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/.

Source for this draft and an issue tracker can be found at https://github.com/cabo/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 30 October 2022.
1. Introduction

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

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 which conforms to the requirements in this specification; the plural is "I-Regexps".

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

I-Regexp supports the entire repertoire of Unicode characters.

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

This document includes rules for converting I-Regexps for use with
several well-known regexp libraries.

1.1. Terminology

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

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.

A brief survey of published RFCs yielded the regexp patterns in
Appendix A (with no attempt at completeness). With certain
exceptions as discussed there, these should be covered by I-Regexps,
both syntactically and with their intended semantics.

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
classExpr = "[" [ ^ ] ( '-' / 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 / IsBlock
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'
 ) ]
IsBlock = %s"Is" 1*( "-" / %x30-39 ; '0'-'9'
/ %x41-5A ; 'A'-Z'
/ %x61-7A ; 'a'-z'
)

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 and multi-character escapes such as `\s`, `\S`, and `\w`.

An I-Regexp implementation **MUST** be a complete implementation of this limited subset. In particular, full Unicode support 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.

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 5.1.)

5. Mapping I-Regexp to Regexp Dialects

(TBD; these mappings need to be further verified in implementation work.)

5.1. 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.

5.2. ECMAScript Regexps

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

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

* Envelope the result in `^` and `$`.

Note that where a regexp literal is required, the actual regexp needs to be enclosed in `/`. 
5.3. PCRE, RE2, Ruby Regexps

Perform the same steps as in Section 5.2 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 on constructs such as \p{IsCoptic}, although the \p/\P feature in general is now quite widely available. While in principle it's possible to translate these into codepoint-range 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-Regexp 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.

9. References

9.1. Normative References


9.2. Informative References

[ECMA-262]


Appendix A. Regexps and Similar Constructs in Recent Published RFCs

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, all regular expressions validate against the ABNF in Figure 1.
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].*|.[^1L].*
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

The multi-character escapes (MCE) or the character classes built around them used here can be substituted as shown in Table 1.

<table>
<thead>
<tr>
<th>MCE/class</th>
<th>Substitute class</th>
</tr>
</thead>
<tbody>
<tr>
<td>\s</td>
<td>[^\t\n\r]</td>
</tr>
<tr>
<td>[\s ]</td>
<td>[^\t\n\r]</td>
</tr>
<tr>
<td>\d</td>
<td>[0-9]</td>
</tr>
</tbody>
</table>

Table 1: Substitutes for multi-character escapes in examples

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 and certainly is not actually meant in the RFCs listed.

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 TZZI
Postfach 330440
D-28359 Bremen
Germany

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

Tim Bray
Textuality

Email: tbray@textuality.com