Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 27, 2008

J. Gregorio, Ed.
Google
M. Hadley, Ed.
Sun Microsystems
M. Nottingham, Ed.

D. Orchard
BEA Systems, Inc.
Mar 26, 2008

URI Template
draft-gregorio-uritemplate-03

Status of this Memo

Abstract

A URI Template is a compact sequence of characters used for the construction of URIs. This specification defines the URI Template syntax and the process for expanding a URI Template into a URI, along with guidelines and security considerations for the use of URI Templates on the Internet. The URI Template syntax allows for the construction of strings that are a superset of URIs, allowing an implementation to process any URI Template without knowing the

scheme-specific requirements of every possible resulting URI.

Editorial Note

To provide feedback on this Internet-Draft, join the W3C URI mailing list (http://lists.w3.org/Archives/Public/uri/) [1].

Table of Contents

1. Introduction

A URI Template provides a simple and extensible format for URI construction. A URI Template is a string that contains embedded expansions, text marked off in matching braces ('{', '}'), that denotes a part of the string that is to be substituted by a template processor to produce a URI. A URI Template is transformed into a URI by substituting the expansions with their calculated value.

Several specifications have defined URI Templates with varying levels of formality, such as WSDL, WADL and OpenSearch. This specification is derived from these concepts, giving a rigorous definition to such templates.

This specification uses the terms "character" and "coded character set" in accordance with the definitions provided in [RFC2978], and "character encoding" in place of what [RFC2978] refers to as a "charset".

1.1. Overview

A URI Template allows a structural description of URIs while allowing a consumer of the template to construct a final URI by providing the values of the expansion variables. For example, given the following URI Template:

```
http://www.example.com/users/{userid}
```

And the following variable value

```
userid := "fred"
```

The expansion of the URI Template is:

```
http://www.example.com/users/fred
```

Here is an example that constructs a query from multiple variables:

```
http://www.example.com/?{-join|&|query,number}
```

And the following variables

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

The expansion of the URI Template is:

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

The template expansion describes in a machine readable manner how the URI is to be constructed.

```
  http://www.example.com/?{-join|&|query,number}
                          \____________________/
                                   |
                                   |
         Join 'var=value' for each variable
         in ['query', 'number'] with '&'.
```

1.2. Design Considerations

The URI Template syntax has been designed to carefully balance the need for a powerful substitution mechanism with ease of implementation and security. The syntax is designed to be easy to parse while at the same time providing enough flexibility to express many common templating scenarios.

Another consideration was to keep the syntax and processing in-line with the pre-existing templating schemes present in OpenSearch, WSDL and WADL.

The final design consideration was control over the placement of reserved characters in the URI generated from a URI Template. The reserved characters in a URI Template can only appear in the non-expansion text, or in the argument to an operator, both locations are dictated by the URI Template author. Given the percent-encoding rules for variable values this means that the source of all structure, i.e reserved characters, in a URI generated from a URI Template is decided by the URI Template author.

1.3. Applicability

While URI Templates use a notation that is similar to some URI path matching notations in web frameworks, URI Templates were not designed for that use case, nor are they appropriate for that purpose. URI Templates are not URIs, they do not identify an abstract or physical resource, they are not to be treated like URIs, nor should not be used in places where a URI would be expected.

1.4. Notational Conventions

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234], including the following core ABNF syntax rules defined by that specification: ALPHA (letters) and DIGIT (decimal digits). See [RFC3986] for the definitions of the URI-reference, percent-encoded, reserved, and unreserved rules.

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

2. Characters

A URI Template is a sequence of characters, and has the same issues as URIs with regard to codepoints and character sets. That is, URI Template characters are frequently encoded as octets for transport or presentation. This specification does not mandate any particular character encoding for mapping between URI characters and the octets used to store or transmit those characters. When a URI appears in a protocol element, the character encoding is defined by that protocol; without such a definition, a URI is assumed to be in the same character encoding as the surrounding text.

The ABNF notation defines its terminal values to be non-negative integers (codepoints) based on the US-ASCII coded character set [ASCII]. Because a URI is a sequence of characters, we must invert that relation in order to understand the URI syntax. Therefore, the integer values used by the ABNF must be mapped back to their corresponding characters via US-ASCII in order to complete the syntax rules.

