Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 29, 2008

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

D. Orchard
BEA Systems, Inc.
Nov 26, 2007

URI Template
draft-gregorio-uritemplate-02

Status of this Memo

Copyright Notice

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 3
1.1. Overview 3
1.2. Design Considerations 4
1.3. Notational Conventions 4
2. Characters 4
3. URI Template 5
3.1. Variables 5
3.2. Template Expansions 6
3.3. URI Template Substitution 6
3.3.1. The 'opt' operator 7
3.3.2. The 'neg' operator 7
3.3.3. The 'prefix' operator 7
3.3.4. The 'append' operator 7
3.3.5. The 'join' operator 7
3.3.6. The 'listjoin' operator 7
3.4. Examples 8
4. Security Considerations 10
5. IANA Considerations 10
6. Appendix A - Parsing URI Template Expansions 10
7. Normative References 11
Appendix A. Contributors 11
Appendix B. Revision History 12
Authors' Addresses 12
Intellectual Property and Copyright Statements 13

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

URI Templates can be used as a machine-readable forms language. By allowing clients to form their own identifiers based on templates given to them by the URI's authority it's possible to construct dynamic systems that use more of the URI than traditional HTML forms. For example:

    http://www.example.org/products/{upc}/buyers?page={page_num}

URI Templates can also be used to compose URI-centric protocols without impinging on authorities' control of their URI space. For example, there are many emerging conventions for passing around login information between sites using URIs. Forcing people to use a well-

known query parameter isn't good practice, but using URI Templates allows different sites to specify local ways of conveying the same information:

http://auth.example.com/userauth;{return-uri}

http://login.example.org/login?back={return-uri}

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. On the balance, the template processing is not Turing complete, thus avoiding a number of security issues, ala the billion-laughs attack of XML DTDs.

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. Notational Conventions

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC4234], 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. URI Template

A URI Template is a sequence of characters that contains one or more embedded template expansions, see Section 3.2. Each expansion references one or more variables whose values are used in when determining the substition value for an expansion. A URI Template becomes a URI when the template expansions are substituted with their values (see Section 3.3). The generated URI will be a URI-reference, i.e. either an absolute URI or a relative reference.

3.1. Variables

The value of every non-list variable, and the individual values in list variables, must come from (unreserved / pct-encoded). For variable values that are strings that have characters outside that range, the entire string must be converted into UTF-8 [RFC3629], and then every octet of the UTF-8 string that falls outside of (unreserved / pct-encoded) MUST be percent-encoded, as per [RFC3986], section 2.1.

This does not imply that every variable value can be decoded into a Unicode string. For example, a variable value may be a binary blob that has been percent-encoded before being passed into the template processor.

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 C (NFC, Canonical Decomposition, followed by Canonical Composition) and Normalization Form KC (NFKC, Compatibility Decomposition, followed by Canonical

Composition). Since different Normalized Forms unicode strings will have different UTF-8 represenations it is RECOMMEDED that unicode strings use Normalized Form NFC.

The meaning of 'defined' for a variable is progamming language and library specific and beyond the scope of this specification. Also beyond the scope of this specification is the allowable programming constructs that can be used for a list variable used in the 'listjoin' operator. 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.

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.

3.2. 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 ) "}"
```

3.3. 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.

Every expansion consists of either a variable ('var') or an operator expression. In a variable ('var') expansion, if the variable is defined and non-empty then substitute the value of the variable, otherwise substitute the default value. If no default value is given then substitute with the empty string.

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.

3.3.1. The 'opt' operator

If the one or more of the variables are defined and non-empty then substitute the value of 'arg', otherwise substitute the empty string.

3.3.2. The 'neg' operator

If all of the variables are un-defined or empty then substitute the value of arg, otherwise substitute the empty string.

3.3.3. The 'prefix' operator

The prefix operator MUST only have one variable in its expansion. If the variable is defined and non-empty then substitute the value of arg followed by the value of the variable, otherwise substitute the empty string.

3.3.4. The 'append' operator

The append operator MUST only have one variable in its expansion. If the variable is defined and non-empty then substitute the value of the variable followed by the value of arg, otherwise substitute the empty string.

3.3.5. The 'join' operator

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.

3.3.6. The 'listjoin' operator

The listjoin operator MUST have only one variable in its expansion and that variable must be a list. If the list is non-empty then substitute the concatenation of all the list members with intevening values of arg.

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

3.4. Examples

Given the following template variable names and values:

Name	Value
a	foo
b	bar
data	10,20,30
points	["10","20", "30"]
list0	[]
str0	
reserved	:/?#[]@!$&'()*+,;=
u	\u2654\u2655
a_b	baz

Table 1

The name 'foo' has not been defined, the value of 'str0' is the empty string, and both list0 and points are lists. The variable 'u' is a string of two unicode characters, the WHITE CHESS KING (0x2654) and the WHITE CHESS QUEEN (0x2655).

The following URI Templates will be expanded as shown:

```
    http://example.org/?q={a}
    http://example.org/?q=foo

    http://example.org/{foo}
    http://example.org/

    relative/{reserved}/
    relative/%3A%2F%3F%23%5B%5D%40%21%24%26%27%28%29%2A%2B%2C%3B%3D/

    http://example.org/{foo=fred}
    http://example.org/fred

    http://example.org/{foo=%25}/
    http://example.org/%25/

    /{-prefix|#|foo}
    /

    ./{-prefix|#|str0}
    ./

    /{-append|/|a}{-opt|data|points}{-neg|@|a}{-prefix|#|b}
    /foo/data#bar

    http://example.org/q={u}
    http://example.org/q=%E2%99%94%E2%99%95

    http://example.org/?{-join|&|a,data}
    http://example.org/?a=foo&data=10%2C20%2C30

    http://example.org/?d={-listjoin|,|points}&{-join|&|a,b}
    http://example.org/?d=10,20,30&a=foo&b=bar

    http://example.org/?d={-listjoin|,|list0}&{-join|&|foo}
    http://example.org/?d=&

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

    http://example.org/{a}{b}/{a_b}
    http://example.org/foobar/baz

    http://example.org/{a}{-prefix|/-/|a}/
    http://example.org/foo/-/foo/


   ----
```

4. 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.

5. 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.

6. 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'})
```

7. 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.

[RFC4234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005.

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

[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: DeWitt Clinton and James Snell.

Appendix B. Revision History

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

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).