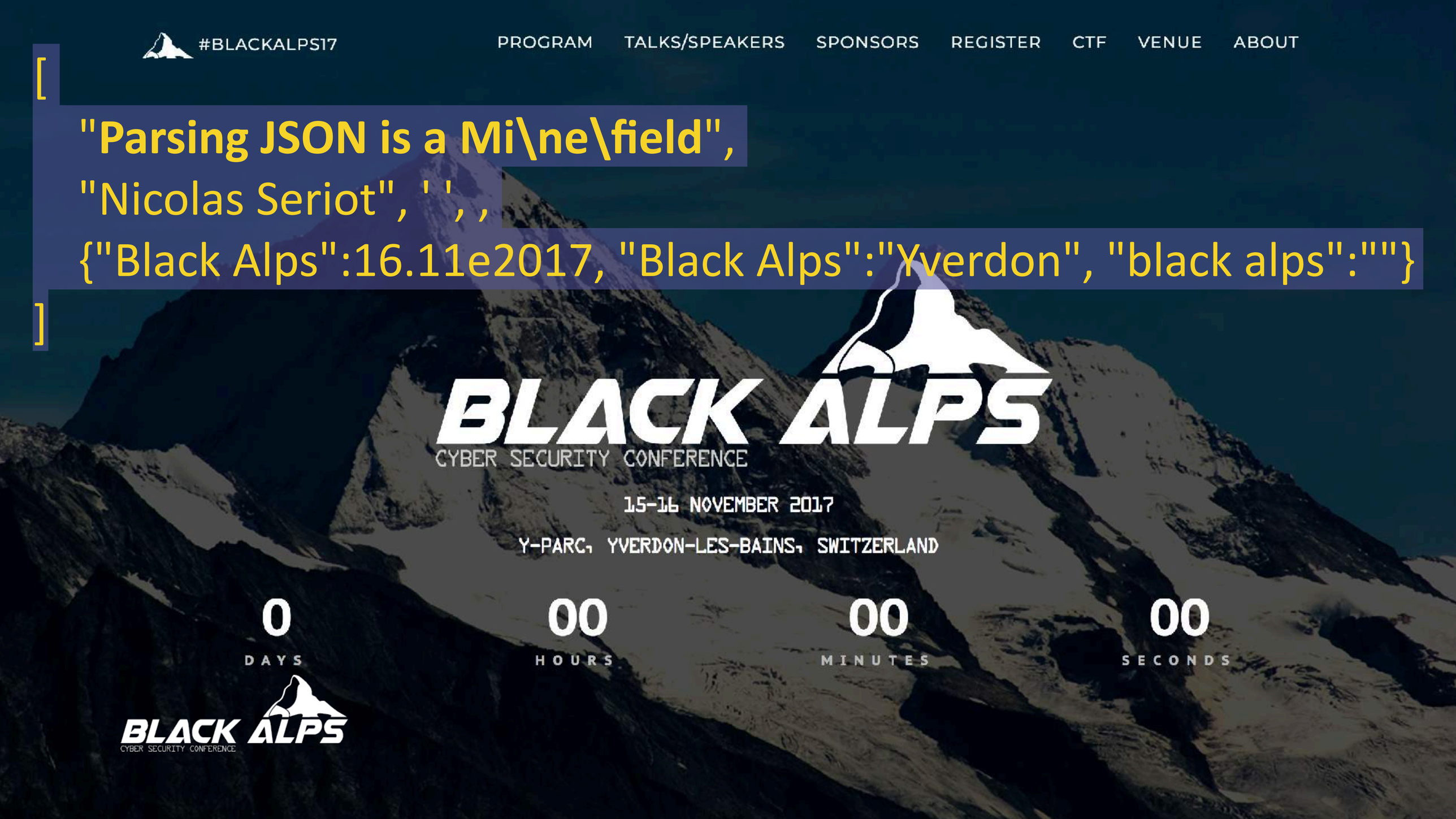
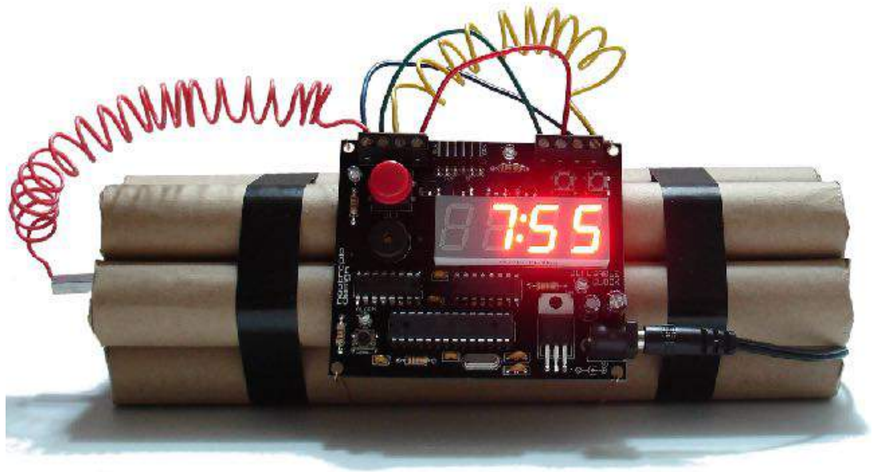

#BLACKALPS17
PROGRAM
TALKS/SPEAKERS
SPONSORS
REGISTER
CTF
VENUE
ABOUT
[
"Parsing JSON is a Mi\ne\field",
"Nicolas Seriot", '', ,
{"Black Alps":16.11e2017, "Black Alps":"Yverdon", "black alps":""}
]
BLACK ALPS
CYBER SECURITY CONFERENCE
15-16 NOVEMBER 2017
Y-PARC, YVERDON-LES-BAINS, SWITZERLAND
0
DAYS
00
HOURS
00
MINUTES
00
SECONDS
BLACK ALPS
CYBER SECURITY CONFERENCE

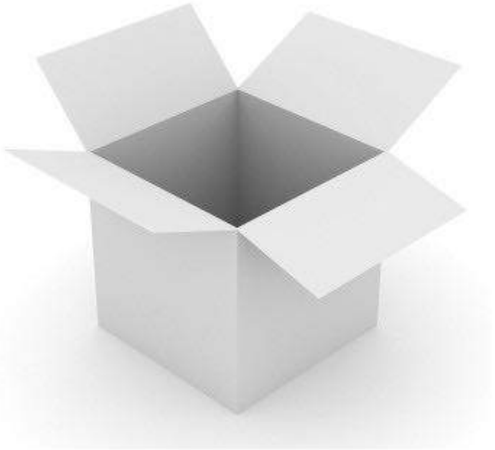
Finding Security Bugs

Just wait :)



White Box

code review, static analysis, ...

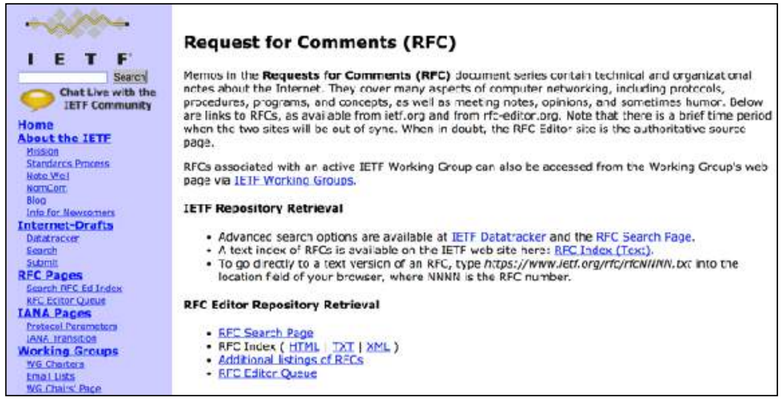


Black Box

fuzzing, …

Specifications

find errors or contradictions, ...



I E T F

Search

Chat Live with the IETF Community

Home
About the IETF
Mission
Standards Process
Note Well
Nomcom
Blog
Info for Newcomers
Internet-Drafts
Datatracker
Search
Submit
RFC Pages
Search RFC Ed Index
RFC Editor Queue
IANA Pages
Protocol Parameters
IANA Transition
Working Groups
WG Charters
Email Lists
WG Chairs' Page

Request for Comments (RFC)

Memos in the **Requests for Comments (RFC)** document series contain technical and organizational notes about the Internet. They cover many aspects of computer networking, including protocols, procedures, programs, and concepts, as well as meeting notes, opinions, and sometimes humor. Below are links to RFCs, as available from ietf.org and from rfc-editor.org. Note that there is a brief time period when the two sites will be out of sync. When in doubt, the RFC Editor site is the authoritative source page.

RFCs associated with an active IETF Working Group can also be accessed from the Working Group's web page via IETF Working Groups.

IETF Repository Retrieval

- Advanced search options are available at IETF Datatracker and the RFC Search Page.
- A text index of RFCs is available on the IETF web site here: RFC Index (Text).
- To go directly to a text version of an RFC, type *https://www.ietf.org/rfc/rfcNNNN.txt* into the location field of your browser, where NNNN is the RFC number.

RFC Editor Repository Retrieval

- RFC Search Page
- RFC Index (HTML | TXT | XML)
- Additional listings of RFCs
- RFC Editor Queue

What do we put in a **refrigerator**?

http://prog21.dadgum.com/212.html

Too big!!!

1. Anything that fits into a refrigerator

Not edible!!!

2. Anything that is edible

flows everywhere!!!

3. Liquids must be in containers

not a liquid, but still requires a container

4. Items normally stored in containers must be in containers.

Doesn't need cold.

5. food must actually need refrigeration

So a penguin is ok ?!

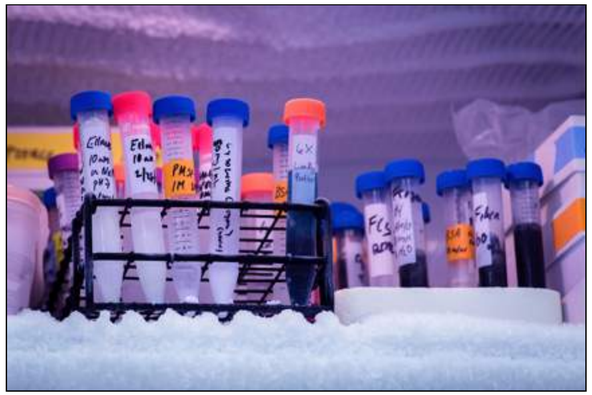
6. Items must not be alive.

Exception:

- yoghurt bacteria

2. Anything that is edible

Exception #1:
- medicine that needs to be kept cool



Exception #2:
- chemicals and organs in laboratory fridges

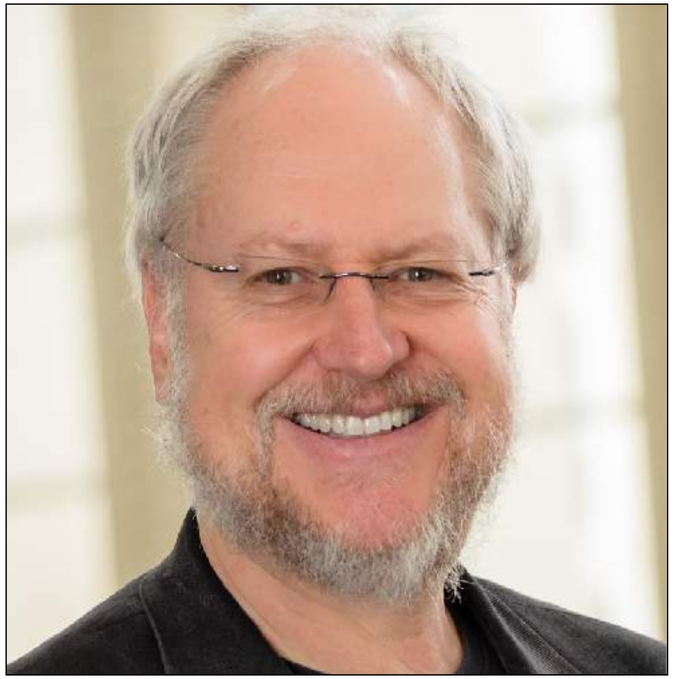
Exception #2.1:
- no Iced coffee in laboratory fridges

But what is cold?

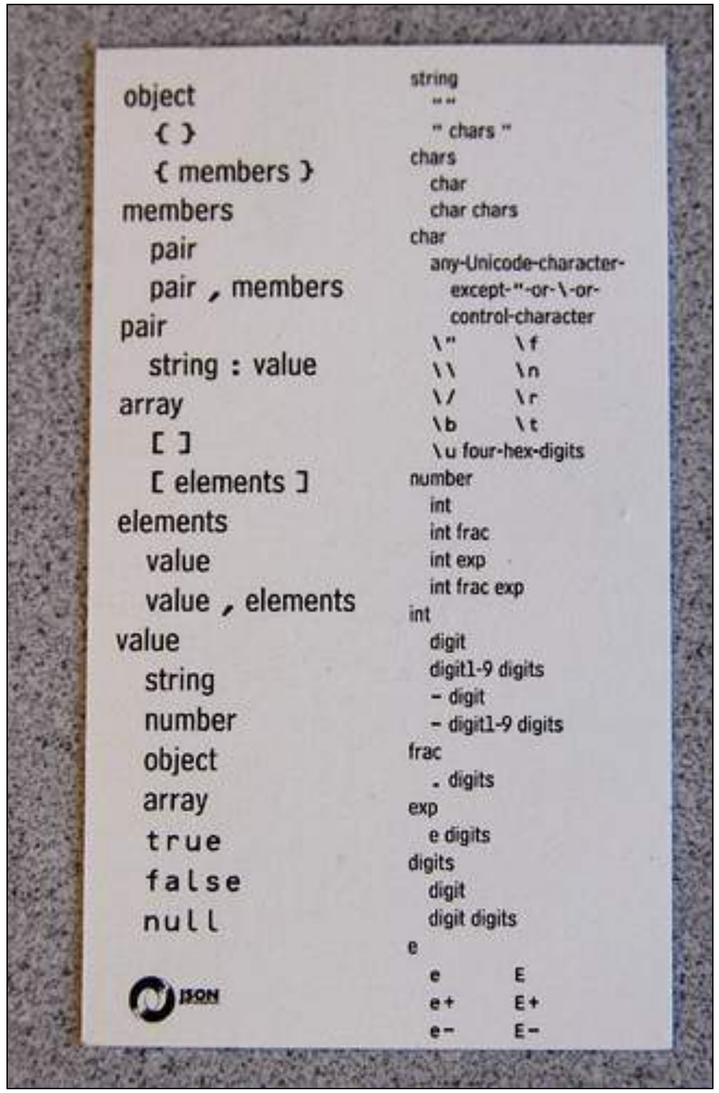
7. Define a temperature range

And for how long?

JSON SPECIFICATIONS



Douglas Crockford



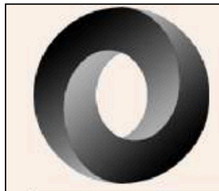
```
object
  { }
  { members }
members
  pair
  pair , members
pair
  string : value
array
  [ ]
  [ elements ]
elements
  value
  value , elements
value
  string
  number
  object
  array
  true
  false
  null
```

```
string
  " "
  " chars "
chars
  char
  char chars
char
  any-Unicode-character-
    except-"-or-\-or-
    control-character
  \"    \f
  \\    \n
  \/    \r
  \b    \t
  \u four-hex-digits
number
  int
  int frac
  int exp
  int frac exp
int
  digit
  digit1-9 digits
  - digit
  - digit1-9 digits
frac
  . digits
exp
  e digits
digits
  digit
  digit digits
e
  e     E
  e+    E+
  e-    E-
```

JSON

So simple, no version.

(1) 2002 – json.org



Introducing JSON

العربية Български 中文 Český Dansk Nederlands English Esperanto Français Deutsch Ελληνικά עברית Magyar Indonesia
Italiano 日本 한국어 فارسی Polski Português Română Русский Српско-хрватски Slovenščina Español Svenska Türkçe Tiếng Việt

ECMA-404 The JSON Data Interchange Standard.

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

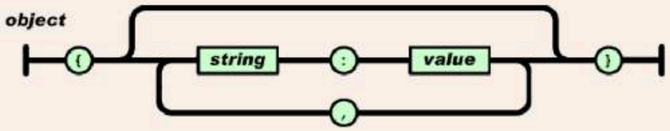
JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an *array*, vector, list, or sequence.

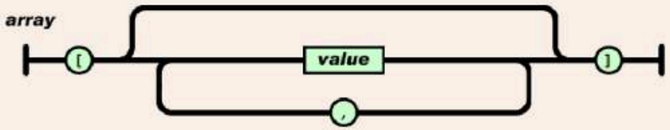
These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

In JSON, they take on these forms:

An *object* is an unordered set of name/value pairs. An object begins with `{` (left brace) and ends with `}` (right brace). Each name is followed by `:` (colon) and the name/value pairs are separated by `,` (comma).

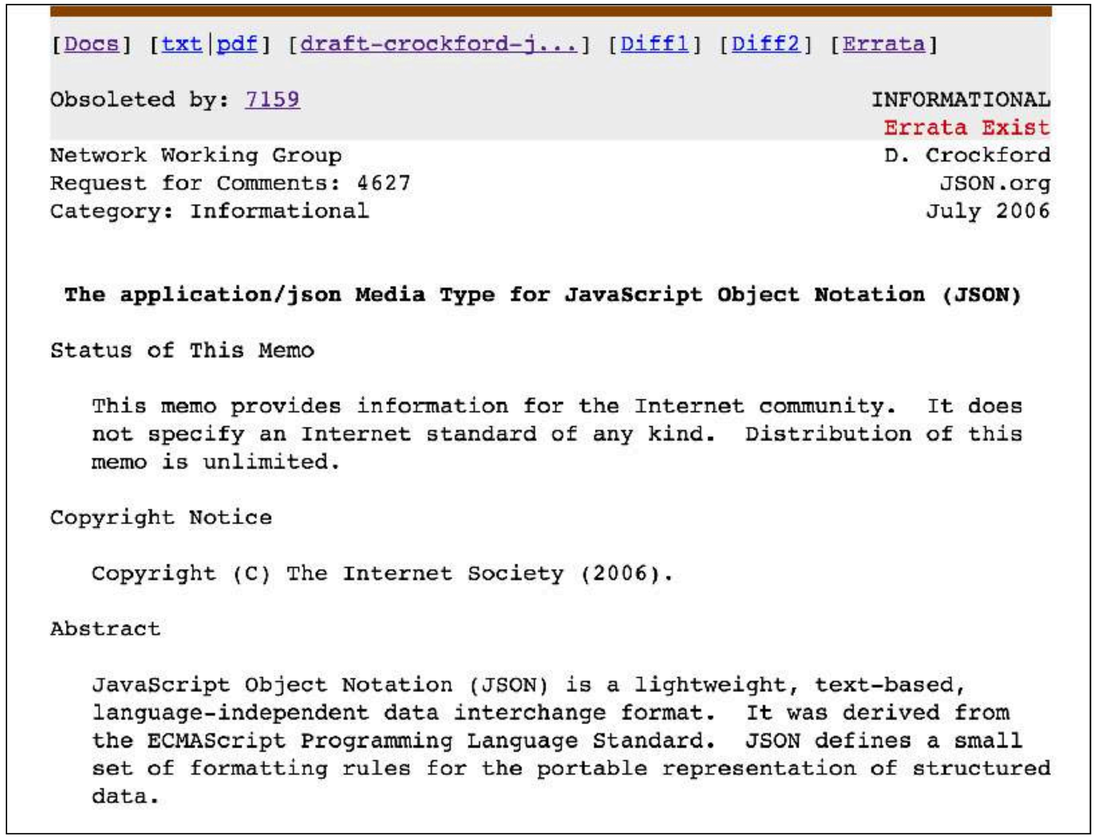


An *array* is an ordered collection of values. An array begins with `[` (left bracket) and ends with `]` (right bracket). Values are separated by `,` (comma).