3. Terminology

- o template processor - A program or library that converts a URI Template into a URI.
- o template expansion - The text between '{' and '}', including the enclosing brackets.

4. URI Template

A URI Template is a sequence of characters that contains any number of embedded template expansions, see Section 4.2. Each expansion references one or more variables whose values are used when determining the substition value for an expansion. A URI Template becomes a URI when all the template expansions are substituted with their values (see Section 4.4). The generated URI will be a URI-reference, i.e. either an absolute URI or a relative reference.

4.1. Variables

Every variable is either a Unicode string or a list of Unicode strings.

A template expansion MAY reference variables that are unknown to the template processor. Those variables are 'undefined' and template expansion takes into consideration 'undefined' variables. Conversely, every variable that he template processor knows about is considered 'defined'.

A variable that contains a string of length zero MUST NOT be considered 'undefined' by the template processor. A list variable that contains no members, that is of zero length, MUST NOT be considered 'undefined' by the template processor.

Beyond the scope of this specification is the allowable programming constructs that can be used for a list variable. For example, a Python implementation might allow only built-in list types, or it may allow any iterable to be used as the source for a list variable.

Some variables may be supplied with default values. The default value must comde from (unreserved / pct-encoded). Note that there is no notation for supplying default values to list variables.

A variable may appear in more than one expansion in a URI Template. The value used for that variable MUST remain the same for every template expansion when converting a URI Template into a URI.

4.2. Template Expansions

Template expansions are the parameterized components of a URI Template. A template expansion MUST match the 'expansion' rule.

```
op          = 1*ALPHA
arg         = *(reserved / unreserved / pct-encoded)
var         = varname [ "=" vardefault ]
vars        = var [ *("," var) ]
varname     = (ALPHA / DIGIT)*(ALPHA / DIGIT / "." / "_" / "-" )
vardefault  = *(unreserved / pct-encoded)
operator    = "-" op "|" arg "|" vars
expansion   = "{" ( var / operator ) "}"
```

4.3. Error Handling

During template substitution error conditions may arise. The exact circumstances for those errors are described in Section 4.4. When an error occurs the template processor MUST NOT return a URI. It is language specific and beyond the scope of this document how the template processor signals that an error has occured and that a URI will not be generated from the template.

4.4. URI Template Substitution

Template substitution is the process of turning a URI Template into a URI given definitions for the variables used in the template. Substitution replaces each expansion with its calculated value. A template processor take two inputs, a URI Template and a set of variables, and returns a URI-reference.

Before substitution the template processor MUST convert every variable value into a sequence of characters in (unreserved / pct-encoded). The template processor does that using the following algorithm: The template processor normalizes the string using NFKC, converts it to UTF-8 [RFC3629], and then every octet of the UTF-8 string that falls outside of (unreserved) MUST be percent-encoded, as per [RFC3986], section 2.1. For variables that are lists, the above algorithm is applied to each value in the list.

The Unicode Standard [UNIV4] defines various equivalences between sequences of characters for various purposes. Unicode Standard Annex #15 [UTR15] defines various Normalization Forms for these equivalences, in particular Normalization Form KC (NFKC, Compatibility Decomposition, followed by Canonical Composition). Since different Normalized Forms unicode strings will have different UTF-8 represenations the only way to guarantee that template processors will produce the same URI is to require a common Normalized Form.

Requiring that all characters outside of (unreserved) be percent encoded means that the only characters outside of (unreserved) that will appear in the generated URI-reference will come from outside the template expansions in the URI Template or from the argument of a template expansion. This means that the designer of the URI Template determines the placement of reserved characters in the resulting URI, and thus the structure of the resulting generated URI-reference.

If the expansion is an operator then the substitution value is determined by the given operator. Each operator works only on the variables that are defined within their expansion.

The result of substitution MUST match the URI-reference rule and SHOULD also match any known rules for the scheme of the resulting URI.

If a template processor encounters an operator that it does not understand then it MUST fail and MUST NOT produce a URI from the URI Template. The list of operators that a template processor knows is not constrained by this specification, that is, later specifications may add new operators.

Every expansion consists of either a variable ('var') or an operator expression ('operator'), and the rules for how to expand each of these is given below. For every expansion a template MUST have at least one variable name in the template expansion. It is an error if no variables are supplied. All of the variables supplied to a template expansion MAY be undefined and the expansion rules below specify how to process the template expansion in that situation.

