect user’s data. The application therefore needs to ensure that the user’s data is secured and no threats are imposed

by "about" URIs.

5. IANA Considerations

5.1. URI Scheme Registration

The registration of the "about" URI scheme in the "URI Schemes" registry is requested. The information below is provided according to the guidelines from RFC 4395 [RFC4395]:

URI scheme name: about

Status: Permanent

URI scheme syntax: see Section 2.1 of RFC xxxx

URI scheme semantics: see Section 2.2 of RFC xxxx

URI scheme encoding considerations: see Section 2.3 of RFC xxxx

Applications that use the scheme: "about" URIs are predominantly used by Web browsers.

Security considerations: see Section 4 of RFC xxxx

Contact: IETF Applications Area Directors <app-ads@tools.ietf.org>

Author/Change controller: IESG <iesg@ietf.org> (on behalf of the IETF)

References: see Section 5 of RFC xxxx

[RFC Editor: Please replace xxxx with assigned RFC number]

5.2. A Registry for Well-known Tokens

This document creates the '"about" URI Well-known Tokens' registry.

The registry entries consist of three fields: Well-known Token, Description and Reference. The Well-known Token field has to conform to <about-token> production defined in Section 2.1. The initial set of assignments is as follows:

Well-known Token	Description	Reference
blank	The about:blank URI references a	RFC xxxx

	blank page.		

5.2.1. Registration procedure

The registration policy for this registry is "First Come First Served" as described in RFC 5226 [RFC5226]. The registrant of the token should provide the information mentioned in the following registration template:

- Registered Token: The desired Well-known token to be used in "about" URIs.
- Intended usage: A short description of how "about" URIs with the registered token is handled including information about the referenced resource.
- Contact/Change controller: Person (including contact information) authorized to change this registration.
- Specification: A stable reference to a document which specifies the registered "about" URI. The question of interoperability does not arise. The key motivation is to have a reference to a specification documenting well-known behavior of the "about" URI in Web browsers. As a rule of thumb if the behavior is common to two or more Web browser implementations it can be considered as well-known. An existing assignment may be duplicated if the registered token is used in more than one Web browser implementation.

The following is a template for the "blank" token:

- Registered Token: blank
- Intended usage: The about:blank URI references a blank page.
- Contact/Change controller: IESG <iesg@ietf.org> (on behalf of IETF).
- Specification: RFC xxxx. [RFC Editor: Please replace xxxx with assigned RFC number]

6. References

6.1. Normative References

[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.

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

6.2. Informative References

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

Appendix A. Acknowledgments

This document has been formed from the draft initially authored by Lachlan Hunt and Joseph Holsten. Additionally, the contributions of Frank Ellermann and Alexey Melnikov are gratefully acknowledged. Barry Leiba and Murray Kucherawy deserve a special credit for providing a great amount of text which has been used in this document.

Lachlan Hunt and Mykyta Yevstifeyev edited previous versions of this document. Tim Bray and John Klensin provided suggestions about how to improve the document.

Authors' Addresses

S. Moonesamy (editor)
76, Ylang Ylang Avenue
Quatre Bornes
Mauritius

EMail: sm+ietf@elandsys.com

Individual Submission
Internet-Draft
Obsoletes: 3462 (if approved)
Intended status: Standards Track
Expires: June 1, 2012

M. Kucherawy, Ed.
Cloudmark

November 29, 2011

The Multipart/Report Media Type for the Reporting of Mail System Administrative Messages
draft-ietf-appsawg-rfc3462bis-04

Abstract

The multipart/report Multipurpose Internet Mail Extensions (MIME) media type is a general "family" or "container" type for electronic mail reports of any kind. Although this memo defines only the use of the multipart/report media type with respect to delivery status reports, mail processing programs will benefit if a single media type is used for all kinds of reports.

This memo obsoletes RFC3462. The IESG is also requested to mark RFC1892 and RFC3462 as "historic".

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 1, 2012.

Table of Contents

1. Introduction

[OLD-REPORT] and its antecedent declared the multipart/report media type for use within the [MIME] construct to create a container for mail system administrative reports of various kinds.

Practical experience has shown that the general requirement of having that media type constrained to be used only as the outermost MIME type of a message is overly restrictive and limits such things as the transmission of multiple administrative reports within a single overall message container. In particular, it prevents one from forwarding a report as part of another multipart MIME message.

This memo removes that constraint. No other changes apart from some editorial ones are made. Other memos might update other documents to establish or clarify the constraints on use of multipart/report in contexts where such are needed.

This memo obsoletes RFC3462. The IESG is also requested to mark RFC1892 and RFC3462 as "historic".

2. Document Conventions

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 [KEYWORDS].

3. The multipart/report Media Type

The multipart/report MIME media type is a general "family" or "container" type for electronic mail reports of any kind. Although this memo defines only the use of the multipart/report media type with respect to delivery status reports, mail processing programs will benefit if a single media type is used for all kinds of reports.

Per [MIME-REG], the multipart/report media type is defined as follows:

Type name: multipart

Subtype name: report

Required parameters: boundary, report-type

Optional parameters: none

Encoding considerations: 7bit should always be adequate

Security considerations: see Section 7 of [this memo]

Interoperability considerations: see Section 1 of [this memo]

Published specification: [this memo]

Applications that use this media type: Mail Transfer Agents, Mail User Agents, spam detection and reporting modules, virus detection modules, message authentication modules

Additional information:

Magic number(s): N/A

File extensions(s): N/A

Macintosh file type code(s): N/A

Person and email address to contact for further information: Murray S. Kucherawy <msk@cloudmark.com>

Intended usage: common

Restrictions on usage: none; however, other applications that register report types may establish such restrictions

Author: Murray S. Kucherawy <msk@cloudmark.com>

Change controller: IESG

The syntax of multipart/report is identical to the multipart/mixed content type defined in [MIME]. The report-type parameter identifies the type of report. The parameter is the MIME sub-type of the second body part of the multipart/report. (See Section 5.)

The multipart/report media type contains either two or three sub-parts, in the following order:

1. (REQUIRED) The first body part contains a human readable message. The purpose of this message is to provide an easily understood description of the condition(s) that caused the report to be generated, for a human reader who might not have a user agent capable of interpreting the second section of the multipart/report. The text in the first section can use any IANA-registered MIME media type, charset, or language. Where a description of the error is desired in several languages or several media, a multipart/alternative construct MAY be used. This body part MAY also be used to send detailed information that cannot be easily formatted into the second body part.

2. (REQUIRED) A machine parsable body part containing an account of the reported message handling event. The purpose of this body part is to provide a machine-readable description of the condition(s) that caused the report to be generated, along with details not present in the first body part that might be useful to human experts. An initial body part, message/delivery-status is defined in [DSN-FORMAT].

3. (OPTIONAL) A body part containing the returned message or a portion thereof. This information could be useful to aid human experts in diagnosing problems. (Although it might also be useful to allow the sender to identify the message about which the report was issued, it is hoped that the envelope-id and original-recipient-address returned in the message/report body part will replace the traditional use of the returned content for this purpose.)

Return of content can be wasteful of network bandwidth and a variety of implementation strategies can be used. Generally the sender needs to choose the appropriate strategy and inform the recipient of the required level of returned content required. In the absence of an explicit request for level of return of content such as that provided in [DSN-SMTP], the agent that generated the delivery service report SHOULD return the full message content.