A *value* can be a *string* in double quotes, or a *number*, or `true` or `false` or `null`, or an *object* or an *array*. These structures can be nested.

(2) 2006 – RFC 4627



[Docs] [txt|pdf] [draft-crockford-j...] [Diff1] [Diff2] [Errata]

Obsoleted by: 7159 — INFORMATIONAL — Errata Exist

Network Working Group — D. Crockford
Request for Comments: 4627 — JSON.org
Category: Informational — July 2006

The application/json Media Type for JavaScript Object Notation (JSON)

Status of This Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

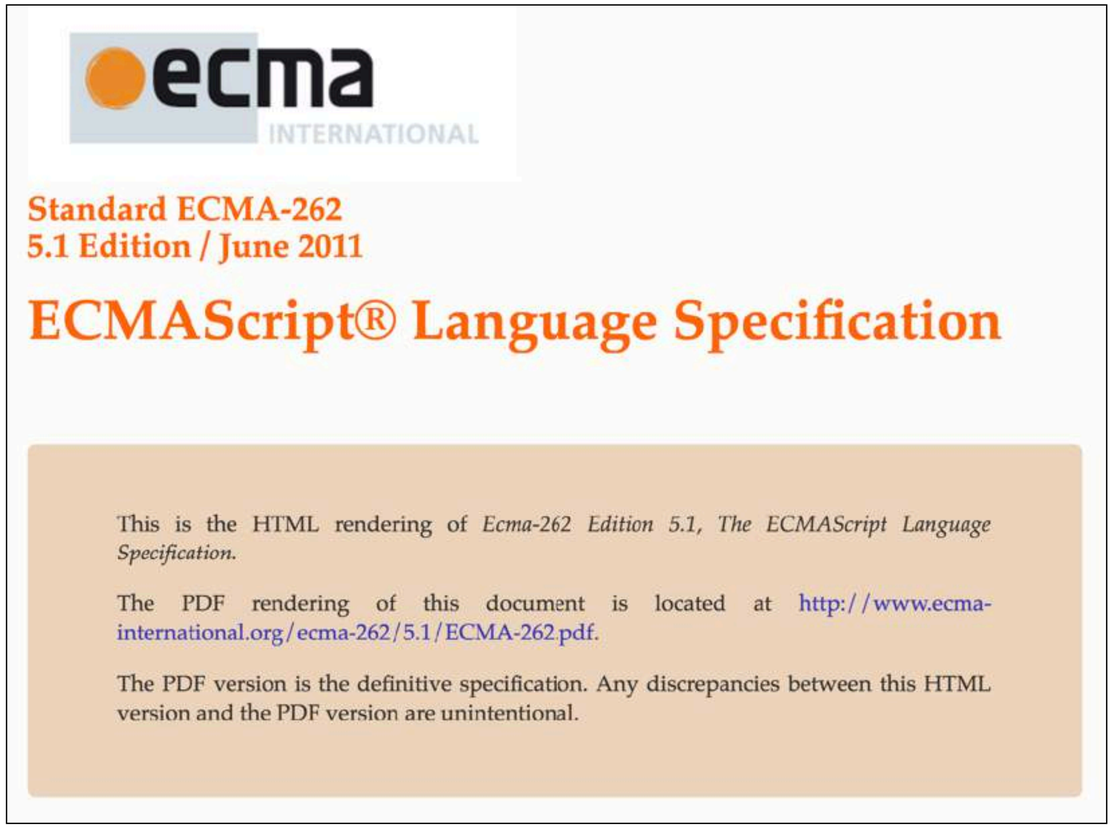
Copyright (C) The Internet Society (2006).

Abstract

JavaScript Object Notation (JSON) is a lightweight, text-based, language-independent data interchange format. It was derived from the ECMAScript Programming Language Standard. JSON defines a small set of formatting rules for the portable representation of structured data.

- Editor: D. Crockford
- application/json MIME media type

(3) 2011 – ECMAScript 262, section 15



ecma INTERNATIONAL

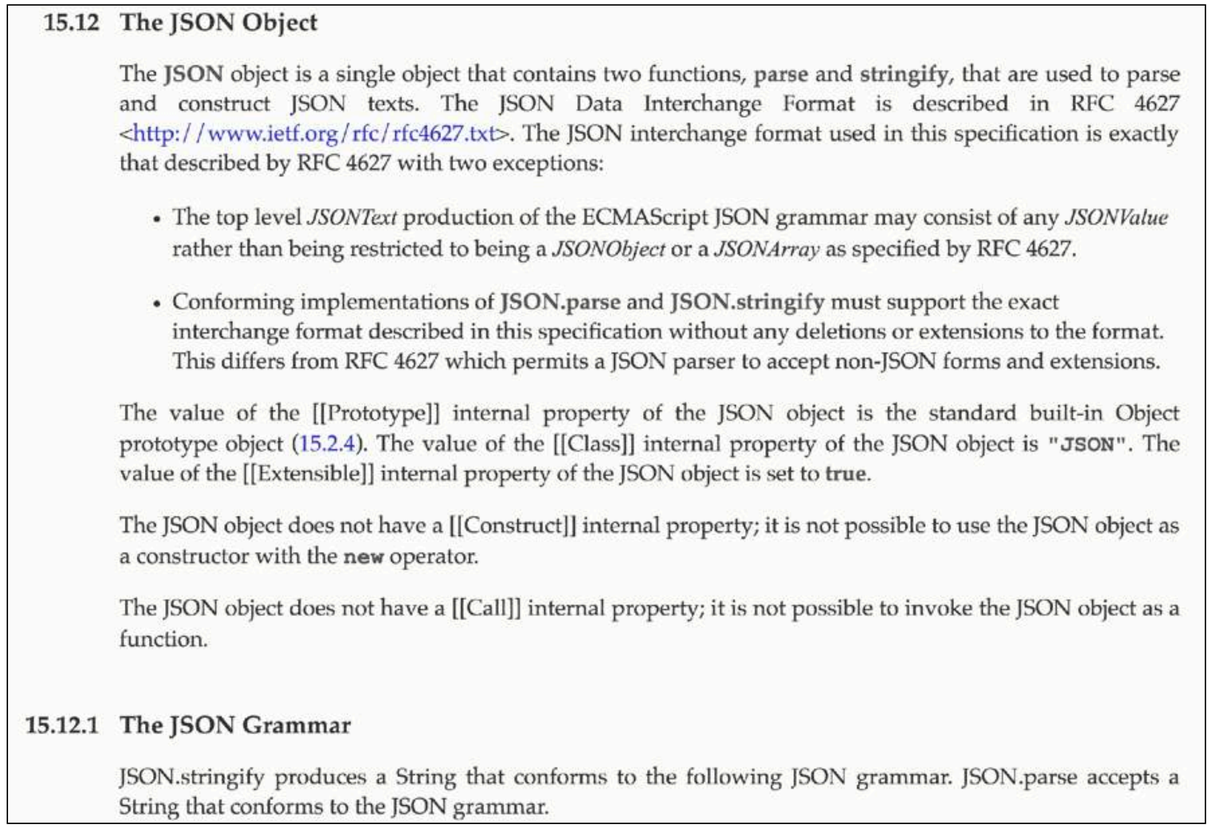
Standard ECMA-262
5.1 Edition / June 2011

ECMAScript® Language Specification

This is the HTML rendering of *Ecma-262 Edition 5.1, The ECMAScript Language Specification.*

The PDF rendering of this document is located at http://www.ecma-international.org/ecma-262/5.1/ECMA-262.pdf.

The PDF version is the definitive specification. Any discrepancies between this HTML version and the PDF version are unintentional.



15.12 The JSON Object

The **JSON** object is a single object that contains two functions, **parse** and **stringify**, that are used to parse and construct JSON texts. The JSON Data Interchange Format is described in RFC 4627 <http://www.ietf.org/rfc/rfc4627.txt>. The JSON interchange format used in this specification is exactly that described by RFC 4627 with two exceptions:

- The top level *JSONText* production of the ECMAScript JSON grammar may consist of any *JSONValue* rather than being restricted to being a *JSONObject* or a *JSONArray* as specified by RFC 4627.
- Conforming implementations of **JSON.parse** and **JSON.stringify** must support the exact interchange format described in this specification without any deletions or extensions to the format. This differs from RFC 4627 which permits a JSON parser to accept non-JSON forms and extensions.

The value of the [[Prototype]] internal property of the JSON object is the standard built-in Object prototype object (15.2.4). The value of the [[Class]] internal property of the JSON object is `"JSON"`. The value of the [[Extensible]] internal property of the JSON object is set to **true**.

The JSON object does not have a [[Construct]] internal property; it is not possible to use the JSON object as a constructor with the **new** operator.

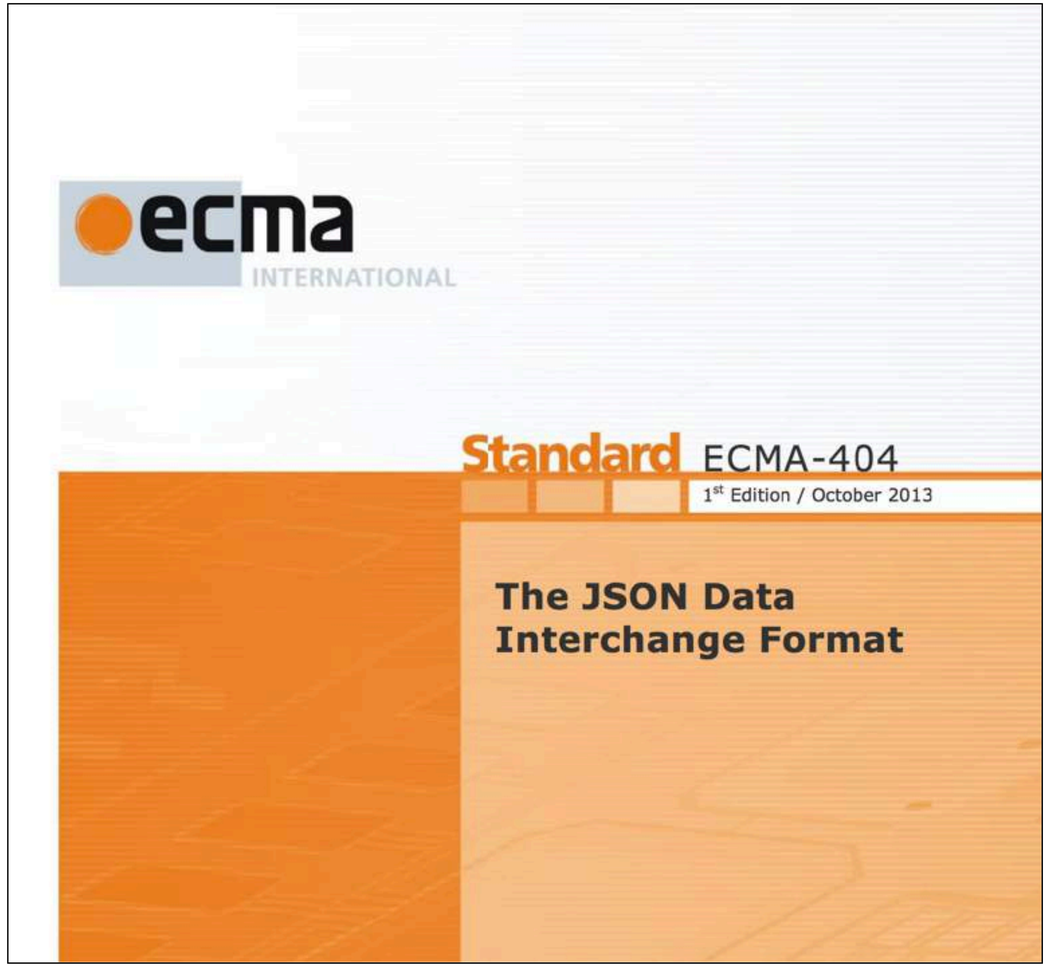
The JSON object does not have a [[Call]] internal property; it is not possible to invoke the JSON object as a function.

15.12.1 The JSON Grammar

JSON.stringify produces a String that conforms to the following JSON grammar. JSON.parse accepts a String that conforms to the JSON grammar.

just the ECMA version…

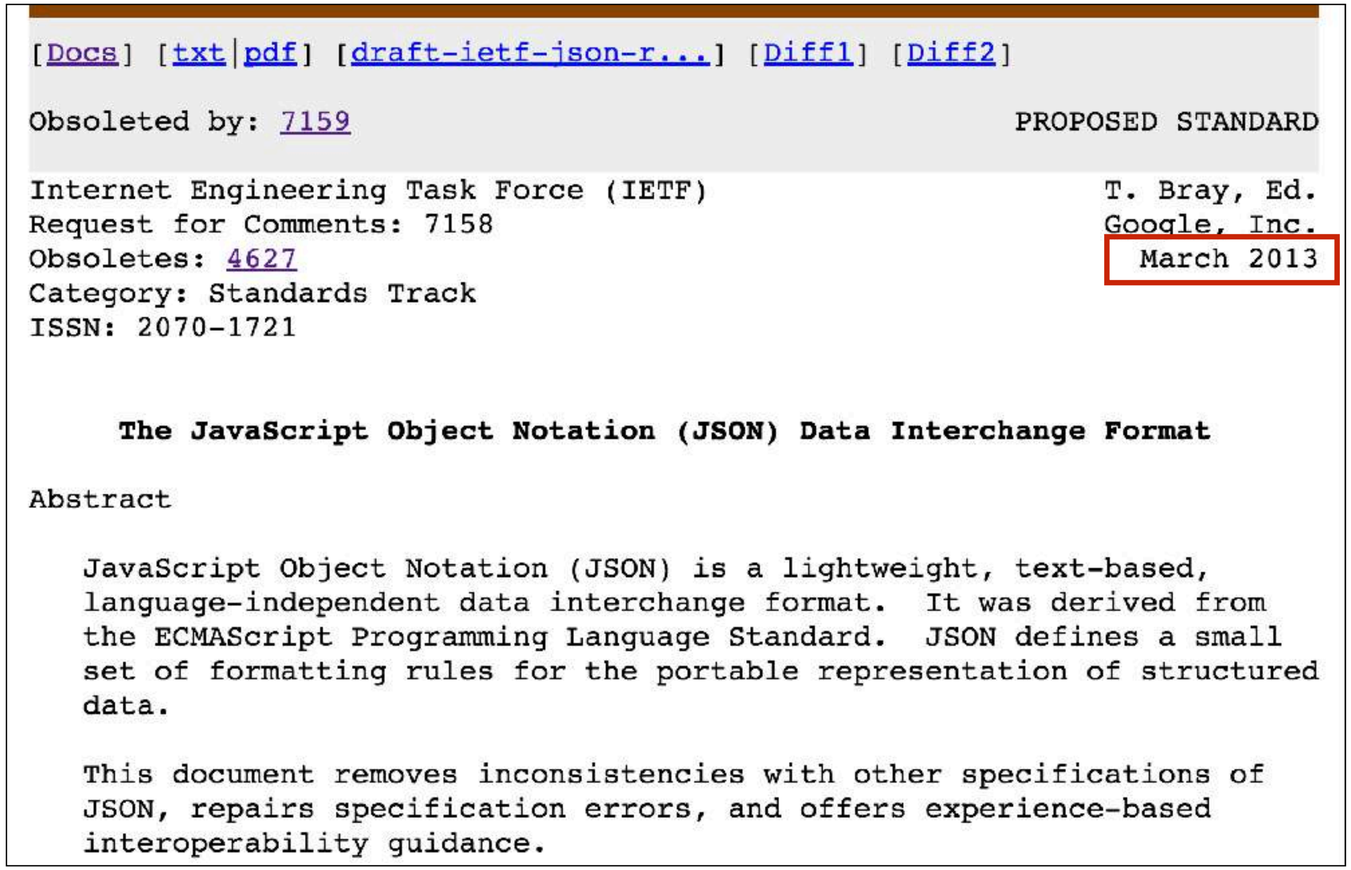
(4) 2013 – ECMA 404



"Someone told the ECMA working group that the IETF had gone crazy and was going to rewrite JSON with no regard for compatibility and break the whole Internet and something had to be done urgently about this terrible situation. (...) It doesn't address any of the gripes that were motivating the IETF revision.

- Tim Bray (Google, RFC 7158/9 editor)

(5) 2014 – RFC 7158



[Docs] [txt|pdf] [draft-ietf-json-r...] [Diff1] [Diff2]

Obsoleted by: 7159 PROPOSED STANDARD

Internet Engineering Task Force (IETF)
Request for Comments: 7158
Obsoletes: 4627
Category: Standards Track
ISSN: 2070-1721

T. Bray, Ed.
Google, Inc.
March 2013

The JavaScript Object Notation (JSON) Data Interchange Format

Abstract

JavaScript Object Notation (JSON) is a lightweight, text-based, language-independent data interchange format. It was derived from the ECMAScript Programming Language Standard. JSON defines a small set of formatting rules for the portable representation of structured data.

This document removes inconsistencies with other specifications of JSON, repairs specification errors, and offers experience-based interoperability guidance.

- allows scalars at root level as ECMA does (`"asd"` or `123`)
- warns about bad practices but does not forbid them (bad Unicode, repeated object keys, …)

(6) 2014 – RFC 7159

[Docs] [txt|pdf] [draft-ietf-json-r...] [Diff1] [Diff2] [Errata]

PROPOSED STANDARD
Errata Exist

Internet Engineering Task Force (IETF)
Request for Comments: 7159
Obsoletes: 4627, 7158
Category: Standards Track
ISSN: 2070-1721

T. Bray, Ed.
Google, Inc.
March 2014

The JavaScript Object Notation (JSON) Data Interchange Format

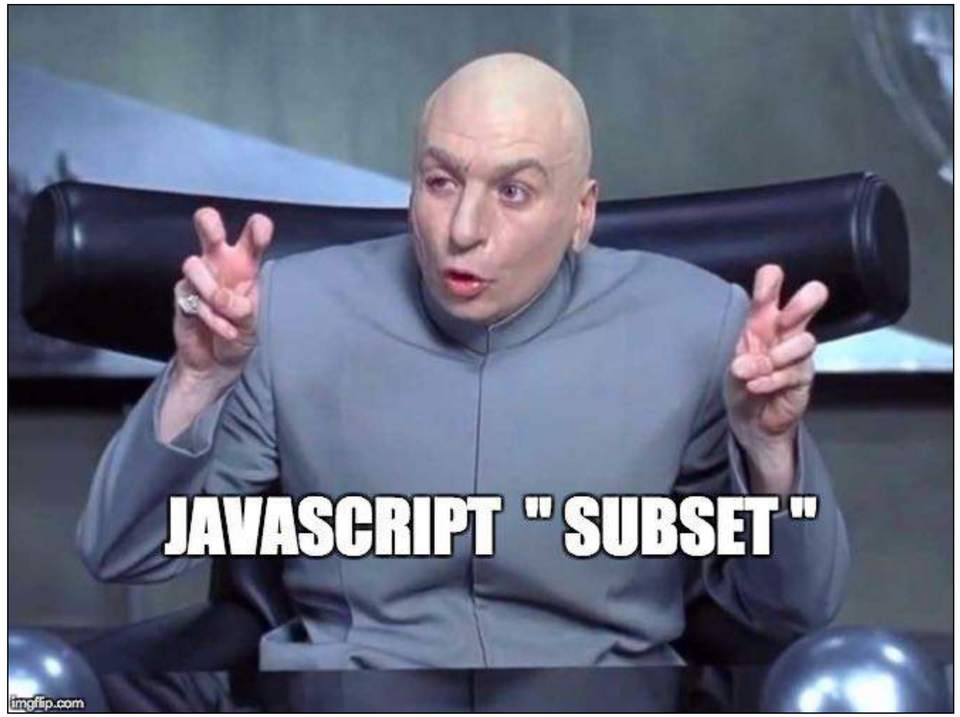
Abstract

JavaScript Object Notation (JSON) is a lightweight, text-based, language-independent data interchange format. It was derived from the ECMAScript Programming Language Standard. JSON defines a small set of formatting rules for the portable representation of structured data.