4.4.1. ('var') substitution

In a variable ('var') expansion, if the variable is defined then substitute the value of the variable, otherwise substitute the default value. If no default value is given then substitute with the empty string.

Example:

```
foo := "fred"

"{foo}"        -> "fred"
"{bar=wilma}"  -> "wilma"
"{baz}"        -> ""
```

4.4.2. The 'opt' operator

If each variable is undefined or an empty list then substitute the empty string, otherwise substitute the value of 'arg'.

Example:

```
 foo := "fred"

 "{-opt|fred@example.org|foo}" -> "fred@example.org"
 "{-opt|fred@example.org|bar}" -> ""
```

4.4.3. The 'neg' operator

If each variable is undefined or an empty list then substitute the value of arg, otherwise substitute the empty string.

Example:

```
 foo := "fred"

 "{-neg|fred@example.org|foo}" -> ""
 "{-neg|fred@example.org|bar}" -> "fred@example.org"
```

4.4.4. The 'prefix' operator

The prefix operator MUST only have one variable in its expansion. More than one variable is an error condition. If the variable is undefined or an empty list then substitute the empty string. If the variable is a defined non-list then substitute the value of arg preceeded by the value of the variable. If the variable is a defined list then substitute the concatenation of every list value preceeded by the arg.

Example:

```
foo := "fred"
bar := ["fee", "fi", "fo", "fum"]
baz := []

"{-prefix|/|foo}" -> "/fred"
"{-prefix|/|bar}" -> "/fee/fi/fo/fum"
"{-prefix|/|baz}" -> ""
"{-prefix|/|qux}" -> ""
```

4.4.5. The 'suffix' operator

The prefix operator MUST only have one variable in its expansion. More than one variable is an error condition. If the variable is undefined or an empty list then substitute the empty string. If the variable is a defined non-list then substitute the value of arg followed by the value of the variable. If the variable is a defined list then substitute the concatenation of every list value followed by the arg.

Example:

```
foo := "fred"
bar := ["fee", "fi", "fo", "fum"]
baz := []

"{-suffix|/|foo}" -> "fred/"
"{-suffix|/|bar}" -> "fee/fi/fo/fum/"
"{-suffix|/|baz}" -> ""
"{-suffix|/|qux}" -> ""
```

4.4.6. The 'join' operator

Supplying a list variable to the join operator is an error. For each variable that is defined and non-empty create a keyvalue string that is the concatenation of the variable name, "=", and the variable value. Concatenate more than one keyvalue string with intervening

values of arg to create the substitution value. The order of variables MUST be preserved during substitution.

Example:

```
foo := "fred"
bar := "barney"
baz := ""

"{-join|&|foo,bar,baz,qux}" -> "foo=fred&bar=barney&baz="
"{-join|&|bar}" -> "bar=barney"
"{-join|&|qux}" -> ""
```

4.4.7. The 'list' operator

The listjoin operator MUST have only one variable in its expansion and that variable must be a list. More than one variable is an error. If the list is non-empty then substitute the concatenation of all the list members with intervening values of arg. If the list is empty or the variable is undefined them substitute the empty string.

Example:

```
foo := ["fred", "barney", "wilma"]
bar := ["a", "", "c"]
baz := ["betty"]
qux := []

"{-list|/|foo}" -> "fred/barney/wilma"
"{-list|/|bar}" -> "a//c"
"{-list|/|baz}" -> "betty"
"{-list|/|qux}" -> ""
"{-list|/|corge}" -> ""
```

4.5. Examples

Given the following template variable names and values:

Name	Value
foo	\u03d3
bar	fred
baz	10,20,30
qux	["10","20","30"]
corge	[]
grault	
garply	a/b/c
waldo	ben & jerrys
fred	["fred", "", "wilma"]
plugh	["\u017F\u0307", "\u0073\u0307"]
1-a_b.c	200

Table 1

The variable 'foo' is the unicode character GREEK UPSILON WITH ACUTE AND HOOK SYMBOL. This character was chosen because it is one of only three characters that has a different normal form for each of the four normalization forms (NFC, NFD, NFKC, NFKD). The name 'xyzzy' has not been defined, the value of 'grault' is the empty string. The variables qux, corge, fred, and plugh are lists.