When 8-bit or binary data not encoded in a 7-bit form is to be returned, and the return path is not guaranteed to be 8-bit or binary capable, two options are available. The original message MAY be re-encoded into a legal 7-bit MIME message or the text/rfc822-headers media type MAY be used to return only the original message headers.

4. The text/rfc822-headers Media Type

The text/rfc822-headers media type provides a mechanism to label and return only the [MAIL] header of a failed message. The header is not the complete message and SHOULD NOT be returned using the message/rfc822 media type defined in [MIME-TYPES]. The returned header is useful for identifying the failed message and for diagnostics based on the Received header fields.

The text/rfc822-headers media type is defined as follows:

Type name: text

Subtype name: rfc822-headers

Required parameters: None

Optional parameters: None

Encoding considerations: 7-bit is sufficient for normal mail headers, however, if the headers are broken or extended and require encoding to make them legal 7-bit content, they MAY be encoded with quoted-printable as defined in [MIME]

Security considerations: See Section 7 of [this memo].

Interoperability considerations: none

Published specification: [this memo]

Applications that use this media type: Mail Transfer Agents, Mail User Agents, spam detection and reporting modules, virus detection modules, message authentication modules

Additional information:

Magic number(s): N/A

File extensions(s): N/A

Macintosh file type code(s): N/A

Person and email address to contact for further information: Murray S. Kucherawy <msk@cloudmark.com>

Intended usage: common

Restrictions on usage: none

Author: Murray S. Kucherawy <msk@cloudmark.com>

Change controller: IESG

The text/rfc822-headers body part SHOULD contain all the mail header fields from the message that caused the report. The header includes all header fields prior to the first blank line in the message. They include the MIME-Version and MIME content description fields.

5. Registering New Report Types

Registration of new media types for the purpose of creating a new report format SHOULD note in the Intended Usage section of the media type registration that the type being registered is suitable for use as a report-type (i.e., the second body part) in the context of this specification.

6. IANA Considerations

IANA is directed to update the Media Type Registry to indicate that this memo contains the current definition of the multipart/report and text/rfc822-headers media types, obsoleting [OLD-REPORT].

7. Security Considerations

Automated use of report types without authentication presents several security issues. Forging negative reports presents the opportunity for denial-of-service attacks when the reports are used for automated maintenance of directories or mailing lists. Forging positive reports can cause the sender to incorrectly believe a message was delivered when it was not.

A signature covering the entire multipart/report structure could be used to prevent such forgeries; such a signature scheme is, however, beyond the scope of this document.

8. References

8.1. Normative References

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

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

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

[MIME-REG] Freed, N. and J. Klensin, "Media Type Specifications and Registration Procedures", RFC 4288, December 2005.

[MIME-TYPES] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

8.2. Informative References

[DSN-FORMAT] Moore, K. and G. Vaudreuil, "An Extensible Message Format for Delivery Status Notifications", RFC 3464, January 2003.

[DSN-SMTP] Moore, K., "Simple Mail Transfer Protocol (SMTP) Service Extension for Delivery Status Notifications (DSNs)", RFC 3461, January 2003.

[OLD-REPORT] Vaudreuil, G., "The Multipart/Report Content Type for the Reporting of Mail System Administrative Messages", RFC 3462, January 2003.

Appendix A. Acknowledgements

The author would like to thank Dave Crocker, Frank Ellermann, Ned Freed, Randall Gellens, Alexey Melnikov and Keith Moore for their input to this update.

Thanks also go to Gregory M. Vaudreuil, the original creator of this media type.

Appendix B. Document History

[RFC Editor: Please remove this section prior to publication.]

Changes from draft-ietf-appsawg-rfc3462bis-01 to draft-ietf-appsawg-rfc3462bis-02:

- Minor copy editing based on WGLC feedback.
- Make OLD-REPORT into an informative reference.
- Update media type registration templates.

Changes from draft-ietf-appsawg-rfc3462bis-00 to draft-ietf-appsawg-rfc3462bis-01:

- Minor copy editing based on WG feedback.

Changes from draft-kucherawy-rfc3462bis-02 to draft-ietf-appsawg-rfc3462bis-00:

- Renamed.

Changes from draft-kucherawy-rfc3462bis-01 to draft-kucherawy-rfc3462bis-02:

- Revert to removing the restriction altogether, noting that the DSN and MDN RFCs re-state it. Thus, removing it here solves MARF's problem but doesn't impact DSN and MDN. The restriction can be clarified on those documents in separate efforts.

Changes from draft-kucherawy-rfc3462bis-00 to draft-kucherawy-rfc3462bis-01:

- Clarify requirement that multipart/report must be the outermost media type; require it only when generating a report.
- Highlight the forwarding-of-reports problem.
- Limit the constraint to time of report generation.
- Remove "Examples" section.

Changes from RFC3462 to draft-kucherawy-rfc3462bis-00:

- Remove requirement that multipart/report not be contained in anything.

- Some minor adjustment to use current terminology, such as distinguishing between a header and a header field.

- More obvious use of the standard normative words.

Author's Address

Murray S. Kucherawy (editor)
Cloudmark
128 King St., 2nd Floor
San Francisco, CA 94107
US

Phone: +1 415 946 3800
Email: msk@cloudmark.com

APPSAWG
Internet-Draft
Intended status: BCP
Expires: October 11, 2012

P. Saint-Andre
Cisco Systems, Inc.
D. Crocker
Brandenburg InternetWorking
M. Nottingham
Rackspace

April 9, 2012

Deprecating the X- Prefix and Similar Constructs in Application Protocols
draft-ietf-appsawg-xdash-05

Abstract

Historically, designers and implementers of application protocols have often distinguished between standardized and unstandardized parameters by prefixing the names of unstandardized parameters with the string "X-" or similar constructs. In practice, that convention causes more problems than it solves. Therefore, this document deprecates the convention for newly-defined parameters with textual (as opposed to numerical) names in application protocols.

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 11, 2012.

Copyright Notice

Table of Contents

1. Introduction

Many application protocols use parameters with textual (as opposed to numerical) names to identify data (media types, header fields in Internet mail messages and HTTP requests, vCard parameters and properties, etc.). Historically, designers and implementers of application protocols have often distinguished between standardized and unstandardized parameters by prefixing the names of unstandardized parameters with the string "X-" or similar constructs (e.g., "x."), where the "X" is commonly understood to stand for "eXperimental" or "eXtension".

Under this convention, the name of a parameter not only identified the data, but also embedded the status of the parameter into the name itself: a parameter defined in a specification produced by a recognized standards development organization (or registered according to processes defined in such a specification) did not start with "X-" or similar constructs, whereas a parameter defined outside such a specification or process started with "X-" or similar constructs.

As explained more fully under Appendix A, this convention was encouraged for many years in application protocols such as file transfer, email, and the World Wide Web. In particular, it was codified for email by [RFC822] (via the distinction between "Extension-fields" and "user-defined-fields"), but then removed by [RFC2822] based on implementation and deployment experience. A similar progression occurred for SIP technologies with regard to the "P-" header, as explained in [RFC5727]. The reasoning behind those changes is explored under Appendix B.

