Workgroup: Network Working Group
Internet-Draft: draft-ietf-jsonpath-iregexp-05
Published: 26 April 2023
Intended Status: Standards Track
Expires: 28 October 2023
Authors: C. Bormann
Universität Bremen TZI
T. Bray
Textuality

I-Regexp: An Interoperable Regexp Format

Abstract

This document specifies I-Regexp, a flavor of regular expressions that is limited in scope with the goal of interoperation across many different regular-expression libraries.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-jsonpath-iregexp/.

Discussion of this document takes place on the JSONPath Working Group mailing list (mailto:JSONPath@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/JSONPath/. Subscribe at https://www.ietf.org/mailman/listinfo/JSONPath/.

Source for this draft and an issue tracker can be found at https://github.com/ietf-wg-jsonpath/iregexp.

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 https://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 28 October 2023.

Copyright Notice

Table of Contents

1. Introduction

This specification describes an interoperable regular expression flavor, I-Regexp.

I-Regexp does not provide advanced regular expression features such as capture groups, lookahead, or backreferences. It supports only a Boolean matching capability, i.e., testing whether a given regular expression matches a given piece of text.

I-Regexp supports the entire repertoire of Unicode characters (Unicode scalar values).

I-Regexp is a subset of XSD regular expressions [XSD-2].

This document includes guidance for converting I-Regexps for use with several well-known regular expression idioms.

The development of I-Regexp was motivated by the work of the JSONPath Working Group. The Working Group wanted to include in its specification [I-D.ietf-jsonpath-base] support for the use of regular expressions in JSONPath filters, but was unable to find a useful specification for regular expressions which would be interoperable across the popular libraries.

1.1. Terminology

This document uses the abbreviation "regexp" for what are usually called regular expressions in programming. "I-Regexp" is used as a noun meaning a character string (sequence of Unicode scalar values) that conforms to the requirements in this specification; the plural is "I-Regexps".

This specification uses Unicode terminology. A good entry point into that is provided by [UNICODE-GLOSSARY].

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The grammatical rules in this document are to be interpreted as ABNF, as described in [RFC5234] and [RFC7405], where the "characters" of Section 2.3 of [RFC5234] are Unicode scalar values.

2. Requirements

I-Regexps should handle the vast majority of practical cases where a matching regexp is needed in a data model specification or a query language expression.

The editors of this document conducted a survey of the regexp syntax used in published RFCs. All examples found there should be covered by I-Regexps, both syntactically and with their intended semantics. The exception is the use of multi-character escapes, for which workaround guidance is provided in Section 5.

3. I-Regexp Syntax

An I-Regexp **MUST** conform to the ABNF specification in Figure 1.

```
i-regexp = branch *( "|" branch )
branch = *piece
piece = atom [ quantifier ]
quantifier = ( %x2A-2B ; '*'-'+'
 / "?" ) / ( "{" quantity "}" )
quantity = QuantExact [ "," [ QuantExact ] ]
QuantExact = 1*%x30-39 ; '0'-'9'

atom = NormalChar / charClass / ( "(" i-regexp ")" )
NormalChar = ( %x00-27 / %x2C-2D ; ','-'-'
 / %x2F-3E ; '/'-'>'
 / %x40-5A ; '@'-'Z'
 / %x5E-7A ; '^'-'z'
 / %x7E-10FFFF )
charClass = "." / SingleCharEsc / charClassEsc / charClassExpr
SingleCharEsc = "\" ( %x28-2B ; '('-'+'
 / %x2D-2E ; '-'-'.'
 / "?" / %x5B-5E ; '['-'^'
 / %s"n" / %s"r" / %s"t" / %x7B-7D ; '{'-'}'
 )
charClassEsc = catEsc / complEsc
charClassExpr = "[" [ "^" ] ( "-" / CCE1 ) *CCE1 [ "-" ] "]"
CCE1 = ( CCchar [ "-" CCchar ] ) / charClassEsc
CCchar = ( %x00-2C / %x2E-5A ; '.'-'Z'
 / %x5E-10FFFF ) / SingleCharEsc
catEsc = %s"\p{" charProp "}"
complEsc = %s"\P{" charProp "}"
charProp = IsCategory
IsCategory = Letters / Marks / Numbers / Punctuation / Separators /
    Symbols / Others
Letters = %s"L" [ ( %x6C-6D ; 'l'-'m'
 / %s"o" / %x74-75 ; 't'-'u'
 ) ]
Marks = %s"M" [ ( %s"c" / %s"e" / %s"n" ) ]
Numbers = %s"N" [ ( %s"d" / %s"l" / %s"o" ) ]
Punctuation = %s"P" [ ( %x63-66 ; 'c'-'f'
 / %s"i" / %s"o" / %s"s" ) ]
Separators = %s"Z" [ ( %s"l" / %s"p" / %s"s" ) ]
Symbols = %s"S" [ ( %s"c" / %s"k" / %s"m" / %s"o" ) ]
Others = %s"C" [ ( %s"c" / %s"f" / %x6E-6F ; 'n'-'o'
 ) ]
```

Figure 1: I-Regexp Syntax in ABNF

As an additional restriction, charClassExpr is not allowed to match [^], which according to this grammar would parse as a positive character class containing the single character ^.