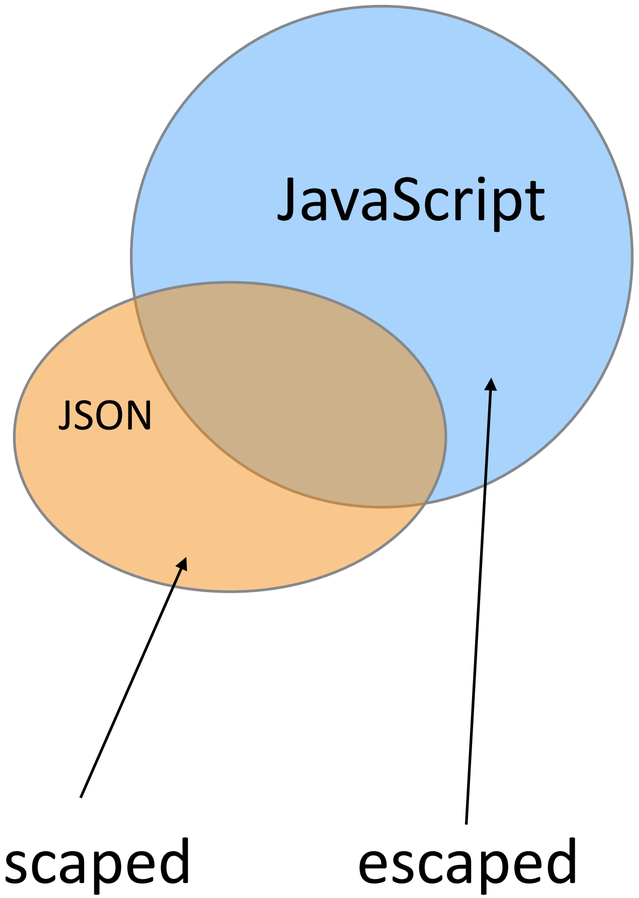
This document removes inconsistencies with other specifications of JSON, repairs specification errors, and offers experience-based interoperability guidance.

- because of typo in RFC 7159: "March 2013"

JSON vs. JavaScript



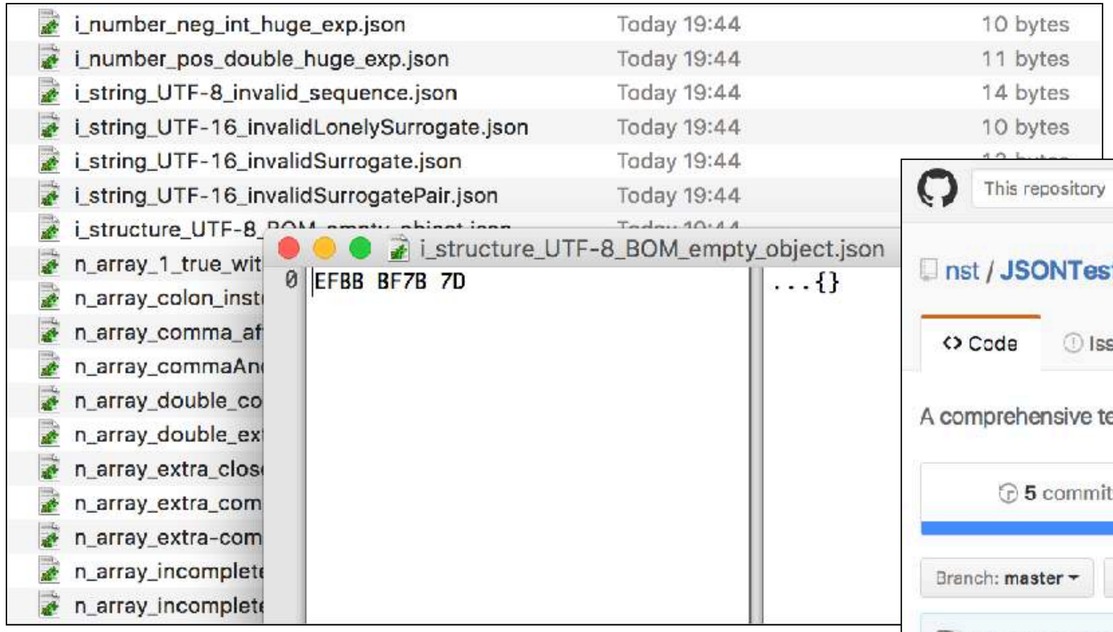
JSON's design goals *were for it to be minimal, portable, textual, and a* ***subset of JavaScript****.*
- RFC 7159



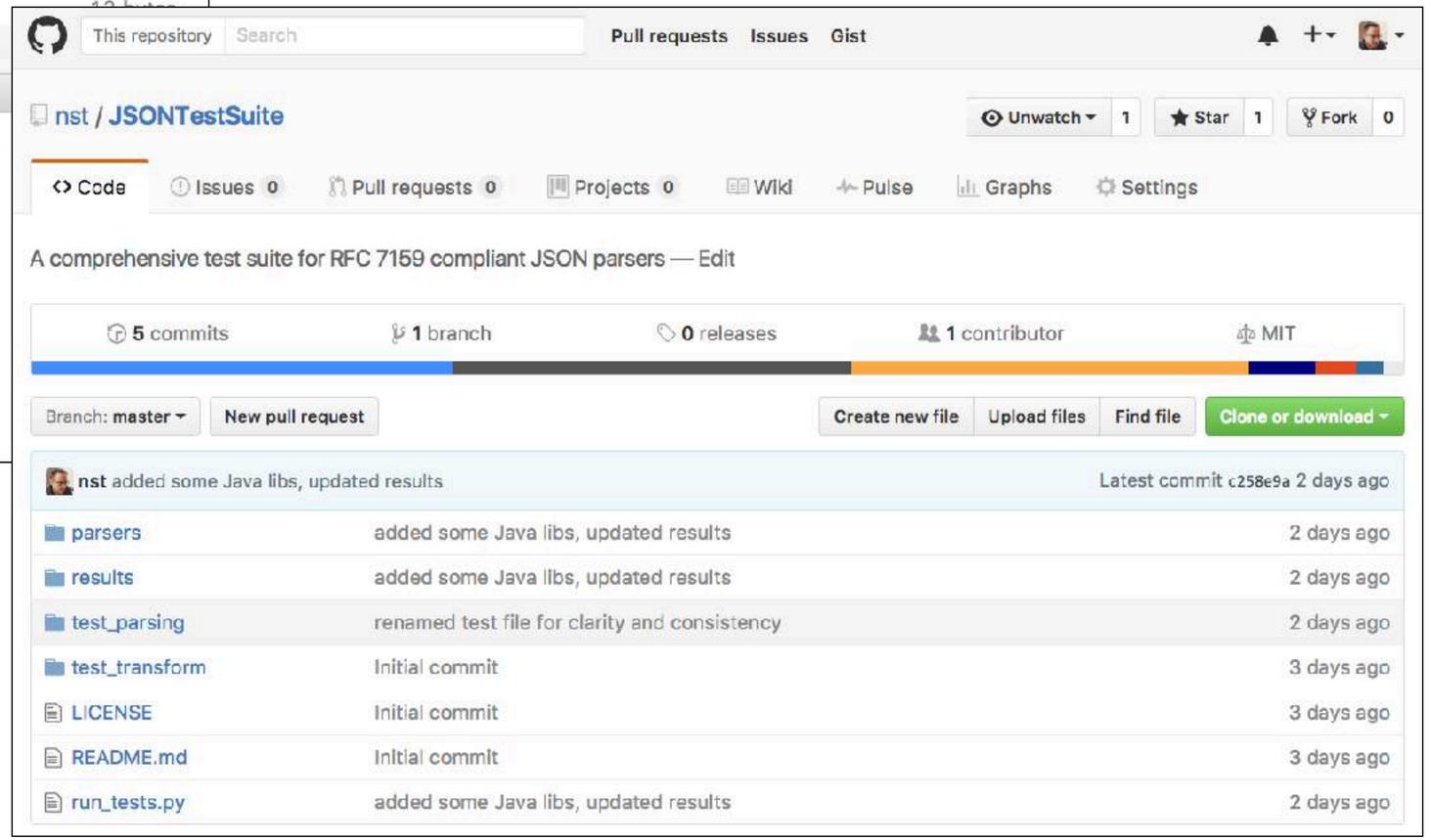

```
U+2028 LINE SEPARATOR
U+2029 PARAGRAPH SEPARATOR
```

WRITE TESTS

300+ JSON Parsing Test Cases



https://github.com/nst/JSONTestSuite



mostly handwritten,
a bit of fuzzing as well

Test Files

no - shouldn't be parsed (could be yes, or impl. defined)

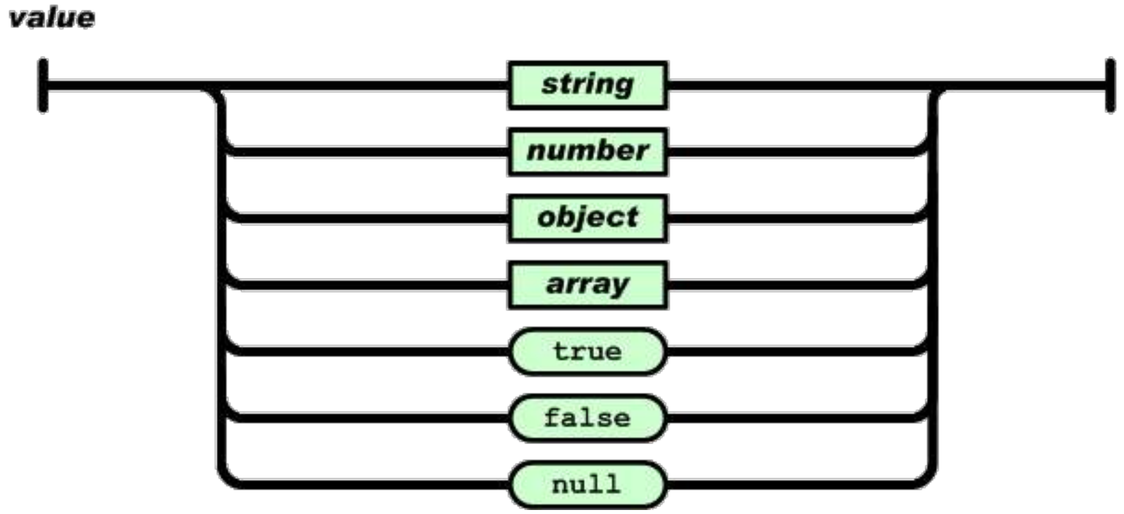
null_char - description of the test

`n_string_null_char.json`

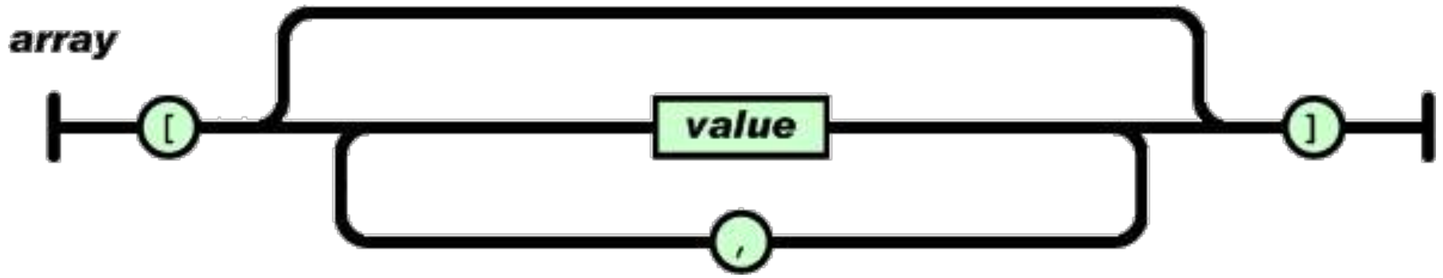
`["00"]`

string - can also be array, object, number or structure

the null character, invalid unescaped

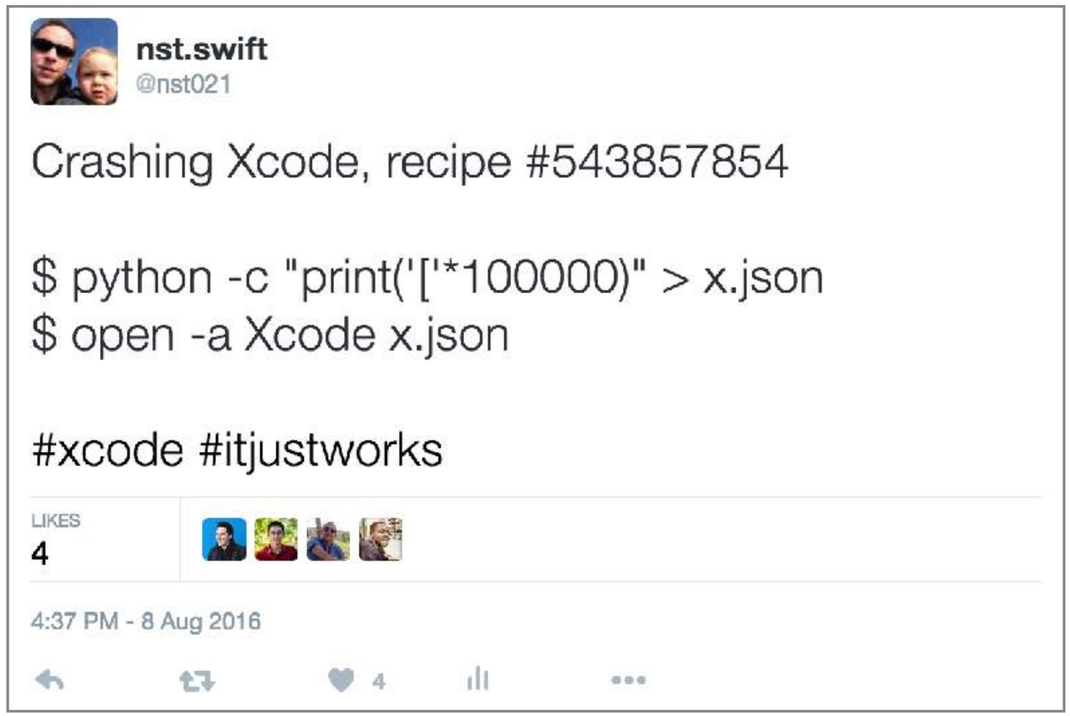


Scalars at root level	allowed in RFC 7159
y_structure_lonely_string.json	"asd"
Trailing commas	JSON "extension"
n_object_trailing_comma.json	{"id":0,} Valid in iOS AppKit
n_object_several_trailing_commas.json	{"id":0,,,,,}
Comments	JSON "extension"
y_string_comments.json	["a/*b*/c/*d//e"]
n_object_trailing_comment.json	{"a":"b"}/**/
n_structure_object_with_comment.json	{"a":/*comment*/"b"}
Unclosed structures	
n_structure_object_unclosed_no_value.json	{"":
n_structure_object_followed_by_closing_object.json	{}



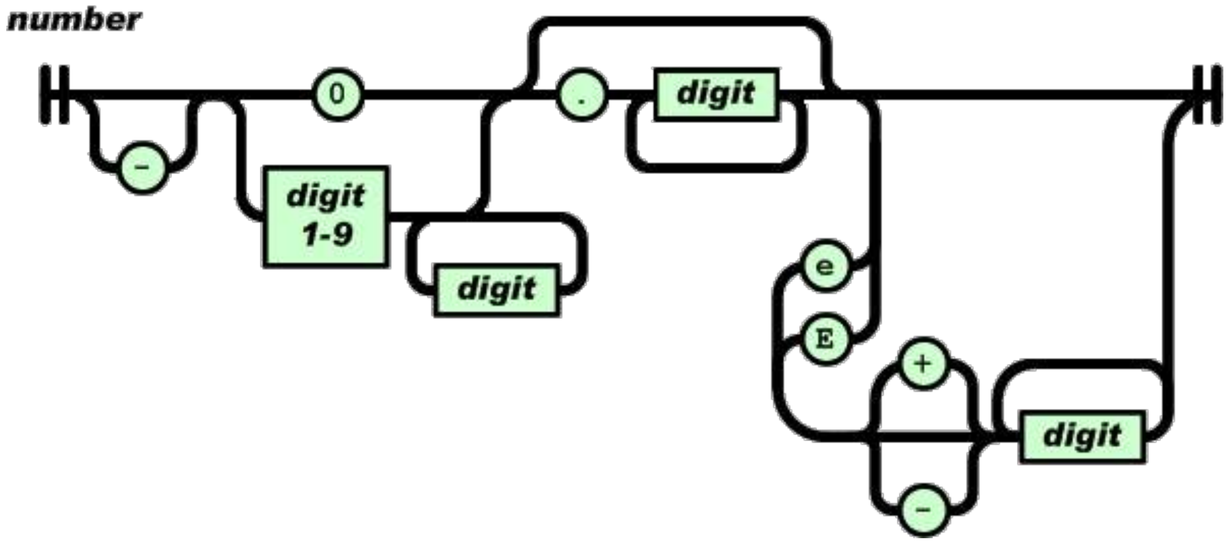
n_array_comma_and_number.json	[,1]
n_array_colon_instead_of_comma.json	["": 1] — Valid in Lua dkjson
n_array_unclosed_with_new_lines.json	[1,0A10A,1

White Spaces - only space, tab, linefeed and cr allowed between `[]{}:,`	
n_structure_whitespace_formfeed.json	[0C] — Valid in json-rust
n_structure_whitespace_U+2060_word_joiner.json	[E281A0]
n_structure_no_data.json	
Nested Structures	parsers may set limits
n_structure_100000_opening_arrays.json	[[[[[[[[[[[[[[[[[[...



Bad isolation of fault domain.

JSON lib or syntax highlighter shouldn't crash the whole process.



Valid JSON in Python,
also generated by JSON encoder

NaN and Infinity	
n_number_NaN.json	[NaN]
n_number_minus_infinity.json	[-Infinity]
Hex Numbers	
n_number_hex_2_digits.json	[0x42]
Range and Precision	
y_number_very_big_negative_int.json	[-237462374673276894(...)
Exponential Notation	
n_number_0_capital_E+.json	[0E+]
n_number_.2e-3.json	[.2e-3]
y_number_double_huge_neg_exp.json	[123.456e-789]

Valid JSON in JS

Numerical Precision Issues

*"A JSON parser **MUST** accept all texts that conform to the JSON grammar"*
(RFC 7159, section 9)

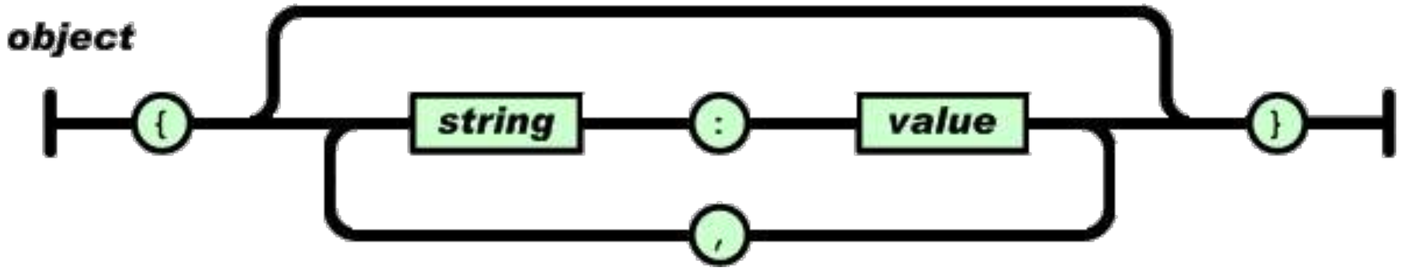
*"An implementation **may** set limits on the range and precision of numbers."*
(RFC 7159, section 9)

```
1e9999
```

```
0.00000000000000000000001
```

So, are parsers allowed to raise errors in these cases??