In short, although in theory the "X-" convention was a good way to avoid collisions (and attendant interoperability problems) between standardized parameters and unstandardized parameters, in practice the benefits have been outweighed by the costs associated with the leakage of unstandardized parameters into the standards space.

This document generalizes from the experience of the email and SIP communities by doing the following:

1. Deprecates the "X-" convention for newly-defined parameters in application protocols, even where that convention was only implicit instead of being codified in a protocol specification (as was done for email in [RFC822]).

2. Makes specific recommendations about how to proceed in a world without the distinction between standardized and unstandardized parameters (although only for parameters with textual names, not

parameters that are expressed as numbers, which are out of scope).

3. Does not recommend against the practice of private, local, preliminary, experimental, or implementation-specific parameters, only against the use of "X-" and similar constructs in the names of such parameters.

4. Makes no recommendation as to whether existing "X-" parameters ought to remain in use or be migrated to a format without the "X-"; this is a matter for the creators or maintainers of those parameters.

5. Does not override existing specifications that legislate the use of "X-" for particular application protocols (e.g., the "x-name" token in [RFC5545]); this is a matter for the designers of those protocols.

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].

2. Recommendations for Implementers of Application Protocols

Implementations of application protocols MUST NOT make any assumptions about the status of a parameter, nor take automatic action regarding a parameter, based solely on the presence or absence of "X-" or a similar construct in the parameter's name.

3. Recommendations for Creators of New Parameters

Creators of new parameters to be used in the context of application protocols:

1. SHOULD assume that all parameters they create might become standardized, public, commonly deployed, or used across multiple implementations.

2. SHOULD employ meaningful parameter names that they have reason to believe are currently unused.

3. SHOULD NOT prefix their parameter names with "X-" or similar constructs.

Note: If the relevant parameter name space has conventions about

associating parameter names with those who create them, a parameter name could incorporate the organization's name or primary domain name (see Appendix B for examples).

4. Recommendations for Protocol Designers

Designers of new application protocols that allow extensions using parameters:

1. SHOULD establish registries with potentially unlimited value-spaces, if appropriate defining both permanent and provisional registries.

2. SHOULD define simple, clear registration procedures.

3. SHOULD mandate registration of all non-private parameters, independent of the form of the parameter names.

4. SHOULD NOT prohibit parameters with the "X-" prefix or similar constructs from being registered.

5. MUST NOT assume that a parameter with an "X-" prefix or similar constructs is unstandardized.

6. MUST NOT assume that a parameter without an "X-" prefix or similar constructs is standard.

5. Security Considerations

Interoperability and migration issues with security-critical parameters can result in unnecessary vulnerabilities (see Appendix B for further discussion).

As a corollary to the recommendation provided under Section 2, implementations MUST NOT assume that standardized parameters are "secure" whereas unstandardized parameters are "insecure", based solely on the names of such parameters.

6. IANA Considerations

This document does not modify registration procedures currently in force for various application protocols. However, such procedures might be updated in the future to incorporate the best practices defined in this document.

7. Acknowledgements

Thanks to Claudio Allocchio, Adam Barth, Nathaniel Borenstein, Eric Burger, Stuart Cheshire, Al Constanzo, Dave Cridland, Ralph Droms, Martin Duerst, Frank Ellermann, J.D. Falk, Ned Freed, Tony Finch, Randall Gellens, Tony Hansen, Ted Hardie, Joe Hildebrand, Alfred Hoenes, Paul Hoffman, Eric Johnson, Scott Kelly, Scott Kitterman, John Klensin, Graham Klyne, Murray Kucherawy, Eliot Lear, John Levine, Bill McQuillan, Alexey Melnikov, Subramanian Moonesamy, Keith Moore, Ben Niven-Jenkins, Zoltan Ordogh, Tim Petch, Dirk Pranke, Randy Presuhn, Julian Reschke, Dan Romascanu, Doug Royer, Andrew Sullivan, Henry Thompson, Martin Thomson, Matthew Wild, Nicolas Williams, Tim Williams, Mykyta Yevstifeyev, and Kurt Zeilenga for their feedback.

8. References

8.1. Normative References

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

8.2. Informative References

[BCP9] Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.

[BCP26] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[BCP82] Narten, T., "Assigning Experimental and Testing Numbers Considered Useful", BCP 82, RFC 3692, January 2004.

[RFC691] Harvey, B., "One more try on the FTP", RFC 691, June 1975.

[RFC737] Harrenstien, K., "FTP extension: XSEN", RFC 737, October 1977.

[RFC743] Harrenstien, K., "FTP extension: XRSQ/XRCP", RFC 743, December 1977.

[RFC775] Mankins, D., Franklin, D., and A. Owen, "Directory oriented FTP commands", RFC 775, December 1980.

[RFC822] Crocker, D., "Standard for the format of ARPA Internet text messages", STD 11, RFC 822, August 1982.

[RFC1123] Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, October 1989.

[RFC1154] Robinson, D. and R. Ullmann, "Encoding header field for internet messages", RFC 1154, April 1990.

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

[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.

[RFC2068] Fielding, R., Gettys, J., Mogul, J., Nielsen, H., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068, January 1997.

[RFC2426] Dawson, F. and T. Howes, "vCard MIME Directory Profile", RFC 2426, September 1998.

[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.

[RFC2822] Resnick, P., "Internet Message Format", RFC 2822, April 2001.

[RFC2939] Droms, R., "Procedures and IANA Guidelines for Definition of New DHCP Options and Message Types", BCP 43, RFC 2939, September 2000.

[RFC3406] Daigle, L., van Gulik, D., Iannella, R., and P. Faltstrom, "Uniform Resource Names (URN) Namespace Definition Mechanisms", BCP 66, RFC 3406, October 2002.

[RFC3427] Mankin, A., Bradner, S., Mahy, R., Willis, D., Ott, J., and B. Rosen, "Change Process for the Session Initiation Protocol (SIP)", RFC 3427, December 2002.

[RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, September 2004.

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

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, July 2005.

[RFC4288] Freed, N. and J. Klensin, "Media Type Specifications and Registration Procedures", BCP 13, RFC 4288, December 2005.

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

[RFC4512] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP): Directory Information Models", RFC 4512, June 2006.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.

[RFC5064] Duerst, M., "The Archived-At Message Header Field", RFC 5064, December 2007.

[RFC5451] Kucherawy, M., "Message Header Field for Indicating Message Authentication Status", RFC 5451, April 2009.

[RFC5545] Desruisseaux, B., "Internet Calendaring and Scheduling Core Object Specification (iCalendar)", RFC 5545, September 2009.

[RFC5646] Phillips, A. and M. Davis, "Tags for Identifying Languages", BCP 47, RFC 5646, September 2009.

[RFC5727] Peterson, J., Jennings, C., and R. Sparks, "Change Process for the Session Initiation Protocol (SIP) and the Real-time Applications and Infrastructure Area", BCP 67, RFC 5727, March 2010.

Appendix A. Background

The beginnings of the "X-" convention can be found in a suggestion made by Brian Harvey in 1975 with regard to FTP parameters [RFC691]:

> Thus, FTP servers which care about the distinction between Telnet print and non-print could implement SRVR N and SRVR T. Ideally the SRVR parameters should be registered with Jon Postel to avoid conflicts, although it is not a disaster if two sites use the same parameter for different things. I suggest that parameters be allowed to be more than one letter, and that an initial letter X be used for really local idiosyncracies.