The following URI Templates will be expanded as shown:

```
http://example.org/?q={bar}
http://example.org/?q=fred

/{xyzzy}
/

http://example.org/?{-join|&|foo,bar,xyzzy,baz}
http://example.org/?foo=%CE%8E&bar=fred&baz=10%2C20%2C30

http://example.org/?d={-list|,|qux}
http://example.org/?d=10,20,30

http://example.org/?d={-list|&d=|qux}
http://example.org/?d=10&d=20&d=30

http://example.org/{bar}{bar}/{garply}
http://example.org/fredfred/a%2Fb%2Fc

http://example.org/{bar}{-prefix|/|fred}
http://example.org/fred/fred//wilma

{-neg|:|corge}{-suffix|:|plugh}
:%E1%B9%A1:%E1%B9%A1:

../{waldo}/
../ben%20%26%20jerrys/

telnet:192.0.2.16{-opt|:80|grault}
telnet:192.0.2.16:80

 :{1-a_b.c}:
 :200:

----
```

5. Security Considerations

A URI Template does not contain active or executable content. Other security considerations are the same as those for URIs, see section 7 of RFC3986.

6. IANA Considerations

In common with RFC3986, URI scheme names form a registered namespace that is managed by IANA according to the procedures defined in [RFC4395]. No IANA actions are required by this document.

7. Appendix A - Parsing URI Template Expansions

Parsing a valid URI Template expansion does not require building a parser from the given ABNF. Instead, the set of allowed characters in each part of URI Template expansion has been chosen to avoid complex parsing, and breaking an expansion into its component parts can be achieved by a series of splits of the character string.

Here is example Python code that parses a URI Template expansion and returns the operator, argument, and variables as a tuple. The variables are returned as a dictionary of variable names mapped to their default values. If no default is given then the name maps to None.

```
 def parse_expansion(expansion):
    if "|" in expansion:
        (op, arg, vars_) = expansion.split("|")
        op = op[1:]
    else:
        (op, arg, vars_) = (None, None, expansion)
    vars_ = vars_.split(",")

    variables = {}
    for var in vars_:
        if "=" in var:
            (varname, vardefault) = var.split("=")
        else:
            (varname, vardefault) = (var, None)
        variables[varname] = vardefault

    return (op, arg, variables)
```

And here is an example of the parse_expansion() function being used.

```
>>> parse_expansion("-join|&|a,b,c=1")
('join', '&', {'a': None, 'c': '1', 'b': None})
>>> parse_expansion("c=1")
(None, None, {'c': '1'})
```

8. 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.

[RFC2978] Freed, N. and J. Postel, "IANA Charset Registration Procedures", BCP 19, RFC 2978, October 2000.

[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.

[RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 115, RFC 4395, February 2006.

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

[UNIV4] The Unicode Consortium, "The Unicode Standard, Version 4.0.1, defined by: The Unicode Standard, Version 4.0 (Reading, MA, Addison-Wesley, 2003. ISBN 0-321-18578-1), as amended by Unicode 4.0.1 (http://www.unicode.org/versions/Unicode4.0.1/)", March 2004.

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

[1] <http://lists.w3.org/Archives/Public/uri/>

Appendix A. Contributors

The following people made significant contributions to this specification: Michaeljohn Clement, DeWitt Clinton, John Cowan, James H. Manger, and James Snell.

Appendix B. Revision History

03 - Added more examples. Introduced error conditions and defined their handling. Changed listjoin to list. Changed -append to -suffix, and allowed -prefix and -suffix to accept list variables. Clarified the handling of unicode.

02 - Added operators and came up with coherent percent-encoding and reserved character story. Added large examples section which is extracted and tested against the implementation.

01

00 - Initial Revision.

Authors' Addresses

Joe Gregorio (editor)
Google

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

Marc Hadley (editor)
Sun Microsystems

Email: Marc.Hadley@sun.com
URI: http://sun.com/

Mark Nottingham (editor)

Email: mnot@pobox.com
URI: http://mnot.net/

David Orchard
BEA Systems, Inc.

Email: dorchard@bea.com
URI: http://bea.com/

Full Copyright Statement

Intellectual Property