In practice, many APIs (FB, Twitter) transmit user IDs as strings to avoid precision issues.



y_object_empty_key.json	{"":0}
y_object_duplicated_key_and_value.json	{"a":"b","a":"b"}
n_object_double_colon.json	{"x"::"b"} — Valid in jsmn
n_object_key_with_single_quotes.json	{key: 'value'}
n_object_missing_key.json	{:"b"}
n_object_non_string_key.json	{1:1}

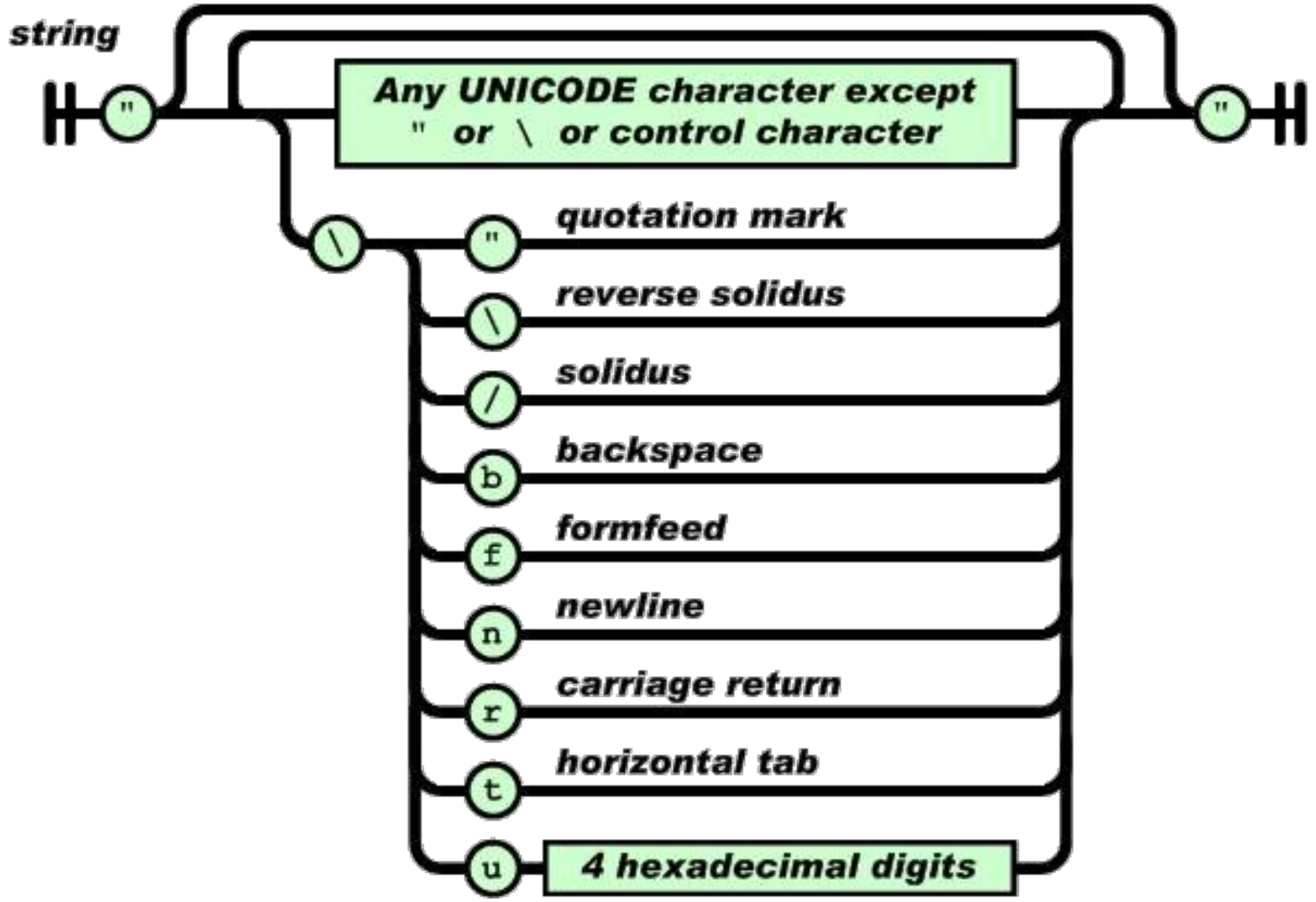
Duplicated Keys

*"The names within an object **should be unique**." (section 4)*

*"(some) implementations report an error or **fail to parse** the object" (section 4)*

*"A JSON parser **MUST accept** all texts that conform to the JSON grammar." (section 9)*

RFC doesn't tell clearly if failing to parse dupe keys is compliant or not.


string
"
Any UNICODE character except
" or \ or control character
"
\
"
quotation mark
\
reverse solidus
/
solidus
b
backspace
f
formfeed
n
newline
r
carriage return
t
horizontal tab
u
4 hexadecimal digits

File Encoding - UTF-8 by default, UTF-16 and UTF-32 allowed	
y_string_utf16.json	FFFE[00"00E900"00]00
n_string_iso_latin_1.json	["E9"]
Byte Order Mark	"may be ignored"
n_structure_UTF8_BOM_no_data.json	EFBBBF — Valid in json.NET
n_structure_incomplete_UTF8_BOM.json	EFBB{}
i_structure_UTF-8_BOM_empty_object.json	EFBBBF{}
Control Characters - all chars may be u-escaped, mandatory for 0x00-0x1F	
n_string_unescaped_crtl_char.json	["a00a"] — Valid in Java gson
y_string_unescaped_char_delete.json	["7F"]
n_string_escape_x.json	["\x00"]
Escape - also, some chars require backslash escape	
y_string_allowed_escapes.json	["\"\\/\b\f\n\r\t"]
n_structure_bad_escape.json	["\
y_string_backslash_and_u_escaped_zero.json	["\u0000"]
n_string_invalid_unicode_escape.json	["\uqqqq"]
n_string_incomplete_escaped_character.json	["\u00A"]

Invalid Codepoints

- Codepoints after U+FFFF are encoded in surrogate pairs
  eg. U+1D11E gets "\uD834\uDD1E"

- **Vector is valid JSON, payload is invalid codepoints**

Escaped non-Unicode Characters - UTF-8 by default, UTF-16 and UTF-32 allowed	
y_string_accepted_surrogate_pair.json	["\uD801\udc37"]
n_string_incomplete_escaped_character.json	["\u00A"]
i_string_incomplete_surrogates_escape_valid.json	["\uD800\uD800\n"]
i_string_lone_second_surrogate.json	["\uDFAA"]
i_string_1st_valid_surrogate_2nd_invalid.json	["\uD888\u1234"]
i_string_inverted_surrogates_U+1D11E.json	["\uDd1e\uD834"]

Invalid Codepoints Replacement

- *“The ABNF cannot at the same time allow non conformant Unicode codepoints (section 7) and states conformance to Unicode (section 1).”* RFC 7159 errata 3984

- Editors ruled that parsers must parse the data, but result is undefined.
   U+FFFD REPLACEMENT CHARACTER free to be used in several ways

Public Review Issue #121

Recommended Practice for Replacement Characters

The Unicode Technical Committee has been requested to specify what the recommended practice is for replacement characters in handling ill-formed subsequences.

When converting between Unicode encoding forms or when validating Unicode text, there is no requirement that an ill-formed subsequence be replaced using U+FFFD character(s); an application can, for example, throw an exception or substitute other characters such as "?". However, even when replacement with U+FFFD is done there are several possible approaches that can be taken. The principal three approaches can be stated as policy options:

1. Replace the entire ill-formed subsequence by a single U+FFFD.
2. Replace each maximal subpart of the ill-formed subsequence by a single U+FFFD.
3. Replace each code unit of the ill-formed subsequence by a single U+FFFD.

In these policy statements, "entire ill-formed subsequence" refers to all code units in the ill-formed subsequence up to but not including the start of the next well-formed code unit sequence. The term "maximal subpart of the ill-formed subsequence" refers to the longest potentially valid initial subsequence or, if none, then to the next single code unit.

The following table illustrates the application of these alternative policies for an example of conversion of UTF-8 to UTF-16, the most common kind of conversion for which the differences are apparent and for which a recommended practice would be desirable for interoperability:

	61	F1	80	80	E1	80	C2	62
1	U+0061	U+FFFD						U+0062
2	U+0061	U+FFFD			U+FFFD		U+FFFD	U+0062
3	U+0061	U+FFFD	U+FFFD	U+FFFD	U+FFFD	U+FFFD	U+FFFD	U+0062

The UTC has indicated a tentative preference for option #2, but is interested in feedback on what would be the best recommended practice, and reasons for that choice. The UTC also requests feedback about which products or libraries are known to follow these or other policies for replacement of ill-formed subsequences on conversion or validation.

RFC 7159 Ambiguities

*A JSON parser **MUST** accept all texts that conform to the JSON grammar.*

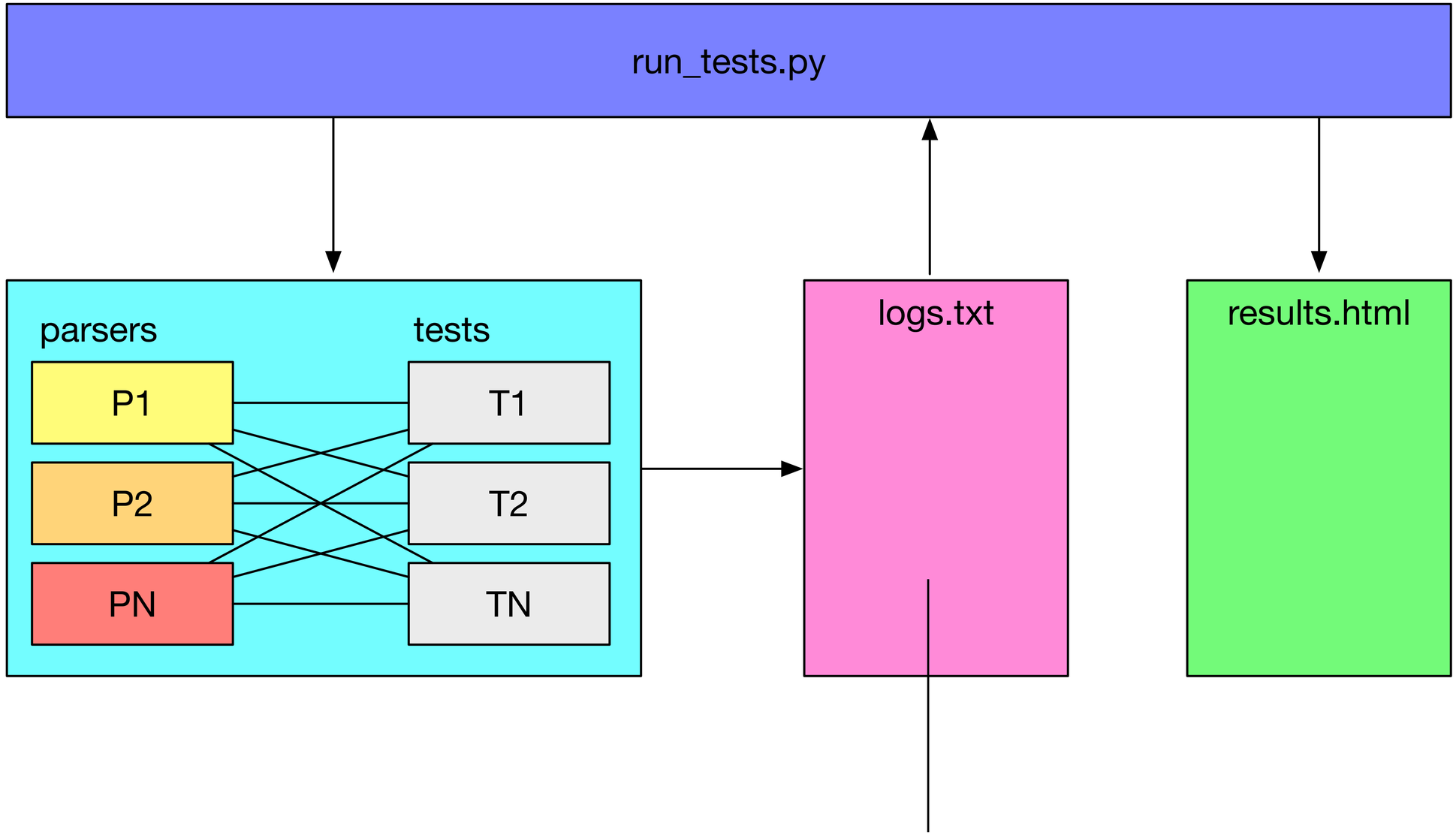
An implementation may set limits on the size of texts that it accepts.
An implementation may set limits on the maximum depth of nesting.
An implementation may set limits on the range and precision of numbers.
An implementation may set limits on the length and character contents of strings.

(section 9)

***MUST** - This word, or the terms "REQUIRED" or "SHALL",*
*mean that the definition is **an absolute requirement** of the specification.*
(RFC 2119)

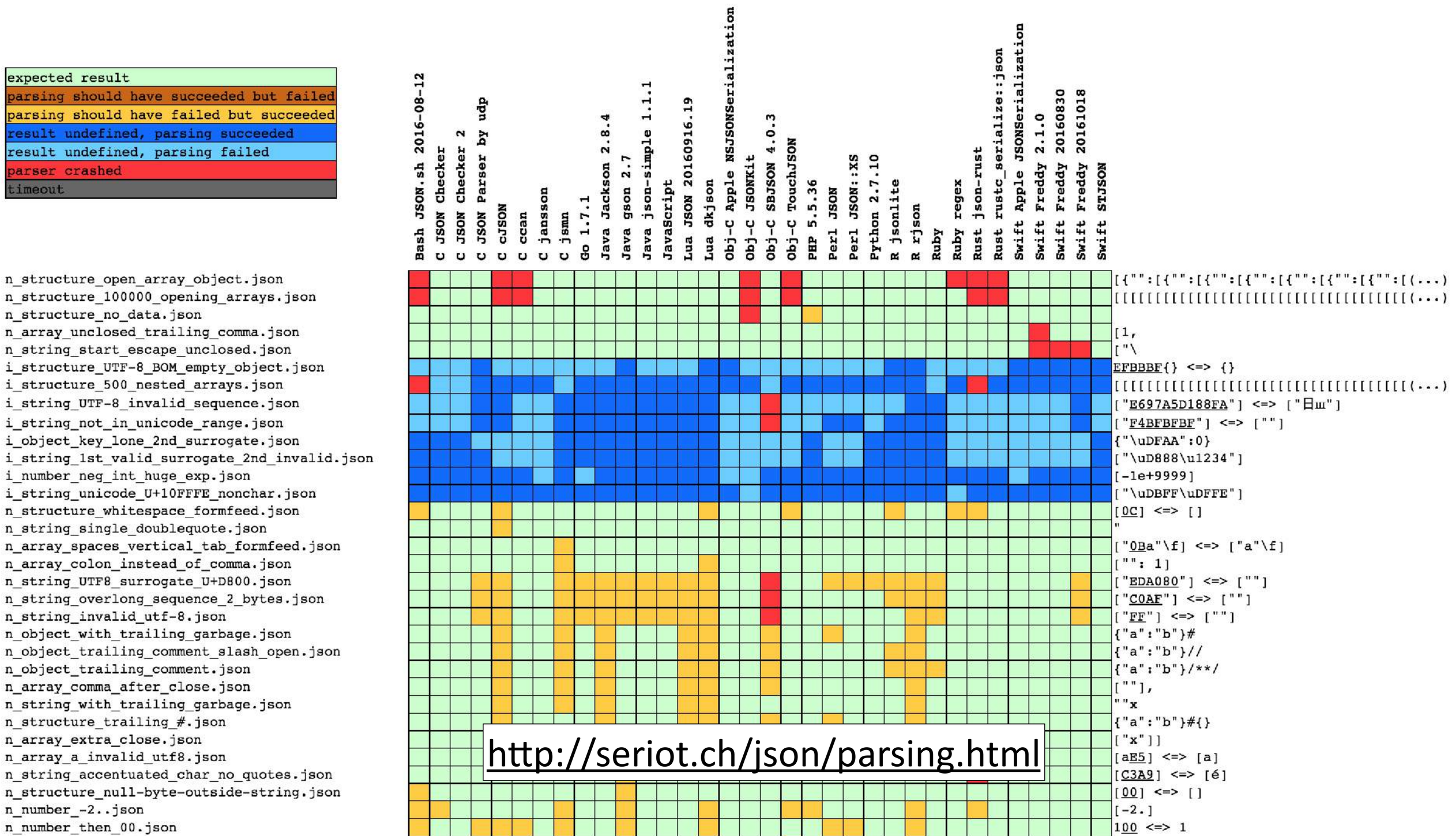
```
f(s) { return null; } # shortest fully RFC 7159-compliant JSON parser
```

	match JSON grammar	don't match JSON grammar
input accepted		parser "extension"
input rejected	parser "limit"	




```
Python 2.7.10    SHOULD_HAVE_FAILED   n_number_infinity.json
```

TESTS RESULTS


expected result
parsing should have succeeded but failed
parsing should have failed but succeeded
result undefined, parsing succeeded
result undefined, parsing failed
parser crashed
timeout
Bash JSON.sh 2016-08-12
C JSON Checker
C JSON Checker 2
C JSON Parser by udp
C cJSON
C ccan
C jansson
C jsmn
Go 1.7.1
Java Jackson 2.8.4
Java gson 2.7
Java json-simple 1.1.1
JavaScript
Lua JSON 20160916.19
Lua dkjson
Obj-C Apple NSJSONSerialization
Obj-C JSONKit
Obj-C SBJSON 4.0.3
Obj-C TouchJSON
PHP 5.5.36
Perl JSON
Perl JSON::XS
Python 2.7.10
R jsonlite
R rjson
Ruby
Ruby regex
Rust json-rust
Rust rustc_serialize::json
Swift Apple JSONSerialization
Swift Freddy 2.1.0
Swift Freddy 20160830
Swift Freddy 20161018
Swift STJSON
n_structure_open_array_object.json
n_structure_100000_opening_arrays.json
n_structure_no_data.json
n_array_unclosed_trailing_comma.json
n_string_start_escape_unclosed.json
i_structure_UTF-8_BOM_empty_object.json
i_structure_500_nested_arrays.json
i_string_UTF-8_invalid_sequence.json
i_string_not_in_unicode_range.json
i_object_key_lone_2nd_surrogate.json
i_string_1st_valid_surrogate_2nd_invalid.json
i_number_neg_int_huge_exp.json
i_string_unicode_U+10FFFE_nonchar.json
n_structure_whitespace_formfeed.json
n_string_single_doublequote.json
n_array_spaces_vertical_tab_formfeed.json
n_array_colon_instead_of_comma.json
n_string_UTF8_surrogate_U+D800.json
n_string_overlong_sequence_2_bytes.json
n_string_invalid_utf-8.json
n_object_with_trailing_garbage.json
n_object_trailing_comment_slash_open.json
n_object_trailing_comment.json
n_array_comma_after_close.json
n_string_with_trailing_garbage.json
n_structure_trailing_#.json
n_array_extra_close.json
n_array_a_invalid_utf8.json
n_string_accentuated_char_no_quotes.json
n_structure_null-byte-outside-string.json
n_number_-2..json
n_number_then_00.json
[{"":[{"":[{"":[{"":[{"":[{"":[(...)
[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[(...)
[1,
["\
EFBBBF{} <=> {}
[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[(...)
["E697A5D188FA"] <=> ["日ш"]
["F4BFBFBF"] <=> [""]
{"\uDFAA":0}
["\uD888\u1234"]
[-1e+9999]
["\uDBFF\uDFFE"]
[0C] <=> []
"
["0Ba"\f] <=> ["a"\f]
[": 1]
["EDA080"] <=> [""]
["C0AF"] <=> [""]
["FF"] <=> [""]
{"a":"b"}#
{"a":"b"}//
{"a":"b"}/**/
[""],
""x
{"a":"b"}#{}
["x"]]
[aE5] <=> [a]
[C3A9] <=> [é]
[00] <=> []
[-2.]
100 <=> 1
http://seriot.ch/json/parsing.html

parser crashed	most critical, will kill the whole process especially if users can POST random data
parsing should have succedded but failed	may prevent parsing a whole document
result undefined, parsing failed	
parsing should have failed but succeeded	wait until you change the parser and cry
result undefined, parsing succeeded	

C Parsers

	jsmn	jansson	ccan	cJSON	json-parser
Parses `["\u0000"]`	YES	NO	NO	NO	YES
Too liberal	YES	NO	NO	YES	YES
Crash on nested structs.	NO	NO	YES	YES	NO
Rejects big numbers	YES	YES	NO	NO	NO

jsmn is missing all that functionality, but instead is designed to be **robust** (it should work fine even with erroneous data), **fast** (it parses data on the fly and is re-entrant), **portable** (no superfluous dependencies or non-standard C extensions). And of course, **simplicity** is a key feature.