This "X" prefix was subsequently used in [RFC737], [RFC743], and [RFC775]. This usage was noted in [RFC1123]:

> FTP allows "experimental" commands, whose names begin with "X". If these commands are subsequently adopted as standards, there may still be existing implementations using the "X" form.... All FTP implementations SHOULD recognize both forms of these commands, by simply equating them with extra entries in the command lookup table.

The "X-" convention has been used for email header fields since at least the publication of [RFC822] in 1982, which distinguished between "Extension-fields" and "user-defined-fields" as follows:

> The prefatory string "X-" will never be used in the names of Extension-fields. This provides user-defined fields with a protected set of names.

That rule was restated by [RFC1154] as follows:

> Keywords beginning with "X-" are permanently reserved to implementation-specific use. No standard registered encoding keyword will ever begin with "X-".

This convention continued with various specifications for media types ([RFC2045], [RFC2046], [RFC2047]), HTTP headers ([RFC2068], [RFC2616]), vCard parameters and properties ([RFC2426]), Uniform Resource Names ([RFC3406]), LDAP field names ([RFC4512]), and other application technologies.

However, use of the "X-" prefix in email headers was effectively deprecated between the publication of [RFC822] in 1982 and the publication of [RFC2822] in 2001 by removing the distinction between the "extension-field" construct and the "user-defined-field" construct (a similar change happened with regard to Session Initiation Protocol "P-" headers when [RFC3427] was obsoleted by [RFC5727]).

Despite the fact that parameters containing the "X-" string have been effectively deprecated in email headers, they continue to be used in

a wide variety of application protocols. The two primary situations motivating such use are:

1. Experiments that are intended to possibly be standardized in the future, if they are successful.

2. Extensions that are intended to never be standardized because they are intended only for implementation-specific use or for local use on private networks.

Use of this naming convention is not mandated by the Internet Standards Process [BCP9] or IANA registration rules [BCP26]. Rather it is an individual choice by each specification that references the convention or each administrative process that chooses to use it. In particular, some standards-track RFCs have interpreted the convention in a normative way (e.g., [RFC822] and [RFC5451]).

Appendix B. Analysis

The primary problem with the "X-" convention is that unstandardized parameters have a tendency to leak into the protected space of standardized parameters, thus introducing the need for migration from the "X-" name to a standardized name. Migration, in turn, introduces interoperability issues (and sometimes security issues) because older implementations will support only the "X-" name and newer implementations might support only the standardized name. To preserve interoperability, newer implementations simply support the "X-" name forever, which means that the unstandardized name has become a de facto standard (thus obviating the need for segregation of the name space into standardized and unstandardized areas in the first place).

We have already seen this phenomenon at work with regard to FTP in the quote from [RFC1123] in the previous section. The HTTP community had the same experience with the "x-gzip" and "x-compress" media types, as noted in [RFC2068]:

> For compatibility with previous implementations of HTTP, applications should consider "x-gzip" and "x-compress" to be equivalent to "gzip" and "compress" respectively.

A similar example can be found in [RFC5064], which defined the "Archived-At" message header field but also found it necessary to define and register the "X-Archived-At" field:

> For backwards compatibility, this document also describes the X-Archived-At header field, a precursor of the Archived-At header field. The X-Archived-At header field MAY also be parsed, but SHOULD NOT be generated.

One of the original reasons for segregation of name spaces into standardized and unstandardized areas was the perceived difficulty of registering names. However, the solution to that problem has been simpler registration rules, such as those provided by [RFC3864] and [RFC4288]. As explained in [RFC4288]:

> [W]ith the simplified registration procedures described above for vendor and personal trees, it should rarely, if ever, be necessary to use unregistered experimental types. Therefore, use of both "x-" and "x." forms is discouraged.

For some name spaces, another helpful practice has been the establishment of separate registries for permanent names and provisional names, as in [RFC4395].

Furthermore, often standardization of a unstandardized parameter leads to subtly different behavior (e.g., the standardized version might have different security properties as a result of security review provided during the standardization process). If implementers treat the old, unstandardized parameter and the new, standardized parameter as equivalent, interoperability and security problems can ensue. Analysis of unstandardized parameters to detect and correct flaws is in general a good thing and is not intended to be discouraged by the lack of distinction in element names. Whenever an originally unstandardized parameter or protocol element is standardized and the new form has differences which affect interoperability or security properties, implementations MUST NOT treat the old form as identical to the new form.

For similar considerations with regard to the "P-" convention in the Session Initiation Protocol, see [RFC5727].

In some situations, segregating the parameter name space used in a given application protocol can be justified:

1. When it is extremely unlikely that some parameters will ever be standardized. In this case implementation-specific and private-use parameters could at least incorporate the organization's name (e.g., "ExampleInc-foo" or, consistent with [RFC4288], "VND.ExampleInc.foo") or primary domain name (e.g., "com.example.foo" or a Uniform Resource Identifier [RFC3986] such as "http://example.com/foo"). In rare cases, truly experimental parameters could be given meaningless names such as nonsense

words, the output of a hash function, or UUIDs [RFC4122].

2. When parameter names might have significant meaning. This case too is rare, since implementers can almost always find a synonym for an existing term (e.g., "urgency" instead of "priority") or simply invent a more creative name (e.g., "get-it-there-fast"). The existence of multiple similarly-named paramaters can be confusing, but this is true regardless if there is an attempt to segregate standardized and unstandardized (e.g., "X-Priority" can be confused with "Urgency").

3. When parameter names need to be very short (e.g., as in [RFC5646] for language tags). In this case it can be more efficient to assign numbers instead of human-readable names (e.g., as in [RFC2939] for DCHP options) and to leave a certain numeric range for implementation-specific extensions or private use (e.g., as with the codec numbers used with the Session Description Protocol [RFC4566]).

There are three primary objections to deprecating the "X-" convention as a best practice for application protocols:

1. Implementers might mistake one parameter for another parameter that has a similar name; a rigid distinction such as an "X-" prefix can make this clear. However, in practice implementers are forced to blur the distinction (e.g., by treating "X-foo" as a de facto standard) and so it inevitably becomes meaningless.

2. Collisions are undesirable and it would be bad for both a standardized parameter "foo" and a unstandardized parameter "foo" to exist simultaneously. However, names are almost always cheap, so an experimental, implementation-specific, or private-use name of "foo" does not prevent a standards development organization from issuing a similarly creative name such as "bar".

3. [BCP82] is entitled "Assigning Experimental and Testing Numbers Considered Useful" and therefore implies that the "X-" prefix is also useful for experimental parameters. However, BCP 82 addresses the need for protocol numbers when the pool of such numbers is strictly limited (e.g., DHCP options) or when a number is absolutely required even for purely experimental purposes (e.g., the Protocol field of the IP header). In almost all application protocols that make use of protocol parameters (including email headers, media types, HTTP headers, vCard parameters and properties, URNs, and LDAP field names), the name space is not limited or constrained in any way, so there is no need to assign a block of names for private use or experimental purposes (see also [BCP26]).

Therefore it appears that segregating the parameter space into a standardized area and a unstandardized area has few if any benefits, and has at least one significant cost in terms of interoperability.