This is essentially XSD regexp without character class subtraction, without multi-character escapes such as \s, \S, and \w, and without Unicode blocks.

An I-Regexp implementation **MUST** be a complete implementation of this limited subset. In particular, full support for the Unicode functionality defined in this specification is **REQUIRED**; the implementation **MUST NOT** limit itself to 7- or 8-bit character sets such as ASCII and **MUST** support the Unicode character property set in character classes.

3.1. Checking Implementations

A *checking* I-Regexp implementation is one that checks a supplied regexp for compliance with this specification and reports any problems. Checking implementations give their users confidence that they didn't accidentally insert non-interoperable syntax, so checking is **RECOMMENDED**. Exceptions to this rule may be made for low-effort implementations that map I-Regexp to another regexp library by simple steps such as performing the mapping operations discussed in Section 5; here, the effort needed to do full checking may dwarf the rest of the implementation effort. Implementations **SHOULD** document whether they are checking or not.

Specifications that employ I-Regexp may want to define in which cases their implementations can work with a non-checking I-Regexp implementation and when full checking is needed, possibly in the process of defining their own implementation classes.

4. I-Regexp Semantics

This syntax is a subset of that of [XSD-2]. Implementations which interpret I-Regexps **MUST** yield Boolean results as specified in [XSD-2]. (See also Section 5.2.)

5. Mapping I-Regexp to Regexp Dialects

The material in this section is non-normative, provided as guidance to developers who want to use I-Regexps in the context of other regular expression dialects.

5.1. Multi-Character Escapes

Common multi-character escapes (MCEs), and character classes built around them, which are not supported in I-Regexp, can usually be replaced as shown for example in Table 1.

MCE/class	Replace with
\S	[^ \t\n\r]
[\S]	[^\t\n\r]
\d	[0-9]

Table 1: Example substitutes for multi-character escapes

Note that the semantics of \d in XSD regular expressions is that of \p{Nd}; however, this would include all Unicode characters that are digits in various writing systems, which is almost certainly not what is required in IETF publications.

The construct \p{IsBasicLatin} is essentially a reference to legacy ASCII, it can be replaced by the character class [\u0000-\u007f].

5.2. XSD Regexps

Any I-Regexp also is an XSD Regexp [XSD-2], so the mapping is an identity function.

Note that a few errata for [XSD-2] have been fixed in [XSD11-2], which is therefore also included as a normative reference. XSD 1.1 is less widely implemented than XSD 1.0, and implementations of XSD 1.0 are likely to include these bugfixes, so for the intents and purposes of this specification an implementation of XSD 1.0 regexps is equivalent to an implementation of XSD 1.1 regexps.

5.3. ECMAScript Regexps

Perform the following steps on an I-Regexp to obtain an ECMAScript regexp [ECMA-262]:

- For any unescaped dots (.) outside character classes (first alternative of charClass production): replace dot by [^\n\r].

- Envelope the result in ^(?: and)$.

The ECMAScript regexp is to be interpreted as a Unicode pattern ("u" flag; see Section 21.2.2 "Pattern Semantics" of [ECMA-262]).

Note that where a regexp literal is required, the actual regexp needs to be enclosed in /.

5.4. PCRE, RE2, Ruby Regexps

Perform the same steps as in Section 5.3 to obtain a valid regexp in PCRE [PCRE2], the Go programming language [RE2], and the Ruby programming language, except that the last step is:

* Enclose the regexp in \A(?: and)\z.

6. Motivation and Background

While regular expressions originally were intended to describe a formal language to support a Boolean matching function, they have been enhanced with parsing functions that support the extraction and replacement of arbitrary portions of the matched text. With this accretion of features, parsing regexp libraries have become more susceptible to bugs and surprising performance degradations which can be exploited in Denial of Service attacks by an attacker who controls the regexp submitted for processing. I-Regexp is designed to offer interoperability, and to be less vulnerable to such attacks, with the trade-off that its only function is to offer a boolean response as to whether a character sequence is matched by a regexp.