"don't forget jsmn is a primitive minimalist parser, if you need a tool to validate your JSON - consider using grownup alternatives"
https://github.com/zserge/jsmn/issues/7

Perl JSON::XS

i_number_neg_int_huge_exp.json	[-1e+9999]
i_number_pos_double_huge_exp.json	[1.5e+9999]
i_object_key_lone_2nd_surrogate.json	{"\uDFAA":0}
i_string_1st_surrogate_but_2nd_missing.json	["\uDADA"]
i_string_1st_valid_surrogate_2nd_invalid.json	["\uD888\u1234"]
i_string_UTF-16_invalid_lonely_surrogate.json	["\ud800"]
…gate.json	["\ud800abc"]
…ce.json	["E697A5D188FA"] <=> ["日ш"]
…_and_escape_valid.json	["\uD800\n"]
…_pair.json	["\uDd1ea"]
…s_escape_valid.json	["\uD800\uD800\n"]
…U+1D11E.json	["\uDd1e\uD834"]
…e.json	["\uDFAA"]
….json	["F4BFBFBF"] <=> [""]
	["E0FF"] <=> [""]
i_string_unicode_U+10FFFE_nonchar.json	["\uDBFF\uDFFE"]
i_string_unicode_U+1FFFE_nonchar.json	["\uD83F\uDFFE"]
i_string_unicode_U+FDD0_nonchar.json	["\uFDD0"]
i_string_unicode_U+FFFE_nonchar.json	["\uFFFE"]
i_structure_500_nested_arrays.json	[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[(...)
i_structure_UTF-8_BOM_empty_object.json	EFBBBF{} <=> {}
n_array_newlines_unclosed.json	["a",0A40A,1, <=> ["a", 4 ,1,
n_array_unclosed_with_new_lines.json	[1,0A10A,1 <=> [1, 1 ,1
n_number_then_00.json	100 <=> 1
n_string_UTF8_surrogate_U+D800.json	["EDA080"] <=> [""]
y_array_with_1_and_newline.json	[10A] <=> [1]
y_object_with_newlines.json	{0A"a": "b"0A} <=> { "a": "b" }
y_string_space.json	" "
y_string_utf16.json	FFFE[00"00E900"00]00 <=> [""]
y_structure_lonely_false.json	false
y_structure_lonely_null.json	null
y_structure_lonely_string.json	"asd"
y_structure_string_empty.json	""

JSON - search.cpan.org

search.cpan.org/~makamaka/JSON-2.90/lib/JSON.pm

- strict checking of JSON correctness

  There is no guessing, no generating of illegal JSON texts by default, and only JSON is accepted as input by default (the latter is a security feature).

Objective-C Parsers

	JSONKit	TouchJSON	SBJSON
Crash on nested structs.	YES	NO	YES
Crash on invalid UTF-8	NO	NO	YES
Parses trailing garbage `[]x`	NO	NO	YES
Rejects big numbers	NO	YES	NO
Parses bad numbers `[0.e1]`	NO	YES	NO
Treats `0x0C FORM FEED` as white space	NO	YES	NO
Parses non-char. `["\uFFFF"]`	NO	YES	YES

Several 3rd-party parsers because NSJSONSerialization only since iOS 5.

Many apps do probably still use them. SBJSON is still maintained.

Apple (NS)JSONSerialization

- **Parser limitations** (undocumented):
  - won’t parse big numbers `[123123e100000]`
  - won’t parse u-escaped invalid codepoints `["\ud800"]` (may be considered as a bug indeed)
- **Parser extensions** (undocumented):
  - does parse trailing commas [1,] and `{"a":0,}`

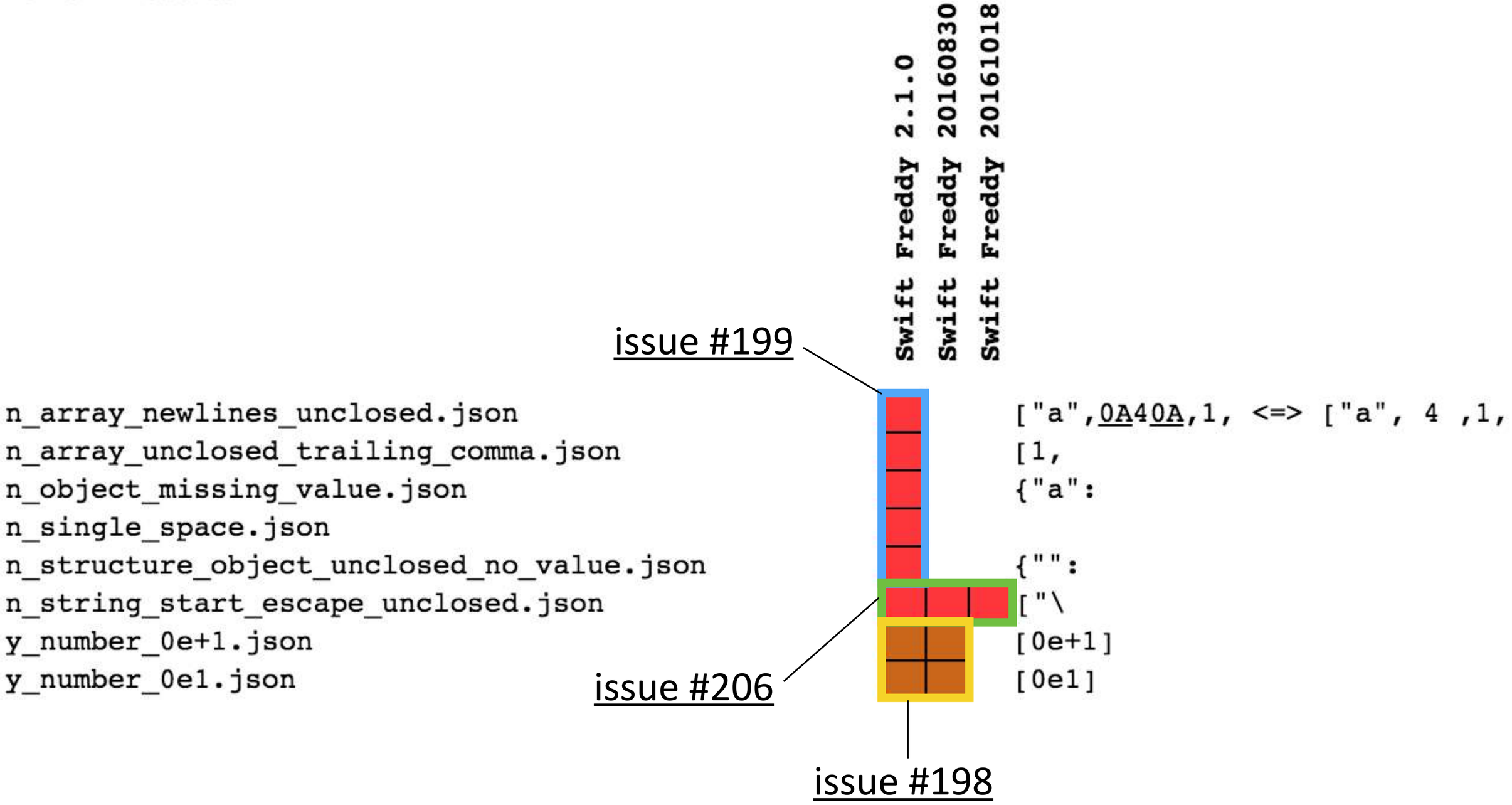
Apple (NS)JSONSerialization

```
do {
    let a = [Double.nan]
    let data = try JSONSerialization.data(withJSONObject: a, options: [])
} catch let e {
}

SIGABRT

*** Terminating app due to uncaught exception 'NSInvalidArgumentException',
    reason: 'Invalid number value (NaN) in JSON write'
```

Freddy

- Cool JSON parser introduced in January 2016 by Big Nerd Ranch
- Does leverage Swift idioms and type safety

```
public enum JSON {
    case Array([JSON])
    case Dictionary([Swift.String: JSON])
    case Double(Swift.Double)
    case Int(Swift.Int)
    case String(Swift.String)
    case Bool(Swift.Bool)
    case Null
}
```


Swift Freddy 2.1.0
Swift Freddy 20160830
Swift Freddy 20161018
issue #199
issue #206
issue #198
n_array_newlines_unclosed.json
n_array_unclosed_trailing_comma.json
n_object_missing_value.json
n_single_space.json
n_structure_object_unclosed_no_value.json
n_string_start_escape_unclosed.json
y_number_0e+1.json
y_number_0e1.json
["a",0A40A,1, <=> ["a", 4 ,1,
[1,
{"a":
{"":
["\
[0e+1]
[0e1]

Bash JSON.sh

```
ESCAPE='(\\\\[^u[:cntrl:]]|\\u[0-9a-fA-F]{4})'
CHAR='[^[:cntrl:]"\\\\]'
# [:cntrl:] is a shorthand for [\x00-\x1F\x7F]
```

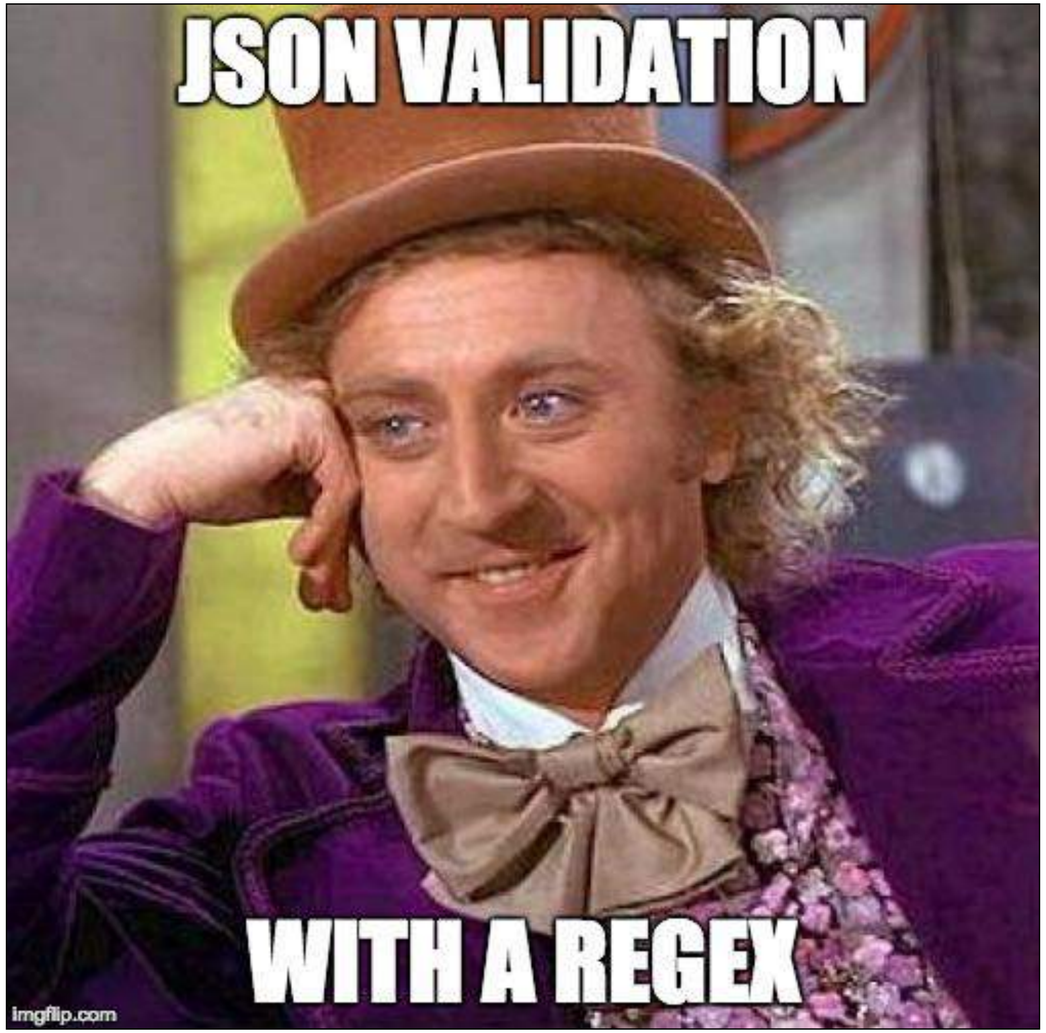
00	nul	01	soh	02	stx	03	etx	04	eot	05	enq	06	ack	07	bel
08	bs	09	ht	0a	nl	0b	vt	0c	np	0d	cr	0e	so	0f	si
10	dle	11	dc1	12	dc2	13	dc3	14	dc4	15	nak	16	syn	17	etb
18	can	19	em	1a	sub	1b	esc	1c	fs	1d	gs	1e	rs	1f	us
20	sp	21	!	22	"	23	#	24	$	25	%	26	&	27	'
28	(	29	)	2a	*	2b	+	2c	,	2d	-	2e	.	2f	/
30	0	31	1	32	2	33	3	34	4	35	5	36	6	37	7
38	8	39	9	3a	:	3b	;	3c	<	3d	=	3e	>	3f	?
40	@	41	A	42	B	43	C	44	D	45	E	46	F	47	G
48	H	49	I	4a	J	4b	K	4c	L	4d	M	4e	N	4f	O
50	P	51	Q	52	R	53	S	54	T	55	U	56	V	57	W
58	X	59	Y	5a	Z	5b	[	5c	\	5d	]	5e	^	5f	_
60	`	61	a	62	b	63	c	64	d	65	e	66	f	67	g
68	h	69	i	6a	j	6b	k	6c	l	6d	m	6e	n	6f	o
70	p	71	q	72	r	73	s	74	t	75	u	76	v	77	w
78	x	79	y	7a	z	7b	{	7c	|	7d	}	7e	~	7f	del

But **DEL 0x7F is a control char** in **Bash** but **not in JSON**!

Unescaped DEL is valid JSON.

JSON.sh won't parse [" 7F"].

Issue #46


JSON VALIDATION
WITH A REGEX
imgflip.com

(This regex was brought to you from the proving-the-naysayers-wrong department.)

126

Yes, a complete regex validation is possible.

Most modern regex implementations allow for recursive regexpressions, which can verify a complete JSON serialized structure. The json.org specification makes it quite straightforward.

```
$pcre_regex = '
  /
  (?(DEFINE)
     (?<number>   -? (?= [1-9]|0(?!\d) ) \d+ (\.\d+)? ([eE] [+-]? \d+)? )
     (?<boolean>   true | false | null )
     (?<string>    " ([^"\\\\]* | \\\\ ["\\\\bfnrt\/] | \\\\ u [0-9a-f]{4} )* " )
     (?<array>     \[  (?:  (?&json)  (?: , (?&json)  )*  )?  \s* \] )
     (?<pair>      \s* (?&string) \s* : (?&json)  )
     (?<object>    \{  (?:  (?&pair)  (?: , (?&pair)  )*  )?  \s* \} )
     (?<json>   \s* (?: (?&number) | (?&boolean) | (?&string) | (?&array) | (?&object) ) \s*
  )
  \A (?&json) \Z
  /six
';
```

It works quite well in PHP with the PCRE functions . Should work unmodified in Perl; and can certainly be adapted for other languages. Also it succeeds with the JSON test cases.

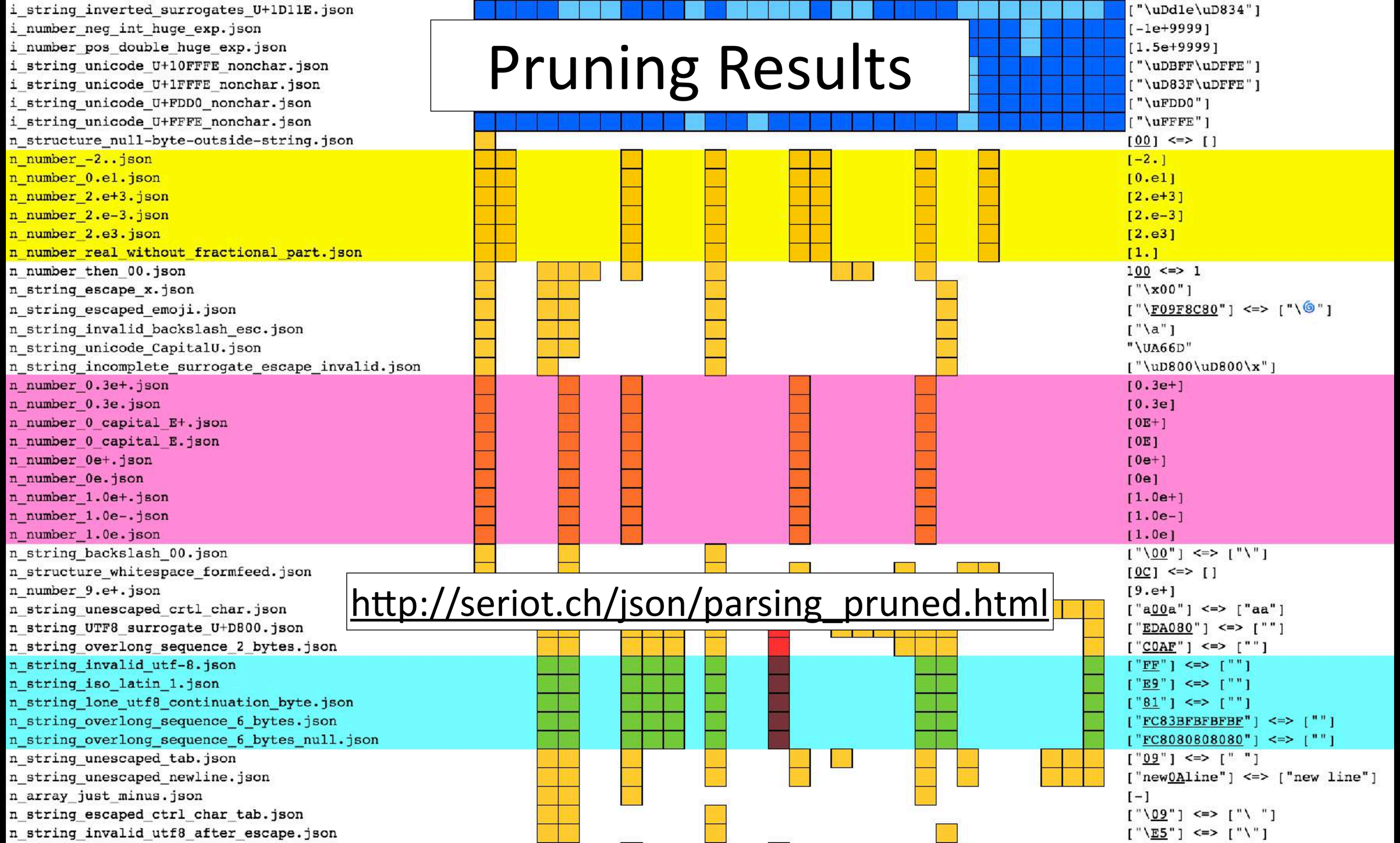
Fail to parse:

```
["\u002c"]
["\\a"]
```

Fail to reject:

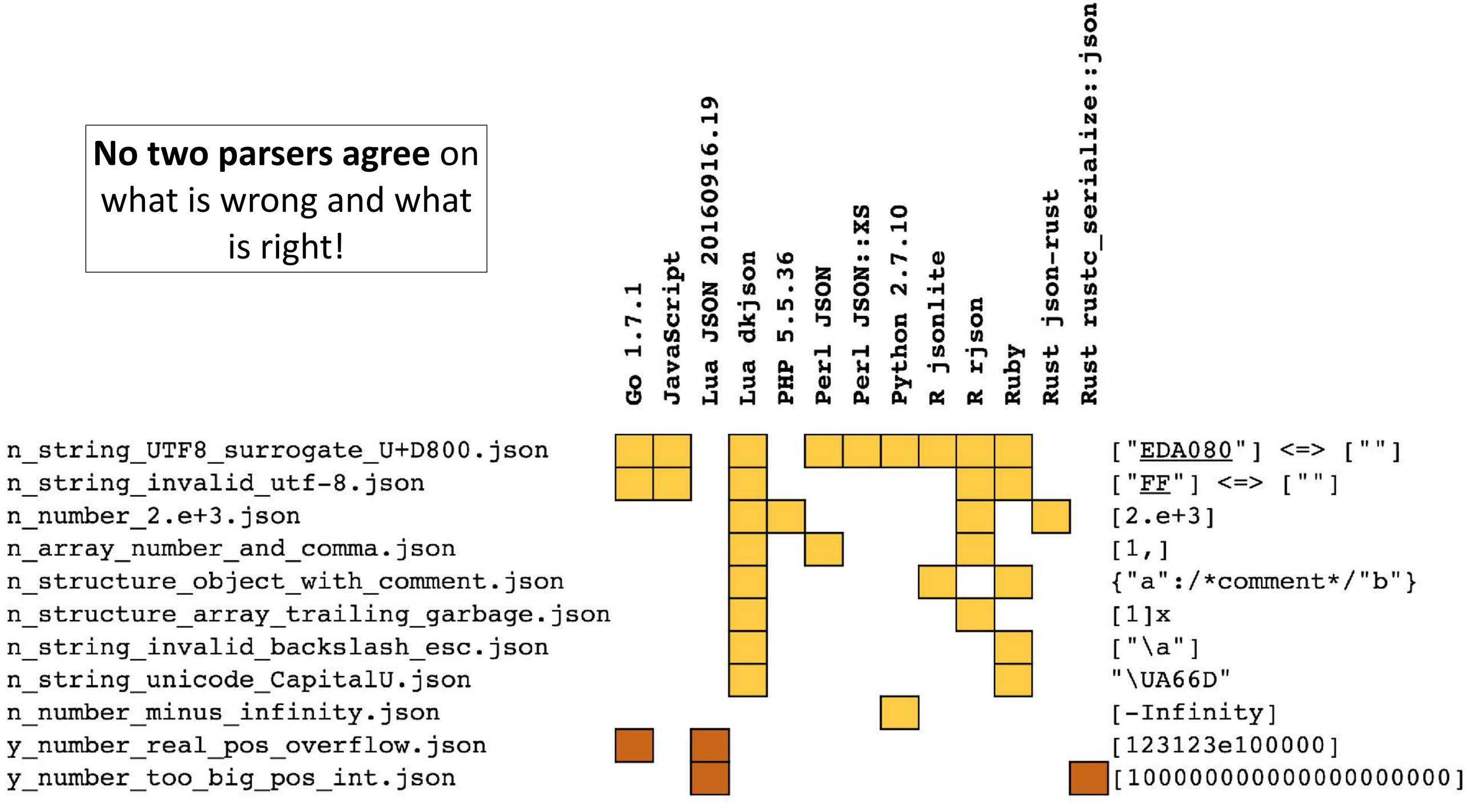
```
[True]
["09"]
```

Pruning Results



http://seriot.ch/json/parsing_pruned.html

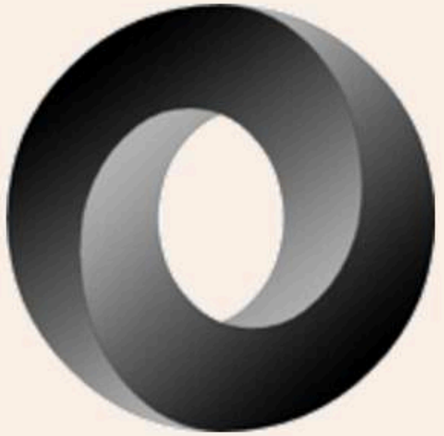
No two parsers agree on what is wrong and what is right!



REFERENCE IMPLEMENTATION?

JSON_checker

www.json.org/JSON_checker/



JSON_checker

JSON_checker is a Pushdown Automaton that very quickly determines if a JSON text is syntactically correct. It could be used to filter inputs to a system, or to verify that the outputs of a system are syntactically correct. It could be adapted to produce a very fast JSON parser.

JSON_checker is made up of these files:

Filename	Description
JSON_checker.c	The JSON_checker.
JSON_checker.h	The JSON_checker header file.
main.c	A sample application.

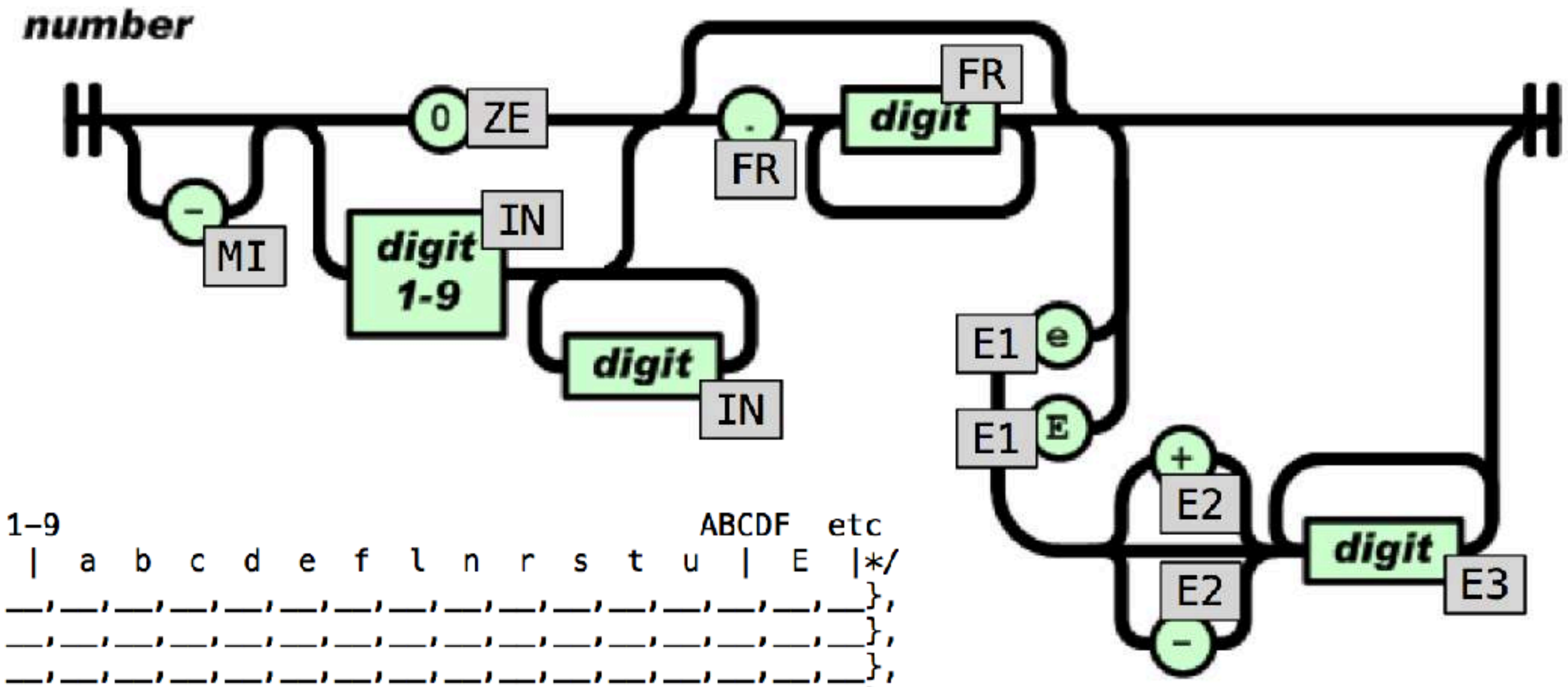
- `JSON_Checker.c` **will parse** `[1.]` and `[0.e1]`, which do not match JSON grammar.

- `JSON_Checker.c` **will reject** `[0e1]`, which is a perfectly valid JSON number

- Has the bug spread to other parsers? `[1.]` is also parsed by Obj-C TouchJSON, PHP, R rjson, Rust json-rust, Bash JSON.sh, C jsmn and Lua dkjson.

Bug #1

rejection of 0e1



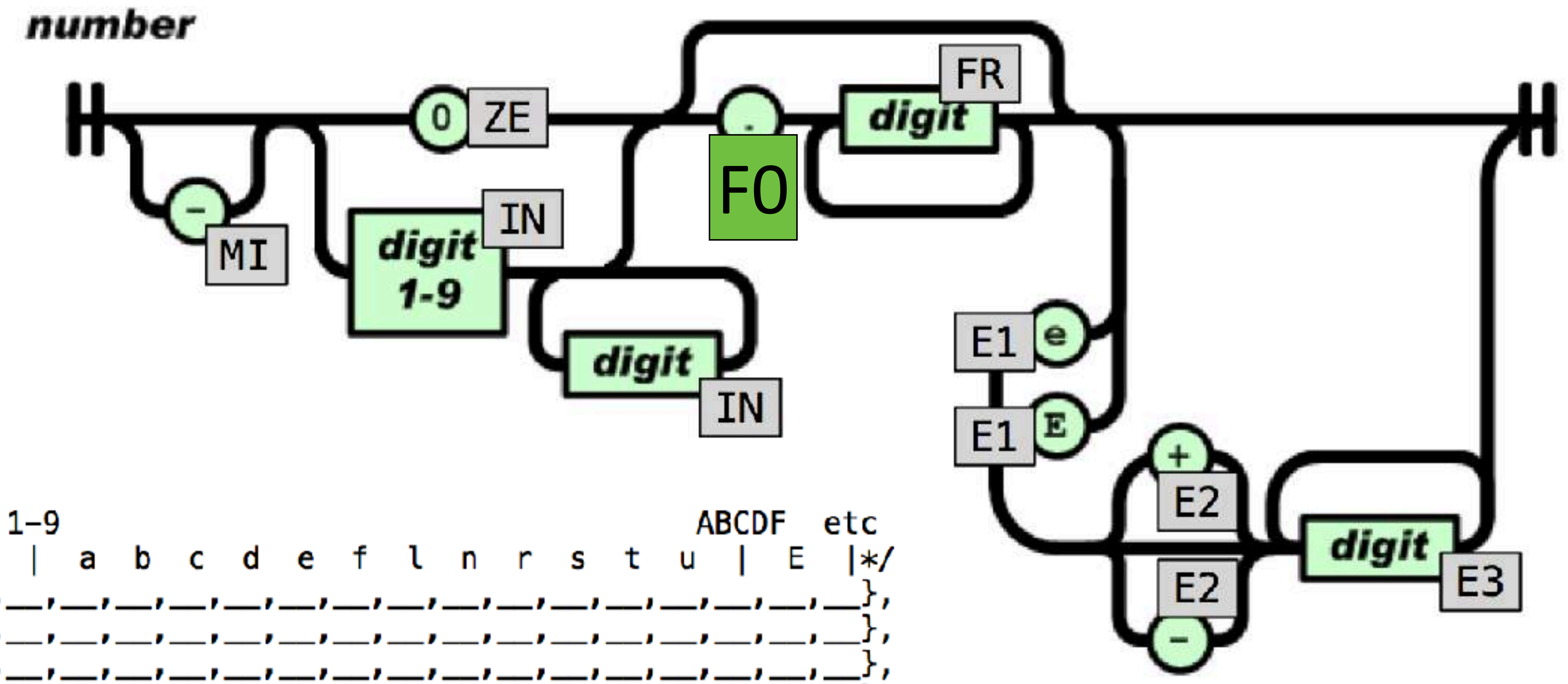

```
                 white                                      1-9                                   ABCDF  etc
             space |  {  }  [  ]  :  ,  "  \  /  +  -  .  0  |  a  b  c  d  e  f  l  n  r  s  t  u  |  E  |*/
/*start  GO*/ {GO,GO,-6,__,-5,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__},
/*ok     OK*/ {OK,OK,__,-8,__,-7,__,-3,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__},
/*object OB*/ {OB,OB,__,-9,__,__,__,__,ST,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__},
/*key    KE*/ {KE,KE,__,__,__,__,__,__,ST,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__},
/*colon  CO*/ {CO,CO,__,__,__,__,-2,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__},
/*value  VA*/ {VA,VA,-6,__,-5,__,__,__,ST,__,__,__,MI,__,ZE,IN,__,__,__,__,__,F1,__,N1,__,__,T1,__,__,__,__},
/*array  AR*/ {AR,AR,-6,__,-5,-7,__,__,ST,__,__,__,MI,__,ZE,IN,__,__,__,__,__,F1,__,N1,__,__,T1,__,__,__,__},
/*string ST*/ {ST,__,ST,ST,ST,ST,ST,ST,-4,ES,ST,ST,ST,ST,ST,ST,ST,ST,ST,ST,ST,ST,ST,ST,ST,ST,ST,ST,ST,ST,ST},
/*escape ES*/ {__,__,__,__,__,__,__,__,ST,ST,ST,__,__,__,__,__,__,ST,__,__,__,ST,__,ST,ST,__,ST,U1,__,__,__},
/*u1     U1*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,U2,U2,U2,U2,U2,U2,U2,U2,__,__,__,__,__,__,U2,U2,__},
/*u2     U2*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,U3,U3,U3,U3,U3,U3,U3,U3,__,__,__,__,__,__,U3,U3,__},
/*u3     U3*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,U4,U4,U4,U4,U4,U4,U4,U4,__,__,__,__,__,__,U4,U4,__},
/*u4     U4*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,ST,ST,ST,ST,ST,ST,ST,ST,__,__,__,__,__,__,ST,ST,__},
/*minus  MI*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,ZE,IN,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__},
-/*zero   ZE*/ {OK,OK,__,-8,__,-7,__,-3,__,__,__,__,__,FR,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__},
+/*zero   ZE*/ {OK,OK,__,-8,__,-7,__,-3,__,__,__,__,__,FR,__,__,__,__,__,__,E1,__,__,__,__,__,__,__,__,E1,__},
/*int    IN*/ {OK,OK,__,-8,__,-7,__,-3,__,__,__,__,__,FR,IN,IN,__,__,__,__,E1,__,__,__,__,__,__,__,__,E1,__},
/*frac   FR*/ {OK,OK,__,-8,__,-7,__,-3,__,__,__,__,__,__,FR,FR,__,__,__,__,E1,__,__,__,__,__,__,__,__,E1,__},
/*e      E1*/ {__,__,__,__,__,__,__,__,__,__,__,E2,E2,__,E3,E3,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__},
/*ex     E2*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,E3,E3,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__},
/*exp    E3*/ {OK,OK,__,-8,__,-7,__,-3,__,__,__,__,__,__,E3,E3,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__},
/*tr     T1*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,T2,__,__,__,__,__,__},
/*tru    T2*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,T3,__,__,__},
/*true   T3*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,OK,__,__,__,__,__,__,__,__,__,__},
/*fa     F1*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,F2,__,__,__,__,__,__,__,__,__,__,__,__,__,__},
/*fal    F2*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,F3,__,__,__,__,__,__,__,__},
/*fals   F3*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,F4,__,__,__,__,__},
/*false  F4*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,OK,__,__,__,__,__,__,__,__,__,__},
/*nu     N1*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,N2,__,__,__},
```

add missing transitions

ZE -> eE -> E1

Bug #2

acceptance of [1.]




```
                white                                      1-9                                   ABCDF  etc
            space |  {  }  [  ]  :  ,  "  \  /  +  -  .  0  |  a  b  c  d  e  f  l  n  r  s  t  u  |  E  |*/
/*start  GO*/ {GO,GO,-6,__,-5,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__},
/*ok     OK*/ {OK,OK,__,-8,__,-7,__,-3,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__},
/*object OB*/ {OB,OB,__,-9,__,__,__,__,ST,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__},
/*key    KE*/ {KE,KE,__,__,__,__,__,__,ST,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__},
/*colon  CO*/ {CO,CO,__,__,__,__,-2,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__},
/*value  VA*/ {VA,VA,-6,__,-5,__,__,__,ST,__,__,__,MI,__,ZE,IN,__,__,__,__,__,F1,__,N1,__,__,T1,__,__,__,__},
/*array  AR*/ {AR,AR,-6,__,-5,-7,__,__,ST,__,__,__,MI,__,ZE,IN,__,__,__,__,__,F1,__,N1,__,__,T1,__,__,__,__},
/*string ST*/ {ST,__,ST,ST,ST,ST,ST,ST,-4,ES,ST,ST,ST,ST,ST,ST,ST,ST,ST,ST,ST,ST,ST,ST,ST,ST,ST,ST,ST,ST,ST},
/*escape ES*/ {__,__,__,__,__,__,__,__,ST,ST,ST,__,__,__,__,__,__,ST,__,__,__,ST,__,ST,ST,__,ST,U1,__,__,__},
/*u1     U1*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,U2,U2,U2,U2,U2,U2,U2,U2,__,__,__,__,__,__,U2,U2,__},
/*u2     U2*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,U3,U3,U3,U3,U3,U3,U3,U3,__,__,__,__,__,__,U3,U3,__},
/*u3     U3*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,U4,U4,U4,U4,U4,U4,U4,U4,__,__,__,__,__,__,U4,U4,__},
/*u4     U4*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,ST,ST,ST,ST,ST,ST,ST,ST,__,__,__,__,__,__,ST,ST,__},
/*minus  MI*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,ZE,IN,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__},
-/*zero  ZE*/ {OK,OK,__,-8,__,-7,__,-3,__,__,__,__,__,FR,__,__,__,__,__,__,E1,__,__,__,__,__,__,__,__,E1,__},
+/*zero  ZE*/ {OK,OK,__,-8,__,-7,__,-3,__,__,__,__,__,F0,__,__,__,__,__,__,E1,__,__,__,__,__,__,__,__,E1,__},
-/*int   IN*/ {OK,OK,__,-8,__,-7,__,-3,__,__,__,__,__,FR,IN,IN,__,__,__,__,E1,__,__,__,__,__,__,__,__,E1,__},
+/*int   IN*/ {OK,OK,__,-8,__,-7,__,-3,__,__,__,__,__,F0,IN,IN,__,__,__,__,E1,__,__,__,__,__,__,__,__,E1,__},
+/*frac0 F0*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,FR,FR,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__},
/*frac   FR*/ {OK,OK,__,-8,__,-7,__,-3,__,__,__,__,__,__,FR,FR,__,__,__,__,E1,__,__,__,__,__,__,__,__,E1,__},
/*e      E1*/ {__,__,__,__,__,__,__,__,__,__,__,E2,E2,__,E3,E3,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__},
/*ex     E2*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,E3,E3,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__},
/*exp    E3*/ {OK,OK,__,-8,__,-7,__,-3,__,__,__,__,__,__,E3,E3,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__},
/*tr     T1*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,T2,__,__,__,__,__,__},
/*tru    T2*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,T3,__,__,__},
/*true   T3*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,OK,__,__,__,__,__,__,__,__,__,__},
/*fa     F1*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,F2,__,__,__,__,__,__,__,__,__,__,__,__,__,__},
/*fal    F2*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,F3,__,__,__,__,__,__,__,__},
/*fals   F3*/ {__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,F4,__,__,__,__,__},
```