Authors' Addresses

Peter Saint-Andre
Cisco Systems, Inc.
1899 Wynkoop Street, Suite 600
Denver, CO 80202
USA

Phone: +1-303-308-3282
Email: psaintan@cisco.com

D. Crocker
Brandenburg InternetWorking
675 Spruce Dr.
Sunnyvale
USA

Phone: +1.408.246.8253
Email: dcrocker@bbiw.net
URI: http://bbiw.net

Mark Nottingham
Rackspace

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

Individual submission
Internet-Draft
Intended status: BCP
Expires: April 26, 2012

M. Kucherawy
Cloudmark, Inc.

October 24, 2011

Best Current Practices for Email Greylisting
draft-kucherawy-greylisting-bcp-01

Abstract

This memo describes best current practices for the art of email greylisting, the practice of providing temporarily degraded service to unknown email clients as an anti-abuse mechanism.

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 26, 2012.

Copyright Notice

Table of Contents

1. Introduction

There are many techniques in use for dealing with email abuse. One is a set of techniques known as "greylisting". Broadly, this refers to any degradation of service for an unknown or suspect source, over a period of time. The narrow use of the term refers to generation of an SMTP temporary failure reply code for traffic from such sources.

There are diverse implementations of this general technique, and, predictably therefore, some blurred terminology.

This memo documents common greylisting techniques and discusses their benefits and costs. It also defines terminology to enable clear distinction and discussion of these techniques.

2. Definitions

2.1. Keywords

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 [KEYWORDS].

2.2. E-Mail Architecture Terminology

Readers should be familiar with the material and terminology discussed in [MAIL] and [EMAIL-ARCH].

3. Deciding Who Is Affected

This section will discuss how it is decided whether or not a particular client session, or specific message, will be selected for greylisting. Discuss selection criteria, e.g., {IP} vs. {IP, from, to}.

4. Connection-Level Greylisting

This section will talk about greylisting applied at the time of decision about whether or not to accept a new connection, even before SMTP begins to take place.

5. SMTP HELO/EHLO Greylisting

This section will talk about greylisting applied within the [SMTP] session at the HELO/EHLO phase.

6. SMTP MAIL Greylisting

This section will talk about greylisting applied within the [SMTP] session at the MAIL FROM phase.

7. SMTP RCPT Greylisting

This section will talk about greylisting applied within the [SMTP] session at the RCPT TO phase.

8. SMTP DATA Greylisting

This section will talk about greylisting applied within the [SMTP] session at the DATA phase.

Some implementations do filtering here because there are clients that don't bother checking SMTP reply codes to commands other than DATA.

9. Effects on Clients

This section will discuss the behaviours of SMTP clients when greylisting is in effect, such as:

- very long retry times
- aggressive retries can hit rate limits
- incorrect handling of greylisting replies (e.g., treat 4xx like 5xx)
- retries may change envelope sender

10. Benefits and Costs

This section will discuss the benefits and also the costs (resources and impacts on generals ervice) of the various implementations.

Discuss failure modes, including:

- all retries fail
- retries go to a different server that doesn't know about previous attempts
- retries come from a different client than earlier ones
- for systems that use body hashes, the retries aren't the same as the previous attempts

11. Recommendations

This section will provide some general recommendations about when and how to deploy greylisting in various conceptual environments.

Some points to discuss:

- logging of a greylisting server vs. one not greylisting can be a good measure of how effective it is
- can also compare greylisting results to DNSBLs and content filtering
- greylisting is more expensive than not greylisting
- greylisting delays legitimate mail, and can cause conversations to arrive out of order
- time limits for greylisting
- special actions to take if the same message is retried before the time limit expires
- recommended termiantion methods (421 vs. 4xx)
- affects/requirements on MXes other than the lowest
- ability to share information between servers

12. IANA Considerations

No actions are requested of IANA in this memo.

13. Security Considerations

This section discusses potential security issues related to greylisting.

14. References

14.1. Normative References

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

14.2. Informative References

[EMAIL-ARCH] Crocker, D., "Internet Mail Architecture", RFC 5598, October 2008.

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

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

Appendix A. Acknowledgments

The author wishes to acknowledge Mike Adkins, Steve Atkins, Dave Crocker, Peter J. Holzer, John Levine, Jose-Marcio Martins da Cruz, Jordan Rosenwald, Gregory Shapiro, and Joe Sniderman for their contributions to this memo.

Author's Address

Murray S. Kucherawy
Cloudmark, Inc.
128 King St., 2nd Floor
San Francisco, CA 94107
US

Phone: +1 415 946 3800
EMail: msk@cloudmark.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 26, 2012

B. Nordman
Lawrence Berkeley National
Laboratory

October 24, 2011

Basic Device Classification
draft-nordman-classification-00

Abstract

This specification addresses how to communicate a basic sense of the type of each device present on the network. This is particularly important as the range of devices with connectivity greatly expands, and as indirect interests in device characteristics increase, such as energy consumption. Many applications will benefit from a single standard enumeration of basic device types. This draft does not address detailed device characteristics or subtypes.

The draft also discusses related identify information. It is an initial discussion document to generate feedback and improvement.

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 26, 2012.

Copyright Notice

1. Overview

As the number of types of devices that are connected to the Internet increases, it will be increasingly necessary for devices to make decisions based on the types of devices they encounter. In particular, devices may be able to discover what other devices share the same physical space, so that the range of devices that they find may be large. Basic decisions about whether or not to respond to a device can be informed by understanding the fundamental nature of each device. Detailed understanding likely requires device-specific information; this draft does not attempt to do this; it only covers a first layer of classification.

Classification is proposed to be represented as a 2-byte value that corresponds to an IANA registry of devices.

Other information also contributes to the identity of a device, such as brand and model, and would benefit from consistency in naming and availability.

2. 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 2119 [RFC2119].

3. Other Relevant Standards

There are many standards that touch on topics related to this one, though generally they have a different specific purpose, or address just a small subset of the devices in scope for this standard. Thus, while they should inform the discussion, none are suitable as a replacement for or substantial basis for this one.

Some example other standards and classifications are:

* The Distributed Management Task Force (DMTF) defines a set of entity times in its Common Information Model (CIM). These include both entire devices and components of devices (components are out of scope of this proposal).
* Systems for labeling retail products (e.g. Universal Product Code, UPC) embody a detailed listing of product types.
* The United Nations publishes a Standard Products and Services Code, for use in facilitating electronic commerce.
* There are efforts to develop standard taxonomies for heating and cooling systems in large commercial buildings. This is particularly needed since that industry has been long dominated by proprietary protocols, each with their own internal taxonomy.
* Some companies have implemented information systems to describe devices. An example is the Cisco Products MIB, which provides for a device to report its model number.

4. Classification and Identity

There are many standards that speak to identity on the network, or service discovery. This document does not address those topics. Rather, the classification addressed is comparable to the first impression a human being might have on a device, to recognize its core function or type.

4.1. General Issues

An analogy can be drawn to the ASCII character coding system. It specifies a single simple and compact numeric correspondence for letters, numbers, and symbols. It makes no attempt to cover characteristics other than the character's identity such as font, size, etc. Because of its simplicity, it can be utilized universally, or nearly so.

The concept of identity is usually for information which is unique to the entity at hand, rather than data which puts it into a group of entities.

Classification of populations of entities is commonly done with a taxonomy; a system of organization or categories. Taxonomies often have multiple layers of organization with groupings having one or more common characteristics; the most widely known of these is the biological classification system which has seven layers or grouping.

