Stream: Internet Engineering Task Force (IETF)
RFC: 9485
Category: Standards Track
Published: October 2023
ISSN: 2070-1721
Authors: C. Bormann, *Universität Bremen TZI*
T. Bray, *Textuality*

RFC 9485
I-Regexp: An Interoperable Regular Expression Format

Abstract

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

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9485.

Copyright Notice

Table of Contents

1. Introduction

This specification describes an interoperable regular expression (abbreviated as "regexp") 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); both the I-Regexp strings themselves and the strings they are matched against are sequences of Unicode scalar values (often represented in UTF-8 encoding form [RFC3629] for interchange).

I-Regexp is a subset of XML Schema Definition (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 (WG). The WG wanted to include support for the use of regular expressions in JSONPath filters in its specification [JSONPATH-BASE], but was unable to find a useful specification for regular expressions that would be interoperable across the popular libraries.

1.1. Terminology

This document uses the abbreviation "regexp" for what is usually called a "regular expression" in programming. The term "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 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. Objectives

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.

At the time of writing, an editor of this document conducted a survey of the regexp syntax used in recently 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 = ( "*" / "+" / "?" ) / range-quantifier
range-quantifier = "{" QuantExact [ "," [ QuantExact ] ] "}"
QuantExact = 1*%x30-39 ; '0'-'9'

atom = NormalChar / charClass / ( "(" i-regexp ")" )
NormalChar = ( %x00-27 / "," / "-" / %x2F-3E ; '/'-'>'
 / %x40-5A ; '@'-'Z'
 / %x5E-7A ; '^'-'z'
 / %x7E-D7FF ; skip surrogate code points
 / %xE000-10FFFF )
charClass = "." / SingleCharEsc / charClassEsc / charClassExpr
SingleCharEsc = "\" ( %x28-2B ; '('-'+'
 / "-" / "." / "?" / %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-D7FF ; skip surrogate code points
 / %xE000-10FFFF ) / SingleCharEsc
catEsc = %s"\p{" charProp "}"
complEsc = %s"\P{" charProp "}"
charProp = IsCategory
IsCategory = Letters / Marks / Numbers / Punctuation / Separators /
    Symbols / Others
Letters = %s"L" [ ( %s"l" / %s"m" / %s"o" / %s"t" / %s"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" / %s"n" / %s"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 an XSD regexp without:

- character class subtraction,
- multi-character escapes such as `\s`, `\S`, and `\w`, and
- 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 syntax that is not interoperable, 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 might dwarf the rest of the implementation effort. Implementations **SHOULD** document whether or not they are checking.

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 that 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 not normative; it is provided as guidance to developers who want to use I-Regexps in the context of other regular expression dialects.

5.1. Multi-Character Escapes

I-Regexp does not support common multi-character escapes (MCEs) and character classes built around them. These can usually be replaced as shown by the examples 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 is also an XSD regexp [XSD-2], so the mapping is an identity function.

Note that a few errata for [XSD-2] have been fixed in [XSD-1.1-2]; therefore, it is also included in the Normative References (Section 9.1). XSD 1.1 is less widely implemented than XSD 1.0, and implementations of XSD 1.0 are likely to include these bugfixes; 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 the dot with `[^\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, and Ruby Regexps

To obtain a valid regexp in Perl Compatible Regular Expressions (PCRE) [PCRE2], the Go programming language's RE2 regexp library [RE2], and the Ruby programming language, perform the same steps as in Section 5.3, 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 that 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 access 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 is 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 has no IANA actions.

8. Security Considerations

While technically out of the scope of this specification, Section 10 ("Security Considerations") of [RFC3629] applies to implementations. Particular note needs to be taken of the last paragraph of Section 3 ("UTF-8 definition") of [RFC3629]; an I-Regexp implementation may need to mitigate limitations of the platform implementation in this regard.

As discussed in Section 6, more complex regexp libraries may contain exploitable bugs, which can lead to crashes and remote code execution. There is also the problem that such libraries often have performance characteristics that are hard to predict, 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]). In any case, a checking implementation is still **RECOMMENDED**.

Implementations that specifically implement the I-Regexp subset can, with care, be designed to generally run in linear time and space in the input and to detect when that would not be the case (see below).

Existing regexp engines should be able to easily handle most I-Regexps (after the adjustments discussed in Section 5), but may consume excessive resources for some types of I-Regexps or outright reject them because they cannot guarantee efficient execution. (Note that different versions of the same regexp library may be more or less vulnerable to excessive resource consumption for these cases.)

Specifically, range quantifiers (as in `a{2,4}`) provide particular challenges for both existing and I-Regexp focused implementations. Implementations may therefore limit range quantifiers in composability (disallowing nested range quantifiers such as `(a{2,4}){2,4}`) or range (disallowing very large ranges such as `a{20,200000}`), or detect and reject any excessive resource consumption caused by range quantifiers.

I-Regexp implementations that are used to evaluate regexps from untrusted sources need to be robust in these cases. Implementers using existing regexp libraries are encouraged:

- to check their documentation to see if mitigations are configurable, such as limits in resource consumption, and
- to document their own degree of robustness resulting from employing such mitigations.

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/info/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/info/rfc5234>.

[RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF", RFC 7405, DOI 10.17487/RFC7405, December 2014, <https://www.rfc-editor.org/info/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/info/rfc8174>.

[XSD-1.1-2] Peterson, D., Ed., Gao, S., Ed., Malhotra, A., Ed., Sperberg-McQueen, C. M., Ed., Thompson, H., Ed., and P. Biron, 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/>.

[XSD-2] Biron, P., Ed. and A. Malhotra, 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/>.

9.2. Informative References

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

[JSONPATH-BASE] Gössner, S., Ed., Normington, G., Ed., and C. Bormann, Ed., "JSONPath: Query expressions for JSON", Work in Progress, Internet-Draft, draft-ietf-jsonpath-base-20, 25 August 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-jsonpath-base-20>.

[PCRE2] "Perl-compatible Regular Expressions (revised API: PCRE2)", <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.", commit 73031bb, <https://github.com/google/re2>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

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

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

Acknowledgements

Discussion in the IETF JSONPATH WG about whether to include a regexp mechanism into the JSONPath query expression specification and previous discussions about the YANG `pattern` and Concise Data Definition Language (CDDL) `.regexp` features motivated this specification.

The basic approach for this specification 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