6.1. Implementing I-Regexp

XSD regexps are relatively easy to implement or map to widely implemented parsing regexp dialects, with these notable exceptions:

* Character class subtraction. This is a very useful feature in many specifications, but it is unfortunately mostly absent from parsing regexp dialects. Thus, it is omitted from I-Regexp.

* Multi-character escapes. \d, \w, \s and their uppercase complement classes exhibit a large amount of variation between regexp flavors. Thus, they are omitted from I-Regexp.

* Not all regexp implementations support accesses to Unicode tables that enable executing constructs such as \p{Nd}, although the \p/\P feature in general is now quite widely available. While in principle it's possible to translate these into character-class matches, this also requires access to those tables. Thus, regexp libraries in severely constrained environments may not be able to support I-Regexp conformance.

7. IANA Considerations

This document makes no requests of IANA.

8. Security considerations

As discussed in Section 6, more complex regexp libraries may contain exploitable bugs leading to crashes and remote code execution. There is also the problem that such libraries often have hard-to-predict performance characteristics, leading to attacks that overload an implementation by matching against an expensive attacker-controlled regexp.

I-Regexps have been designed to allow implementation in a way that is resilient to both threats; this objective needs to be addressed throughout the implementation effort. Non-checking implementations (see Section 3.1) are likely to expose security limitations of any regexp engine they use, which may be less problematic if that engine has been built with security considerations in mind (e.g., [RE2]); a checking implementation is still **RECOMMENDED**.

9. References

9.1. Normative References

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

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

[RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF", RFC 7405, DOI 10.17487/RFC7405, December 2014, <https://www.rfc-editor.org/rfc/rfc7405>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[XSD-2] Malhotra, A., Ed. and P. V. Biron, Ed., "XML Schema Part 2: Datatypes Second Edition", W3C REC REC-xmlschema-2-20041028, W3C REC-xmlschema-2-20041028, 28 October 2004, <https://www.w3.org/TR/2004/REC-xmlschema-2-20041028/>.

[XSD11-2] Malhotra, A., Ed., Peterson, D., Ed., Thompson, H., Ed., Sperberg-McQueen, M., Ed., Biron, P. V., Ed., and S. Gao, Ed., "W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes", W3C REC REC-xmlschema11-2-20120405, W3C

REC-xmlschema11-2-20120405, 5 April 2012, <https://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/>.

9.2. Informative References

[ECMA-262] Ecma International, "ECMAScript 2020 Language Specification", ECMA Standard ECMA-262, 11th Edition, June 2020, <https://www.ecma-international.org/wp-content/uploads/ECMA-262.pdf>.

[I-D.ietf-jsonpath-base] Gössner, S., Normington, G., and C. Bormann, "JSONPath: Query expressions for JSON", Work in Progress, Internet-Draft, draft-ietf-jsonpath-base-13, 15 April 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-jsonpath-base-13>.

[PCRE2] "Perl-compatible Regular Expressions (revised API: PCRE2)", n.d., <http://pcre.org/current/doc/html/>.

[RE2] "RE2 is a fast, safe, thread-friendly alternative to backtracking regular expression engines like those used in PCRE, Perl, and Python. It is a C++ library.", n.d., <https://github.com/google/re2>.

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

[UNICODE-GLOSSARY] Unicode, Inc., "Glossary of Unicode Terms", <https://unicode.org/glossary/>.

Appendix A. Regexps and Similar Constructs in Recent Published RFCs

This section is to be removed before publishing as an RFC.