4.2. Registry Proposal

Classification should be a characteristic of a device that never changes, though it may be unnecessary to prohibit this.

Classification of a device is self-determined. It will be desirable to have standard translations of the classification code into all major languages.

A possible implementation of this is an IANA registry of 2-byte values for class. Each device would be a member of at most one class. The class for a particular device would be set by the manufacturer.

This proposal is for a single listing of device types, as the choice of criteria for grouping might change with the particular application. As the number of device types anticipated is not so large, an application can readily impose its own categorization system on top of the basic classes.

Some devices have more than one major function (e.g. a combination television and DVD player). While it would be possible to allow more than one classification, this draft is crafted on the assumption that this is unnecessary.

This mechanism is not intended to solve all problems with an application understanding characteristics of devices it encounters. There are many existing mechanisms for service discovery and the like which enable much more detailed information, but none of these are universal.

Categories would be drawn broadly. For example, a likely one is "refrigerator" which would encompass any device that cools its inside space regardless of size, technology, features, or intended market (e.g. residential or commercial). Obtaining more detailed information would require a different mechanism.

4.3. Application to Energy Management

One application that would benefit from such a classification system is Energy Management. A management system for energy will gather up data about all devices in a building, and have no functional relationship to many of them. The energy information collected is generic to any device type; the data are unrelated to the services that the device provides functionally. Classification information may be the only data that the management system has about the device other than its identity on the network. Thus, any classification (and identity) data should be universal. Also, many buildings will be partly or entirely unmanaged, with some devices that come and go asynchronously so that classification information should be gathered automatically.

With classification data, a management system can readily and

automatically provide a breakdown of energy use by major category of device. A person may be investigating energy use in a building after some devices have left the network, or in a situation where they lack direct access to the network (an example of the latter is where the analysis is outsourced to a third party). Thus, it is desirable that the most critical information be available in what is reported during energy management queries.

5. Related Needs

Beyond the basic class, it is often necessary or desirable to have the manufacturer's brand name and model number. These can be used in reports, or to help gather additional information about a device. These are text strings of some modest length. A human-readable name is often associated with such information. There are some existing MIBs which have these variables.

Related to this, manufacturers often have a web page for information about each model. It would be helpful if a device could report a URL for such a page. Additionally, the page could have both human-readable and machine-readable portions, with the latter specified according to some standard format. This would enable useful information about devices to be gathered automatically by a management system.

Many applications that use classification will also want to know the current location of a device, either geographically, or within a room or building. This draft does not speak to location but it is important.

6. Issues

Should there be a repository somewhere of standard translations from IANA code to human languages? Where?

What issues are introduced when classification is proxied for non-IP devices?

7. Security Considerations

In general none. It is worth noting that a device may report a type other than what it is.

8. Privacy Considerations

None.

9. Acknowledgement

We would like to thank <get your name here>.

10. Normative References

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

Author's Address

Bruce Nordman
Lawrence Berkeley National Laboratory
1 Cyclotron Road
Berkeley, CA 94720
USA

Email: BNordman@LBL.gov

Network Working Group
Internet-Draft
Updates: 2616 (if approved)
Intended status: Standards Track
Expires: August 7, 2012

M. Nottingham
Rackspace
R. Fielding
Adobe

February 4, 2012

Additional HTTP Status Codes
draft-nottingham-http-new-status-04

Abstract

This document specifies additional HyperText Transfer Protocol (HTTP) status codes for a variety of common situations.

Editorial Note (To be removed by RFC Editor before publication)

Distribution of this document is unlimited. Although this is not a work item of the HTTPbis Working Group, comments should be sent to the Hypertext Transfer Protocol (HTTP) mailing list at ietf-http-wg@w3.org [1], which may be joined by sending a message with subject "subscribe" to ietf-http-wg-request@w3.org [2].

Discussions of the HTTPbis Working Group are archived at <http://lists.w3.org/Archives/Public/ietf-http-wg/>.

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 August 7, 2012.

Copyright Notice

Table of Contents

1. Introduction

This document specifies additional HTTP [RFC2616] status codes for a variety of common situations, to improve interoperability and avoid confusion when other, less precise status codes are used.

Note that these status codes are optional; servers cannot be required to support them. However, because clients will treat unknown status codes as a generic error of the same class (e.g., 499 is treated as 400 if it is not recognized), they can be safely deployed by existing servers (see [RFC2616] Section 6.1.1 for more information).

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. 428 Precondition Required

The 428 status code indicates that the origin server requires the request to be conditional.

Its typical use is to avoid the "lost update" problem, where a client GETs a resource's state, modifies it, and PUTs it back to the server, when meanwhile a third party has modified the state on the server, leading to a conflict. By requiring requests to be conditional, the server can assure that clients are working with the correct copies.

Responses using this status code SHOULD explain how to resubmit the request successfully. For example:

```
HTTP/1.1 428 Precondition Required
Content-Type: text/html

<html>
 <head>
  <title>Precondition Required</title>
 </head>
 <body>
  <h1>Precondition Required</h1>
   <p>This request is required to be conditional;
      try using "If-Match".</p>
 </body>
</html>
```

Responses with the 428 status code MUST NOT be stored by a cache.

4. 429 Too Many Requests

The 429 status code indicates that the user has sent too many requests in a given amount of time ("rate limiting").

The response representations SHOULD include details explaining the condition, and MAY include a Retry-After header indicating how long to wait before making a new request.

For example:

```
HTTP/1.1 429 Too Many Requests
Content-Type: text/html
Retry-After: 3600

<html>
   <head>
      <title>Too Many Requests</title>
   </head>
   <body>
      <h1>Too Many Requests</h1>
      <p>I only allow 50 requests per hour to this Web site per
         logged in user. Try again soon.</p>
   </body>
</html>
```

Note that this specification does not define how the origin server identifies the user, nor how it counts requests. For example, an origin server that is limiting request rates can do so based upon counts of requests on a per-resource basis, across the entire server, or even among a set of servers. Likewise, it might identify the user by its authentication credentials, or a stateful cookie.

Responses with the 429 status code MUST NOT be stored by a cache.

5. 431 Request Header Fields Too Large

The 431 status code indicates that the server is unwilling to process the request because its header fields are too large. The request MAY be resubmitted after reducing the size of the request header fields.

It can be used both when the set of request header fields in total are too large, and when a single header field is at fault. In the latter case, the response representation SHOULD specify which header

field was too large.

For example:

```
HTTP/1.1 431 Request Header Fields Too Large
Content-Type: text/html

<html>
   <head>
      <title>Request Header Fields Too Large</title>
   </head>
   <body>
      <h1>Request Header Fields Too Large</h1>
      <p>The "Example" header was too large.</p>
   </body>
</html>
```

Responses with the 431 status code MUST NOT be stored by a cache.

6. 511 Network Authentication Required

The 511 status code indicates that the client needs to authenticate to gain network access.

The response representation SHOULD contain a link to a resource that allows the user to submit credentials (e.g. with a HTML form).

Note that the 511 response SHOULD NOT contain a challenge or the login interface itself, because browsers would show the login interface as being associated with the originally requested URL, which may cause confusion.

The 511 status SHOULD NOT be generated by origin servers; it is intended for use by intercepting proxies that are interposed as a means of controlling access to the network.