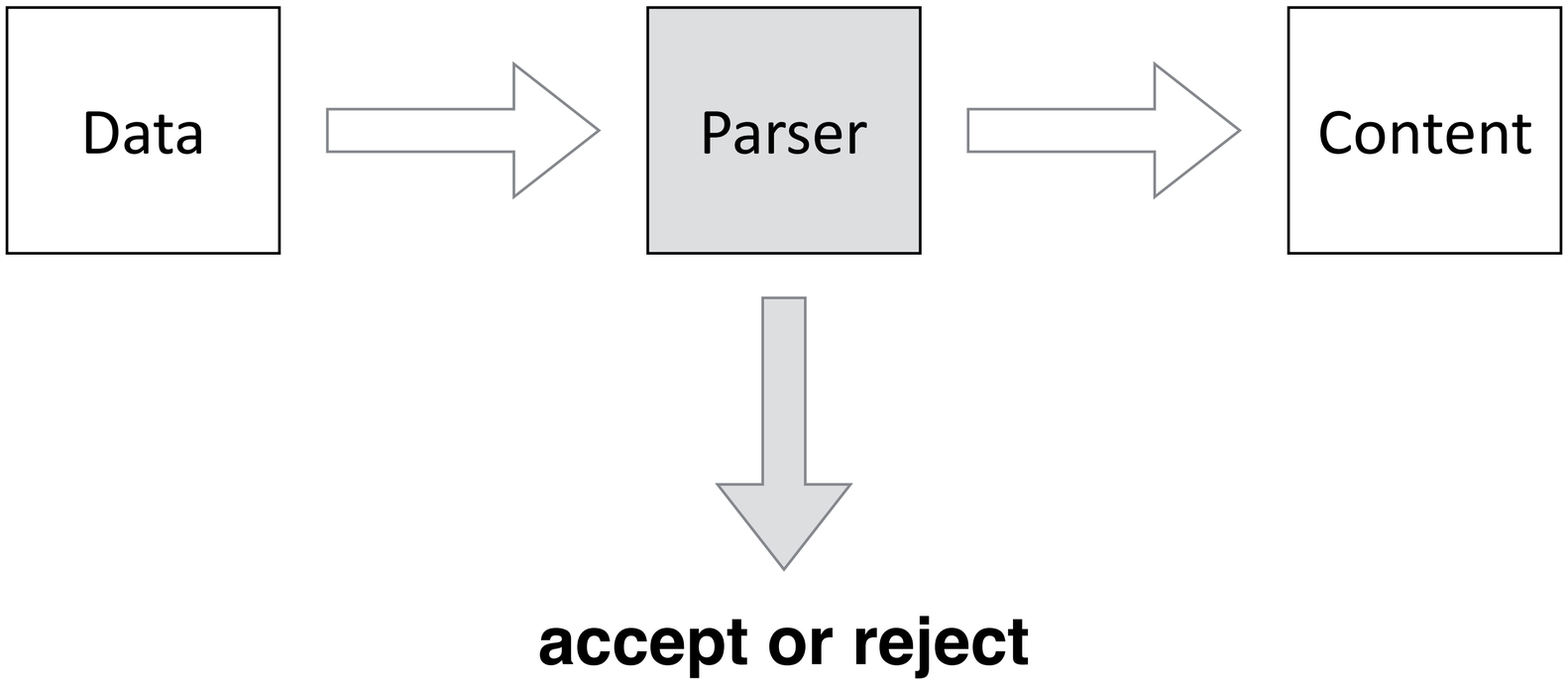
require digit after dot

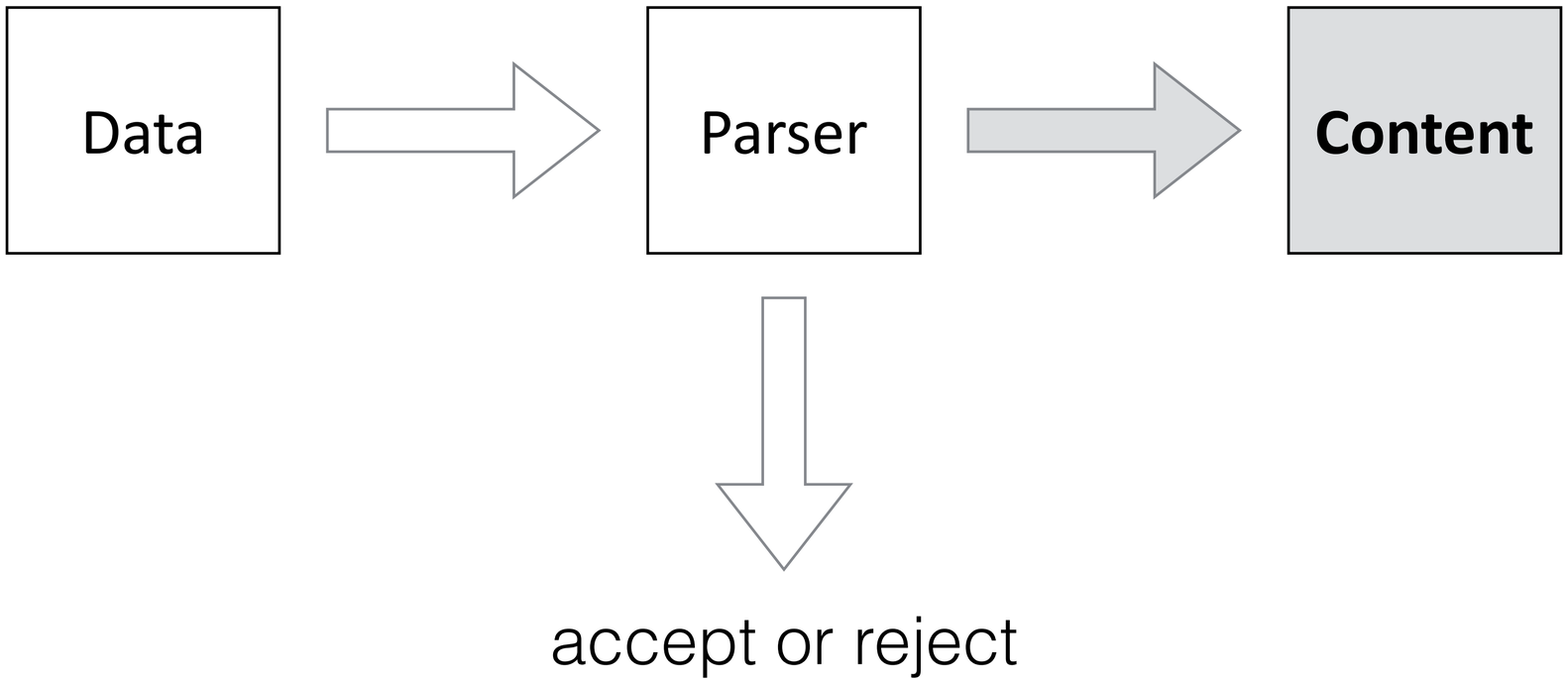
add missing state F0

update transition

IN -> . -> F0

PARSING OUTPUT


Data
Parser
Content
accept or reject


Data
Parser
Content
accept or reject

{"C3A9:"NFC","65CC81":"NFD"} keys are NFC and NFD representations of "é"
{"C3A9:"NFC","65CC81":"NFD"}
{"C3A9:"NFC"} Swift parsers

{"a":1,"a":2}
{"a":2} (Freddy, SBJSON, Go, Python, JavaScript, Ruby, Rust, Lua dksjon)
{"a":1} (Obj-C Apple NSJSONSerialization, Swift Apple JSONSerialization, Swift Freddy)
{"a":1,"a":2} (cJSON, R, Lua JSON)

{"a":1,"a":1}
{"a":1}
{"a":1,"a":1} (cJSON, R and Lua JSON)

{"a":0,"a":-0}
{"a":0}
{"a":-0} (Obj-C JSONKit, Go, JavaScript, Lua)
{"a":0, "a":0} (cJSON, R)

Objects

Numbers (Precision Issues)

1.0000000000000005

-> float 1.0 or 1.0000000000000005

1E-999

-> float, double 0.0, string "1E-999" or error

1000000000000000999

-> double, unsigned long long or string

-> cJSON will yield new number 1000000000000000**2048**

Problem when two software components parse the same file differently

Strings

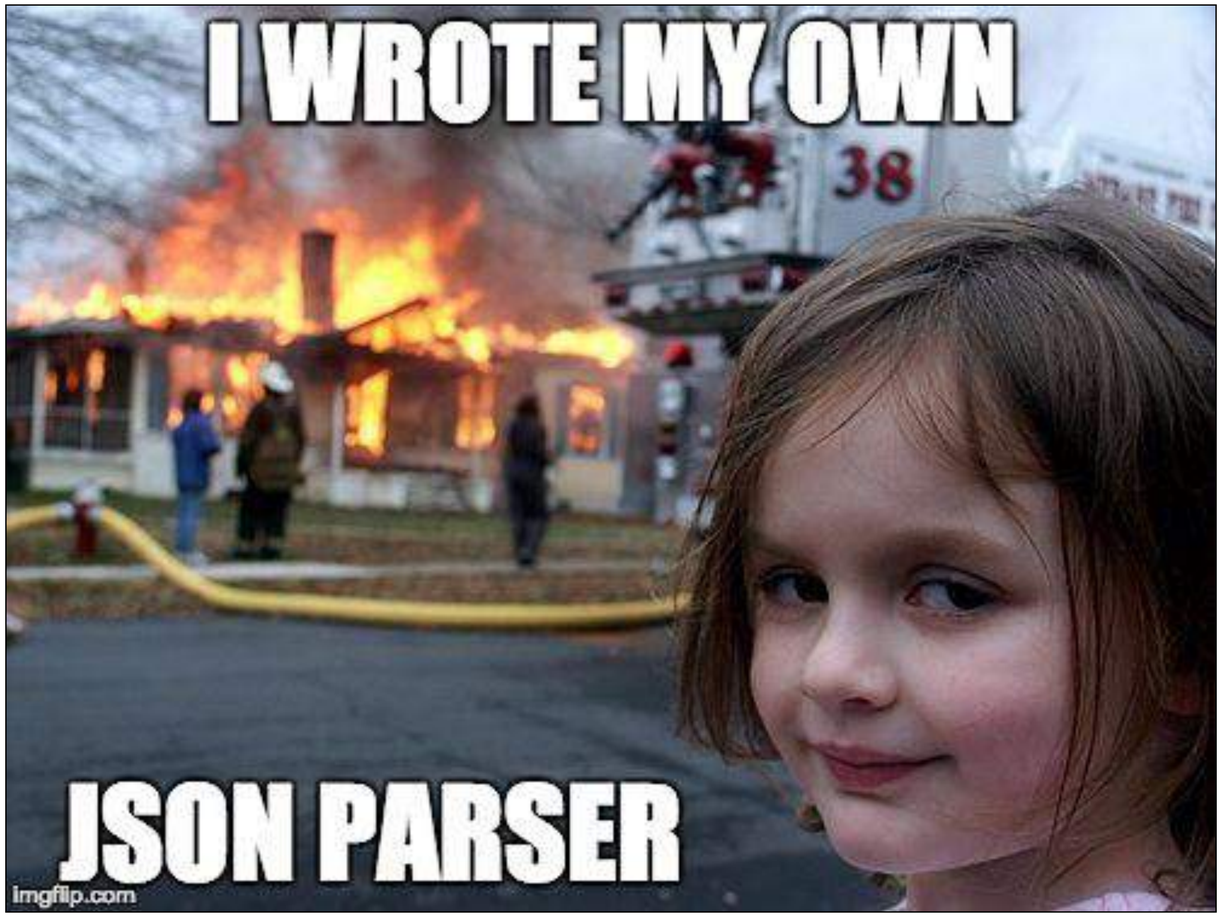
["A\u0000B"]
sometimes becomes **["A"]**

["\uD800"] is the u-escaped form of U+D800, an invalid lone surrogate
["\uD800"]
["EFBFBD"] U+FFFD REPLACEMENT CHARACTER
["EDA080"] (UTF-8 for U+D800)

["EDA080"] raw UTF-8 bytes, invalid UTF-8, for U+D800, invalid lone surrogate
["EDA080"] (cJSON, R rjson and jsonlite, Lua JSON, Lua dkjson and Ruby)
["EFBFBDEFBFBDEFBFBD"] (Go, JS) 3 replacement characters
["\ud800"] (Python 2)
UnicodeDecodeError (Python 3)

["\uD800\uD800"]
["\U00010000"] (R rjson)
["F0908080"] (Ruby) WTF??

valid vector, invalid payload


I WROTE MY OWN
JSON PARSER
imgflip.com

nst / STJSON

Code Issues 0 Pull requests 0 Projects 0 Wiki

A JSON Parser in Swift 3 compliant with RFC 7159 — Edit

3 commits 1 branch 0 releases

Branch: master New pull request

nst added reference to the article

STJSON.xcodeproj	first commit
STJSON	first commit
STJSONTests	first commit
.gitignore	Initial commit
LICENSE	Initial commit
README.md	added reference to the article

README.md

STJSON

A JSON Parser in Swift 3 compliant with RFC 7159

STJSON was written along with the article Parsing JSON is a Minefield.

Basic usage:

```
var p = STJSONParser(data: data)

do {
    let o = p.parse()
} catch let e {
    print(e)
}
```

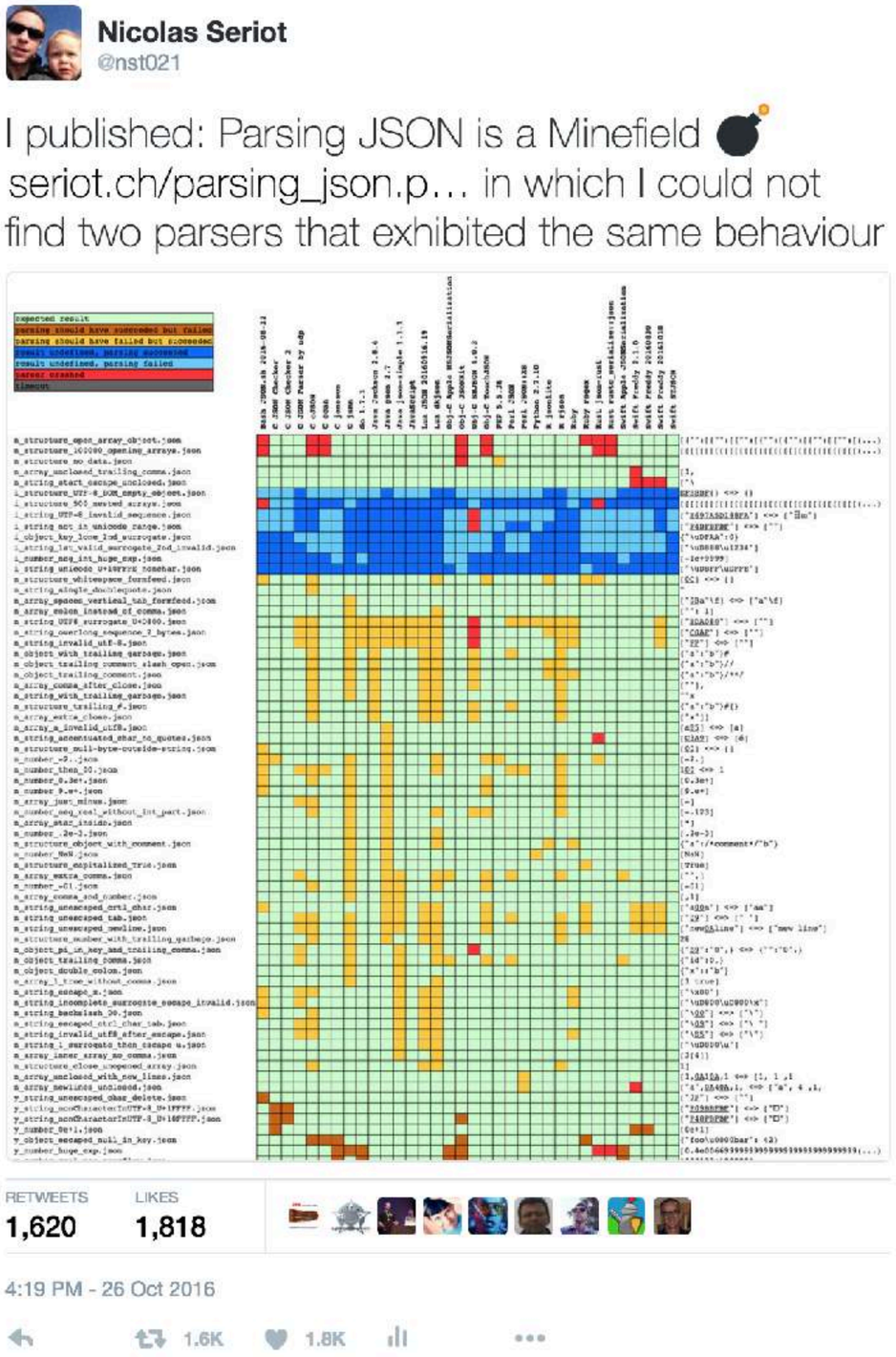
Instantiation with options:

```
    var p = STJSON(data:data,
                   maxParserDepth:1024,
                   options:[.useUnicodeReplacementCharacter])
```

Conclusion

- **No two parsers behave the same way**
- RFC 7159 is underspecified, **hidden complexity**
- No official test suite
- Ref. implementation `JSON_Checker.c` was buggy until 2016-11
- **JSON is not the harmless, general purpose data-interchange format as many do believe**
- Fragile standards often prevail (HTML, CSS, JavaScript, JSON)

FOLLOW UP


Nicolas Seriot
@nst021
I published: Parsing JSON is a Minefield
seriot.ch/parsing_json.p… in which I could not
find two parsers that exhibited the same behaviour
RETWEETS
1,620
LIKES
1,818
4:19 PM - 26 Oct 2016
1.6K
1.8K


Hacker News new | threads | comments | show | ask | jobs | submit
* Parsing JSON is a Minefield (seriot.ch)
546 points by beefburger 1 day ago | hide | past | web 284 comments favorite


reddit
PROGRAMMING
comments
other discussions (2)
740
Parsing JSON is a Minefield
(seriot.ch)
submitted 1 day ago by nst021
172 comments share save hide delete nsfw

T + 48h

Funny Issues

Can't checkout repository on Windows #17

Open **JamesNK** opened this issue 10 hours ago · 0 comments

JamesNK commented 10 hours ago

Two files contain invalid characters:

```
Cloning into 'JSONTestSuite'...
remote: Counting objects: 1241, done.
remote: Compressing objects: 100% (442/442), done.
remote: Total 1241 (delta 199), reused 1210 (delta 168), pack-reused 0
Receiving objects: 100% (1241/1241), 34.76 MiB | 1.41 MiB/s, done.
Resolving deltas: 100% (199/199), done.
Checking connectivity... done.
error: unable to create file test_parsing/n_structure_<.>.json (Invalid argument)
error: unable to create file test_parsing/n_structure_<null>.json (Invalid argument)
Checking out files: 100% (910/910), done.
fatal: unable to checkout working tree
warning: Clone succeeded, but checkout failed.
You can inspect what was checked out with 'git status'
and retry the checkout with 'git checkout -f HEAD'
```

crystal-lang / crystal

Watch 326 | Star 6,390 | Fork 524

Code | Issues 283 | Pull requests 52 | Projects 0 | Wiki | Pulse | Graphs

JSON: some fixes based on http://seriot.ch/parsing_json.html

Browse files

```
- Correctly error on control characters inside strings
- Prevent stack overflow for too nested structures
- Minor parsing fixes
```

everyone fixing their parser overnight

master

asterite committed 4 hours ago — 1 parent 61a3b69 — commit 7eb738f550818825786e90389ac84d2a2eb13e13

Showing 6 changed files with 103 additions and 30 deletions.