This appendix contains a number of regular expressions that have been extracted from some recently published RFCs based on some ad-hoc matching. Multi-line constructions were not included. With the exception of some (often surprisingly dubious) usage of multi-character escapes and a reference to the IsBasicLatin Unicode block, all regular expressions validate against the ABNF in Figure 1.

```
rfc6021.txt  459 (([0-1](\.[1-3]?[0-9]))|(2\.(0|([1-9]\d*))))
rfc6021.txt  513 \d*(\.\d*){1,127}
rfc6021.txt  529 \d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d+)?
rfc6021.txt  631 ([0-9a-fA-F]{2}(:[0-9a-fA-F]{2})*)?
rfc6021.txt  647 [0-9a-fA-F]{2}(:[0-9a-fA-F]{2}){5}
rfc6021.txt  933 ((:|[0-9a-fA-F]{0,4}):)([0-9a-fA-F]{0,4}:){0,5}
rfc6021.txt  938 (([^:]+:){6}(([^:]+:[^:]+)|(.*\..*)))|
rfc6021.txt 1026 ((:|[0-9a-fA-F]{0,4}):)([0-9a-fA-F]{0,4}:){0,5}
rfc6021.txt 1031 (([^:]+:){6}(([^:]+:[^:]+)|(.*\..*)))|
rfc6020.txt 6647 [0-9a-fA-F]*
rfc6095.txt 2544 \S(.*\S)?
rfc6110.txt 1583 [aeiouy]*
rfc6110.txt 3222 [A-Z][a-z]*
rfc6536.txt 1583 \*
rfc6536.txt 1632 [^\*].*
rfc6643.txt  524 \p{IsBasicLatin}{0,255}
rfc6728.txt 3480 \S+
rfc6728.txt 3500 \S(.*\S)?
rfc6991.txt  477 (([0-1](\.[1-3]?[0-9]))|(2\.(0|([1-9]\d*))))
rfc6991.txt  525 \d*(\.\d*){1,127}
rfc6991.txt  541 [a-zA-Z_][a-zA-Z0-9\-_.]*
rfc6991.txt  542 .|..|[^xX].*|.[^mM].*|..[^lL].*
rfc6991.txt  571 \d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d+)?
rfc6991.txt  665 ([0-9a-fA-F]{2}(:[0-9a-fA-F]{2})*)?
rfc6991.txt  693 [0-9a-fA-F]{2}(:[0-9a-fA-F]{2}){5}
rfc6991.txt  725 ([0-9a-fA-F]{2}(:[0-9a-fA-F]{2})*)?
rfc6991.txt  743 [0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-
rfc6991.txt 1041 ((:|[0-9a-fA-F]{0,4}):)([0-9a-fA-F]{0,4}:){0,5}
rfc6991.txt 1046 (([^:]+:){6}(([^:]+:[^:]+)|(.*\..*)))|
rfc6991.txt 1099 [0-9\.]*
rfc6991.txt 1109 [0-9a-fA-F:\.]*
rfc6991.txt 1164 ((:|[0-9a-fA-F]{0,4}):)([0-9a-fA-F]{0,4}:){0,5}
rfc6991.txt 1169 (([^:]+:){6}(([^:]+:[^:]+)|(.*\..*)))|
rfc7407.txt  933 ([0-9a-fA-F]){2}(:([0-9a-fA-F]){2}){0,254}
rfc7407.txt 1494 ([0-9a-fA-F]){2}(:([0-9a-fA-F]){2}){4,31}
rfc7758.txt  703 \d{2}:\d{2}:\d{2}(\.\d+)?
rfc7758.txt 1358 \d{2}:\d{2}:\d{2}(\.\d+)?
rfc7895.txt  349 \d{4}-\d{2}-\d{2}
rfc7950.txt 8323 [0-9a-fA-F]*
rfc7950.txt 8355 [a-zA-Z_][a-zA-Z0-9\-_.]*
rfc7950.txt 8356 [xX][mM][lL].*
rfc8040.txt 4713 \d{4}-\d{2}-\d{2}
rfc8049.txt 6704 [A-Z]{2}
rfc8194.txt  629 \*
rfc8194.txt  637 [0-9]{8}\.[0-9]{6}
rfc8194.txt  905 Z|[\+\-]\d{2}:\d{2}
rfc8194.txt  963 (2((2[4-9])|(3[0-9]))\.).*
rfc8194.txt  974 (([fF]{2}[0-9a-fA-F]{2}):).*
rfc8299.txt 7986 [A-Z]{2}
```

```
rfc8341.txt 1878 \*
rfc8341.txt 1927 [^\*].*
rfc8407.txt 1723 [0-9\.]*
rfc8407.txt 1749 [a-zA-Z_][a-zA-Z0-9\-_.]*
rfc8407.txt 1750 .|..|[^xX].*|.[^mM].*|..[^lL].*
rfc8525.txt  550 \d{4}-\d{2}-\d{2}
rfc8776.txt  838 /?([a-zA-Z0-9\-_.]+)(/[a-zA-Z0-9\-_.]+)*
rfc8776.txt  874 ([a-zA-Z0-9\-_.]+:)*
rfc8819.txt  311 [\S ]+
rfc8944.txt  596 [0-9a-fA-F]{2}(:[0-9a-fA-F]{2}){7}
```

Figure 2: Example regular expressions extracted from RFCs

Acknowledgements

This draft has been motivated by the discussion in the IETF JSONPATH WG about whether to include a regexp mechanism into the JSONPath query expression specification, as well as by previous discussions about the YANG pattern and CDDL .regexp features.

The basic approach for this draft was inspired by The I-JSON Message Format [RFC7493].

Authors' Addresses

Carsten Bormann
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany

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

Tim Bray
Textuality
Canada

Email: tbray@textuality.com