Responses with the 511 status code MUST NOT be stored by a cache.

6.1. The 511 Status Code and Captive Portals

The 511 status code is designed to mitigate problems caused by "captive portals" to software (especially non-browser agents) that is expecting a response from the server that a request was made to, not the intervening network infrastructure. It is not intended to encouraged deployment of captive portals, only to limit the damage caused by them.

A network operator wishing to require some authentication, acceptance of terms or other user interaction before granting access usually does so by identifing clients who have not done so ("unknown clients") using their MAC addresses.

Unknown clients then have all traffic blocked, except for that on TCP port 80, which is sent to a HTTP server (the "login server") dedicated to "logging in" unknown clients, and of course traffic to the login server itself.

For example, a user agent might connect to a network and make the following HTTP request on TCP port 80:

```
GET /index.htm HTTP/1.1
Host: www.example.com
```

Upon receiving such a request, the login server would generate a 511 response:

```
HTTP/1.1 511 Network Authentication Required
Content-Type: text/html

<html>
   <head>
      <title>Network Authentication Required</title>
      <meta http-equiv="refresh"
            content="0; url=https://login.example.net/">
   </head>
   <body>
      <p>You need to <a href="https://login.example.net/">
      authenticate with the local network</a> in order to gain
      access.</p>
   </body>
</html>
```

Here, the 511 status code assures that non-browser clients will not interpret the response as being from the origin server, and the META HTML element redirects the user agent to the login server.

7. Security Considerations

7.1. 428 Precondition Required

The 428 status code is optional; clients cannot rely upon its use to prevent "lost update" conflicts.

7.2. 429 Too Many Requests

When a server is under attack or just receiving a very large number of requests from a single party, responding to each with a 429 status code will consume resources.

Therefore, servers are not required to use the 429 status code; when limiting resource usage, it may be more appropriate to just drop connections, or take other steps.

7.3. 431 Request Header Fields Too Large

Servers are not required to use the 431 status code; when under attack, it may be more appropriate to just drop connections, or take other steps.

7.4. 511 Network Authentication Required

In common use, a response carrying the 511 status code will not come from the origin server indicated in the request's URL. This presents many security issues; e.g., an attacking intermediary may be inserting cookies into the original domain's name space, may be observing cookies or HTTP authentication credentials sent from the user agent, and so on.

However, these risks are not unique to the 511 status code; in other words, a captive portal that is not using this status code introduces the same issues.

Also, note that captive portals using this status code on an SSL or TLS connection (commonly, port 443) will generate a certificate error on the client.

8. IANA Considerations

The HTTP Status Codes Registry should be updated with the following entries:

- Code: 428
- Description: Precondition Required
- Specification: [this document]

- Code: 429
- Description: Too Many Requests
- Specification: [this document]

- Code: 431
- Description: Request Header Fields Too Large
- Specification: [this document]

- Code: 511
- Description: Network Authentication Required
- Specification: [this document]

9. References

9.1. Normative References

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

[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.

9.2. Informative References

[RFC4791] Daboo, C., Desruisseaux, B., and L. Dusseault, "Calendaring Extensions to WebDAV (CalDAV)", RFC 4791, March 2007.

[RFC4918] Dusseault, L., "HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)", RFC 4918, June 2007.

URIs

[1] <mailto:ietf-http-wg@w3.org>

[2] <mailto:ietf-http-wg-request@w3.org?subject=subscribe>

Appendix A. Acknowledgements

Thanks to Jan Algermissen and Julian Reschke for their suggestions and feedback.

Appendix B. Issues Raised by Captive Portals

Since clients cannot differentiate between a portal's response and that of the HTTP server that they intended to communicate with, a number of issues arise. The 511 status code is intended to help mitigate some of them.

One example is the "favicon.ico" <http://en.wikipedia.org/wiki/Favicon> commonly used by browsers to identify the site being accessed. If the favicon for a given site is fetched from a captive portal instead of the intended site (e.g., because the user is unauthenticated), it will often "stick" in the browser's cache (most implementations cache favicons aggressively) beyond the portal session, so that it seems as if the portal's favicon has "taken over" the legitimate site.

Another browser-based issue comes about when P3P <http://www.w3.org/TR/P3P/> is supported. Depending on how it is implemented, it's possible a browser might interpret a portal's response for the p3p.xml file as the server's, resulting in the privacy policy (or lack thereof) advertised by the portal being interpreted as applying to the intended site. Other Web-based protocols such as WebFinger <http://code.google.com/p/webfinger/wiki/WebFingerProtocol>, CORS <http://www.w3.org/TR/cors/> and OAuth <http://tools.ietf.org/html/draft-ietf-oauth-v2> may also be vulnerable to such issues.

Although HTTP is most widely used with Web browsers, a growing number of non-browsing applications use it as a substrate protocol. For example, WebDAV [RFC4918] and CalDAV [RFC4791] both use HTTP as the basis (for remote authoring and calendaring, respectively). Using these applications from behind a captive portal can result in spurious errors being presented to the user, and might result in content corruption, in extreme cases.

Similarly, other non-browser applications using HTTP can be affected as well; e.g., widgets <http://www.w3.org/TR/widgets/>, software updates, and other specialised software such as Twitter clients and the iTunes Music Store.

It should be noted that it's sometimes believed that using HTTP redirection to direct traffic to the portal addresses these issues. However, since many of these uses "follow" redirects, this is not a good solution.

Authors' Addresses

Mark Nottingham
Rackspace

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

Roy T. Fielding
Adobe Systems Incorporated
345 Park Ave
San Jose, CA 95110
USA

Email: fielding@gbiv.com
URI: http://roy.gbiv.com/

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: June 6, 2012

P. Bryan, Ed.
ForgeRock US, Inc.

December 4, 2011

JSON Patch
draft-pbryan-json-patch-04

Abstract

JSON Patch defines the media type "application/json-patch", a JSON document structure for expressing a sequence of operations to apply to a JSON document.

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 6, 2012.

Table of Contents

1. Introduction

JavaScript Object Notation (JSON) [RFC4627] is a common format for the exchange and storage of structured data. HTTP PATCH [RFC5789] extends HTTP [RFC2616] with a method to perform partial modifications to resources.

The JSON Patch media type "application/json-patch" is a JSON document structure for expressing a sequence of operations to apply to a target JSON document, suitable for use with the HTTP PATCH method.

2. Conventions

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].

3. Document Structure

A JSON Patch document contains a JSON array of objects. Each object contains a single operation to apply to the target JSON document.

An example JSON Patch document:

```
[
    { "test": "/a/b/c", value: "foo" },
    { "remove": "/a/b/c" },
    { "add": "/a/b/c", "value": [ "foo", "bar" ] },
    { "replace": "/a/b/c", "value": 42 },
    { "move": "/a/b/c", to: "/a/b/d" }
]
```

Evaluation of a JSON Patch document begins with a target JSON document. Operations are applied sequentially in the order they appear in the array. Each operation in the sequence is applied to the target document; the resulting document becomes the target of the next operation. Evaluation continues until all operations are successfully applied or an error condition is encountered.

4. Operations

The operation to apply is expressed in a member of the operation object. The name of the operation member is one of: "add", "remove", "replace", "move" or "test". The member value is a string containing a [JSON Pointer], which references the location in the target

document to apply the operation. It is an error condition if an operation object contains no operation member, or more than one operation member.