Unified | Split

13 spec/std/json/parser_spec.cr — View

```
@@ -52,6 +52,19 @@ describe JSON::Parser do
```

```
52    it_raises_on_parse "[0]1"
53    it_raises_on_parse "[0] 1 "
54    it_raises_on_parse "[\"\\u123z\"]"
```

```
52    it_raises_on_parse "[0]1"
53    it_raises_on_parse "[0] 1 "
54    it_raises_on_parse "[\"\\u123z\"]"
55  + it_raises_on_parse "[1 true]"
56  + it_raises_on_parse %({"foo": 1 "bar": 2})
57  + it_raises_on_parse %([2.])
58  + it_raises_on_parse %("hello\nworld")
59  + it_raises_on_parse %("\\u201cello\nworld")
60  + it_raises_on_parse %("hello\tworld")
61  + it_raises_on_parse %("\\u201cello\tworld")
```

☆ **Kevin Ballard <kevin@sb.org>** Inbox - nicolas@s

Message on seriot.ch from Kevin Ballard

To: Nicolas Seriot <nicolas@seriot.ch>

Reply-To: Kevin Ballard <kevin@sb.org>

I just read through http://seriot.ch/parsing_json.html and it looks like a great article. I'm writing you because I have my own pure-Swift parser that I wrote a while back and was kind of bummed that you didn't include it in your article. I haven't yet run your test suite against my parser, but I will be doing so as soon as I have the time. If you're interested, my parser is https://github.com/postmates/PMJSON. Offhand I expect that it will be slightly liberal in what it accepts (since I based it off of json.org rather than RFC 7159) but should never crash.

-Kevin Ballard

--
Browser: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_1) AppleWebKit/602.2.14 (KHTML, like Gecko) Version/10.0.1 Safari/602.2.14
IP Address: 38.122.6.194

☆ **Nicolas Seriot <nicolas@seriot.ch>**

Re: Message on seriot.ch from Kevin Ballard

To: Kevin Ballard <kevin@sb.org>

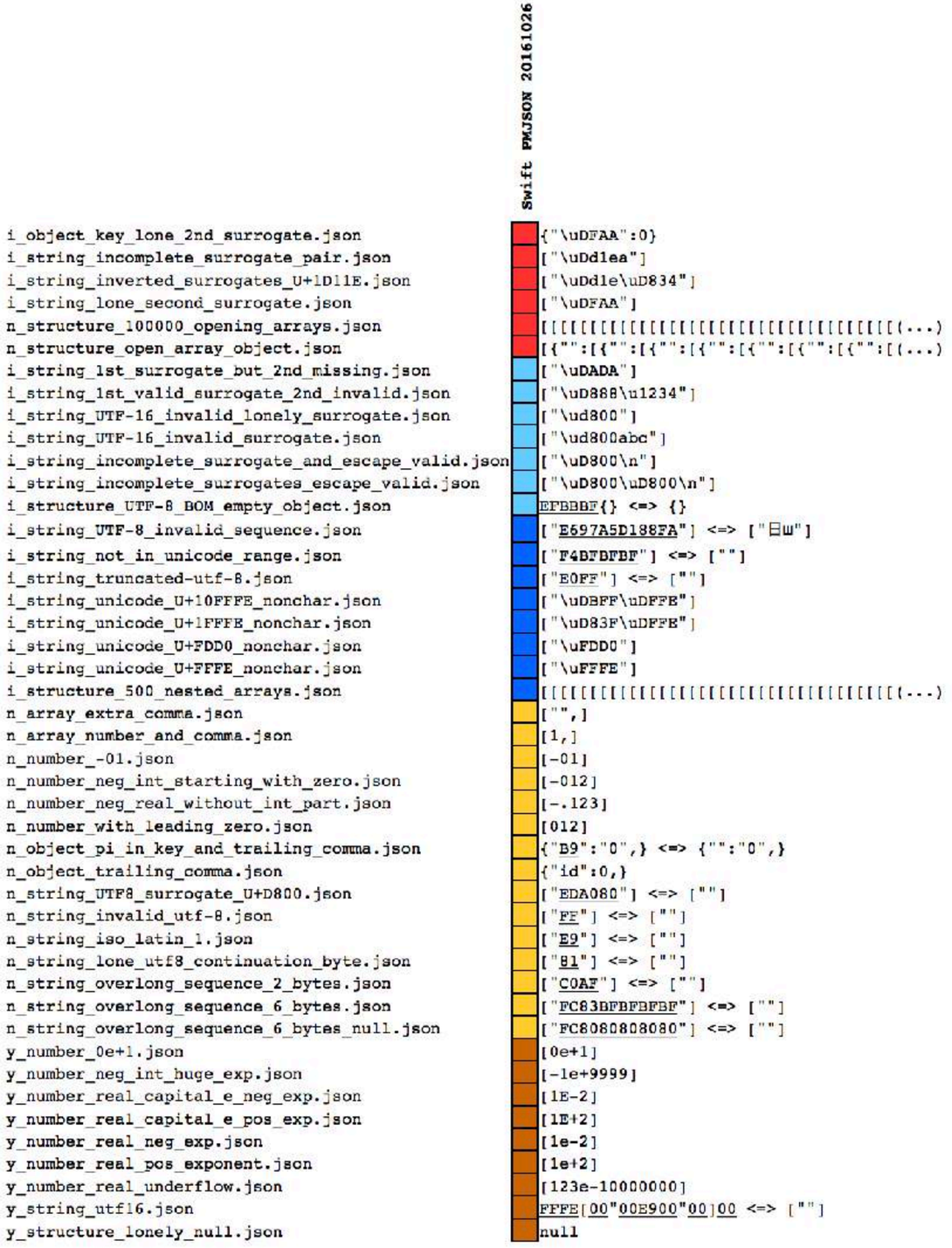
Hello Kevin,

Thank you for reading the article and writing to me.

My mistake, how does it come that I did not know PMJSON?!

Here are the results that the automated test yields.

Nicolas



☆ **Kevin Ballard <kevin@sb.org>**

Re: Message on seriot.ch from Kevin Ballard

To: Nicolas Seriot <nicolas@seriot.ch>

Inbox - nicolas@seriot.ch 02:11

KB

Hi Nicolas,

Thanks for the info. PMJSON definitely wasn't handling lone trail surrogates (but it was handling lone leading surrogates), which is a surprising oversight on my part. As for the other 2 crashes, it's simply blowing the stack. In my tests, it dies on the 87367th stack frame. I can't decide if the right solution here is to force a depth limit (if so, what limit is appropriate?), or if I should rework my decoder to reify the stack into an array instead of recursing (which would then handle anything that fits in available memory).

Regarding the trailing comma ones, the parser actually has a strict mode that turns those into errors, you just didn't enable it (it's an optional argument to JSON.decode). For the invalid numbers, I'm actually surprised it parses those, I thought I had made it strictly follow the grammar defined at json.org. Same goes for the inability to parse things like [1E-2]. Clearly I'm going to have to revisit that section of the parser. I'm also surprised that large sample.json file from https://code.google.com/archive/p/json-test-suite/ doesn't seem to test the scientific notation issues.

For the invalid UTF-8 ones, I'm not convinced those should be considered an error. JSON is a text format, not a binary format. RTFC 7159 says that JSON text SHALL be encoded as UTF-8, UTF-16, or UTF-32, but that's an interoperability concern. The JSON format itself is defined using unicode characters, and interpreting a byte stream as UTF-8 is outside the scope of the language spec. And in fact, PMJSON is actually a parser over unicode characters (check out JSONParser, it operates on any Sequence of UnicodeScalars) but JSON.decode(someData) is a convenience wrapper that decodes the data as UTF-8 (replacing ill-formed byte sequences with U+FFFD) for you. Also, the ISO-Latin-1 test is indistinguishable from an invalid UTF-8 test.

Similarly, for the UTF-16 test, I'm not entirely convinced this should be considered a failure. The JSON spec says it SHALL be encoded in UTF-8, UTF-16, or UTF-32, but it admits that implementations may not be able to handle anything other than UTF-8, and since the JSON spec is a text format, it doesn't really seem like a hard requirement that a parser be able to handle UTF-16. If you want to parse a UTF-16 file with PMJSON, you can always convert the Data to a String using a mechanism that handles UTF-16 before passing that String to the parser. That said, I may consider at least handling the case of a UTF-16 document with a valid BOM (I'm not really a fan of trying to detect UTF-16 without a BOM because if the UTF-16 doesn't include any characters outside the ASCII range, the byte stream is technically valid UTF-8, despite having a bunch of NULs in it).

Anyway, thanks a bunch for putting this together. I'll be working on fixing up the errors in PMJSON as soon as possible.

-Kevin Ballard

New Friends

Allen S. Rout <allen.rout@gmail.com>
Message on seriot.ch from Allen S. Rout
To: Nicolas Seriot <nicolas@seriot.ch>
Reply-To: Allen S. Rout <allen.rout@gmail.com>

Saw your JSON parsing comparison on Hacker News.

You're doing God's work.

I'd love to buy you a beverage-of-your-choice (I used to say Beer, but some folks are totally not into alcohol). If you'll send me an email address, I'll send you $12; I like the big bottle of KWAK, which costs that at my local.

http://bestbelgianspecialbeers.be/en/kwak

In any case, thanks and keep it up!

Amos <amos.gutman@sisense.com>
Inbox - nicolas@seriot.ch
Message on seriot.ch from Amos
To: Nicolas Seriot <nicolas@seriot.ch>,
Reply-To: Amos <amos.gutman@sisense.com>

hi,

i stumbled upon your work on "JSON is a minefield".

i have to personally respond to you, exemplary work!

I work at a BI company - Sisense,
we deal with parsing JSON for customers all the time and small issues like you mentioned come from time to time.

you sorted a lot for me.

I had a look at your website,
really beautiful work all around.

overall, just wanted to show some appreciation and gratitude.

Amos.

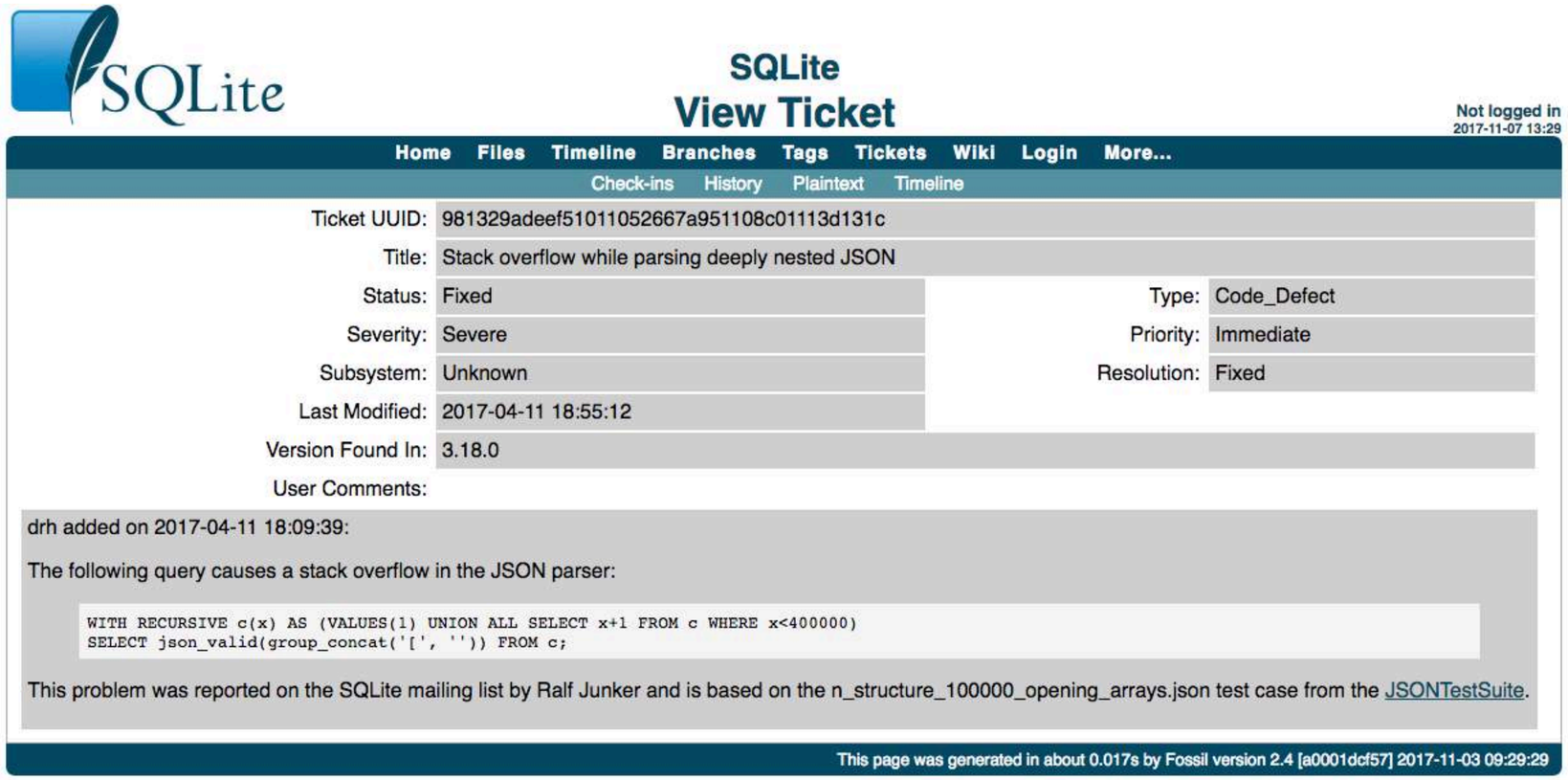
--
Browser: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36
IP Address: 217.132.29.254

Oleg Lavrovsky <Oleg@utou.ch>
Message on seriot.ch from Oleg Lavrovsky
To: Nicolas Seriot <nicolas@seriot.ch>
Reply-To: Oleg Lavrovsky <Oleg@utou.ch>

Finally! Someone who is into actual marathons, datavis and sharing wisdom on the Internet in my vicinity! Fantastic blog, visuals and attitude. Would be really glad to get to know the wise person behind them. I started a small community last year which you may relate to at soda.camp

Best wishes,
Oleg

--
Browser: Mozilla/5.0 (Linux; Android 5.1; FP2 Build/FP2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.124 Mobile Safari/537.36
IP Address: 92.105.107.210



SQLite

SQLite
View Ticket

Not logged in
2017-11-07 13:29

Home Files Timeline Branches Tags Tickets Wiki Login More...

Check-ins History Plaintext Timeline

Ticket UUID:	981329adeef51011052667a951108c01113d131c		
Title:	Stack overflow while parsing deeply nested JSON		
Status:	Fixed	Type:	Code_Defect
Severity:	Severe	Priority:	Immediate
Subsystem:	Unknown	Resolution:	Fixed
Last Modified:	2017-04-11 18:55:12		
Version Found In:	3.18.0		
User Comments:			

drh added on 2017-04-11 18:09:39:

The following query causes a stack overflow in the JSON parser:

```
WITH RECURSIVE c(x) AS (VALUES(1) UNION ALL SELECT x+1 FROM c WHERE x<400000)
SELECT json_valid(group_concat('[', '')) FROM c;
```

This problem was reported on the SQLite mailing list by Ralf Junker and is based on the n_structure_100000_opening_arrays.json test case from the JSONTestSuite.

This page was generated in about 0.017s by Fossil version 2.4 [a0001dcf57] 2017-11-03 09:29:29

SQLite

JSONTestSuite revealed a crasher in SQLite 3.18 (macOS <= 10.12)

```
$ sqlite3
SQLite version 3.16.0 2016-11-04 19:09:39
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> WITH RECURSIVE c(x) AS (VALUES(1) UNION ALL SELECT x+1 FROM c WHERE x<1000000)
SELECT json_valid(group_concat('[', '')) FROM c;
Segmentation fault: 11
```

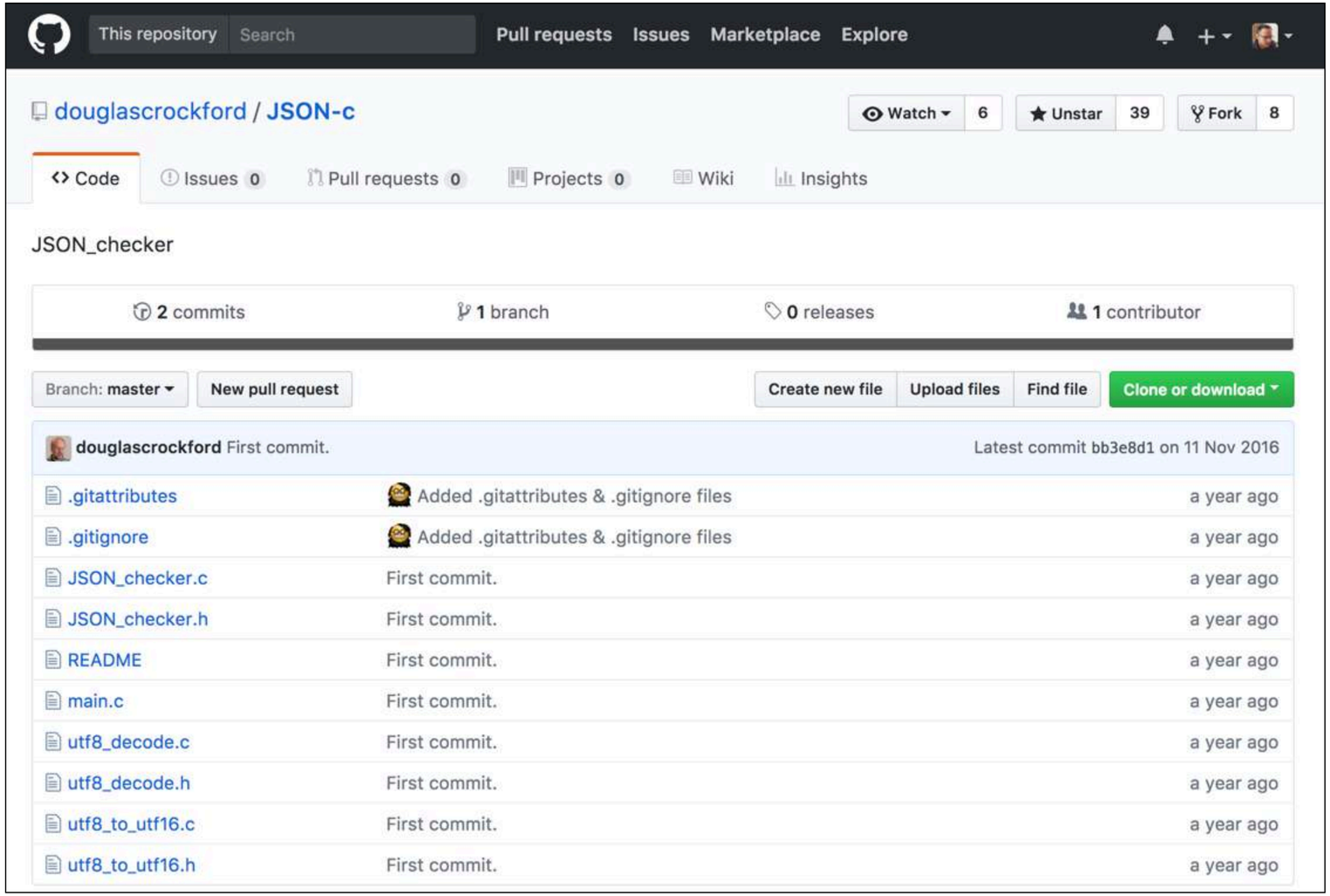
Remote Code Execution in CouchDB

Two different parsers process the same data. The Erlang JSON parser will store both values, but the Javascript parser will only store the last one -> admin in Erlang, but not in JavaScript.

```
curl -X PUT 'http://localhost:5984/_users/org.couchdb.user:oops'
--data-binary '{
  "type": "user",
  "name": "oops",
  "roles": ["_admin"],
  "roles": [],
  "password": "password"
}'
```

https://justi.cz/security/2017/11/14/couchdb-rce-npm.html

New json.org Implementation



2016-11-11

now correctly accepts `0e1`

now correctly rejects `[1.]`

Parsing JSON is a Minefield 💣

http://seriot.ch/parsing_json.php