4.1. add

The "add" operation adds a new value to the target document at the specified location. The location must reference one of: the root of the target document, a member to add to an existing object, or an element to add to an existing array. The operation object contains a "value" member, which specifies the value to be added.

Example:

```
{ "add": "/a/b/c", "value": [ "foo", "bar" ] }
```

If the location references the root of the target document or a member of an existing object, it is an error condition if a value at the specified location already exists.

If the location references an element of an existing array, any elements at or above the specified index are shifted one position to the right. It is an error condition if the specified index is greater than the number of elements in the array.

4.2. remove

The "remove" operation removes the value in the target document at the specified location.

Example:

```
{ "remove": "/a/b/c" }
```

If removing an element from an array, any elements above the specified index are shifted one position to the left.

It is an error condition if a value at the specified location does not exist.

4.3. replace

The "replace" operation replaces the value in the target document at the specified location with a new value. The operation object contains a "value" member, which specifies the replacement value.

Example:

```
{ "replace": "/a/b/c", "value": 42 }
```

This operation is identical to expressing a "remove" operation for a value, followed immediately by an "add" operation at the same location with the replacement value.

It is an error condition if a value at the specified location does not exist.

4.4. move

The "move" operation removes the value at one location and adds it to another location in the target document.

The operation object contains a "to" member, a string containing a JSON Pointer which references the location in the target document to move the value to. This location must reference one of: the member to add to an existing object, or an element to add to an existing array.

Example:

```
{ "move": "/a/b/c", to: "/a/b/d" }
```

This operation is identical to expressing a "remove" operation, followed immediately by an "add" operation at the new location with the value that was just removed.

If the location in the "to" member references a member of an existing object in the target document, it is an error condition if a value at the specified location already exists.

If the location in the "to" member references an element of an existing array, any elements at or above the specified index are shifted one position to the right. It is an error condition if the specified index is greater than the number of elements in the array.

4.5. test

The "test" operation tests that a value in the target document at the specified location is equal to a specified value. The operation object contains a "value" member, which specifies the value to test for.

Example:

```
{ "test": "/a/b/c", value: "foo" }
```

It is an error condition if the value in the target document is not equal to the specified value.

5. Error Handling

If an error condition occurs, evaluation of the JSON Patch document SHOULD terminate and application of the entire patch document SHALL NOT be deemed successful.

6. IANA Considerations

The Internet media type for a JSON Patch document is application/json-patch.

Type name: application

Subtype name: json-patch

Required parameters: none

Optional parameters: none

Encoding considerations:
Per JSON [RFC4627]: 8bit if UTF-8; binary if UTF-16 or UTF-32.

Security considerations:
See Security Considerations in section 7.

Interoperability considerations: N/A

Published specification:
draft-pbryan-json-patch-04

Applications that use this media type:
Applications that manipulate JSON documents.

Additional information:

Magic number(s): N/A

File extension(s): .json-patch

Macintosh file type code(s): TEXT

Person & email address to contact for further information:
Paul C. Bryan <paul.bryan@forgerock.com>

Intended usage: COMMON

Restrictions on usage: none

Author: Paul C. Bryan <paul.bryan@forgerock.com>

Change controller: Paul C. Bryan <paul.bryan@forgerock.com>

7. Security Considerations

This specification has the same security considerations as JSON [RFC4627] and JSON Pointer [JSON Pointer].

8. Acknowledgements

The following individuals contributed ideas, feedback and wording, which contributed to the content of this specification:

Mike Amundsen, Paul Davis, Dean Landolt, Randall Leeds, Mark Nottingham, Julian Reschke, Eli Stevens.

The structure of a JSON Patch document was initially informed by the XML Patch document [RFC5261] specification.

9. References

9.1. Normative References

[JSON Pointer] Bryan, P. and K. Zyp, "JSON Pointer", October 2011, <http://tools.ietf.org/html/draft-pbryan-zyp-json-pointer-02>.

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

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

9.2. Informative References

[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.

[RFC5261] Urpalainen, J., "An Extensible Markup Language (XML) Patch Operations Framework Utilizing XML Path Language (XPath) Selectors", RFC 5261, September 2008.

[RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, March 2010.

Appendix A. Examples

A.1. Adding an Object Member

An example target JSON document:

```
{
    "foo": "bar"
}
```

A JSON Patch document:

```
[
    { "add": "/baz", "value": "qux" }
]
```

The resulting JSON document:

```
{
    "baz": "qux",
    "foo": "bar"
}
```

A.2. Adding an Array Element

An example target JSON document:

```
{
    "foo": [ "bar", "baz" ]
}
```

A JSON Patch document:

```
[
    { "add": "/foo/1", "value": "qux" }
]
```

The resulting JSON document:

```
{
    "foo": [ "bar", "qux", "baz" ]
}
```

A.3. Removing an Object Member

An example target JSON document:

```
{
    "baz": "qux",
    "foo": "bar"
}
```

A JSON Patch document:

```
[
    { "remove": "/baz" }
]
```

The resulting JSON document:

```
{
    "foo": "bar"
}
```

A.4. Removing an Array Element

An example target JSON document:

```
{
    "foo": [ "bar", "qux", "baz" ]
}
```

A JSON Patch document:

```
[
    { "remove": "/foo/1" }
]
```

The resulting JSON document:

```
{
    "foo": ["bar", "baz"]
}
```

A.5. Replacing a Value

An example target JSON document:

```
{
    "baz": "qux",
    "foo": "bar"
}
```

A JSON Patch document:

```
[
      { "replace": "/baz", "value": "boo" }
]
```

The resulting JSON document:

```
{
    "baz": "boo",
    "foo": "bar"
}
```

A.6. Moving a Value

An example target JSON document:

```
{
    "foo": {
        "bar": "baz",
        "waldo": "fred"
    }
    "qux": {
        "corge": "grault"
    }
}
```

A JSON Patch document:

```
[
    { "move": "/foo/waldo", to: "/qux/thud" }
]
```

The resulting JSON document:

```
{
    "foo": {
        "bar": "baz"
    }
    "qux": {
        "corge": "grault",
        "thud": "fred"
    }
}
```

A.7. Moving an Array Element

An example target JSON document:

```
{
    "foo": [ "all", "grass", "cows", "eat" ]
}
```

A JSON Patch document:

```
[
    { "move": "/foo/1", "to": "/foo/3" }
]
```

The resulting JSON document:

```
{
    "foo": [ "all", "cows", "eat", "grass" ]
}
```

A.8. Testing a Value: Success

An example target JSON document:

```
{
    "baz": "qux",
    "foo": [ "a", 2, "c" ]
}
```

A JSON Patch document, which will result in successful evaluation:

```
[
    { "test": "/baz", "value": "qux" },
    { "test": "/foo/1", "value": 2 }
]
```

A.9. Testing a Value: Error

An example target JSON document:

```
{
    "baz": "qux",
}
```

A JSON Patch document, which will result in an error condition:

```
[
    { "test": "/baz", "value": "bar" }
]
```

Author's Address

Paul C. Bryan (editor)
ForgeRock US, Inc.
201 NE Park Plaza Drive Suite 196
Vancouver, WA 98684
USA

Phone: +1 604 783 1481
Email: paul.bryan@forgerock.com