Ecma/TC39/2013/033

2st Draft Standard ECMA-XXX

1st Edition / July 2013

The JSON Data Interchange Format

Contents

Page

Introduction

JSON is a text format, a programming language-independent data representation, allowing structured data interchange between all programming languages. It describes a syntax of braces, brackets, colons, and commas that is useful in many contexts, profiles, and applications. JSON was inspired by the object literals of JavaScript aka ECMAScript, as defined in the **ECMASCRIPT PROGRAMMING LANGUAGE STANDARD, THIRD EDITION**. It does not attempt to impose ECMAScript's internal data representations on other language. Instead, it shares a small subset of ECMAScript's textual representations with all other programming languages.

JSON is agnostic about numbers. In each programming language, there can be a variety of number types of various capacities and complements, fixed or floating, binary or decimal. That can make interchange between different programming languages difficult. JSON offers only the representation of numbers that humans use: a sequence of digits. All programming languages know how to make sense of digit sequences, even if they disagree on internal representations. That is enough to allow interchange.

It is wise to encode JSON in UNICODE, but JSON itself does not require that. JSON's only dependence on Unicode in the hex numbers used in the `\u` escapement notation. JSON can be used in contexts where there is no character encoding at all, such as paper documents and marble monuments.

Programming languages vary widely on whether they support objects, and if so, what characteristics and constraints the objects offer. The models of object systems can be wildly divergent, and are continuing to evolve. JSON instead provides a simple notation for expressing collections of name/value pairs. All programming languages will have some feature for representing such collections, which can go by names like `record`, `struct`, `dict`, `hash`, or `object`.

JSON also provides support for ordered lists of values. All programming languages will have some feature for representing such lists, which can go by names like `array`, `vector`, or `list`. Because objects and arrays can nest, trees and other complex data structures can be represented. By accepting JSON's simple convention, complex data structures can be easily interchanged between incompatible programming languages.

JSON does not support cyclic graphs, at least not directly. JSON is not indicated for applications requiring binary data.

It is expected that other standards will refer to this one, strictly adhering to the JSON format, while imposing restrictions on various encoding details. Such standards may require specific behaviours. JSON itself specifies no behaviour.

Because it is so simple, it not expected that the JSON grammar will ever change. This gives JSON, as a foundational notation, tremendous stability. JSON was first presented to the world at the `JSON.org` website in 2001. JSON stands for JavaScript Object Notation.

The JSON Data Interchange Format

1 Scope

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

2 Conformance

A conforming JSON generator or encoder will produce texts that strictly conform to the JSON grammar.

3 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 10646-1:1993*, Information Technology – Universal Multiple-Octet Coded Character Set (UCS) plus its amendments and corrigenda*

ECMA-262, *The ECMAScript Programming Language*, 3th edition (December 1999)

4 JSON Text

A JSON text is a sequence of tokens. The set of tokens includes six structural characters, strings, numbers, and three literal names.

The six structural characters:

```
{    }    [    ]    :    ,
```

Insignificant whitespace is allowed before or after any of the six structural characters.

There are three literal names:

```
true   false   null
```

5 JSON Values

A JSON value can be a string, number, object, array, `true`, `false`, or `null`.

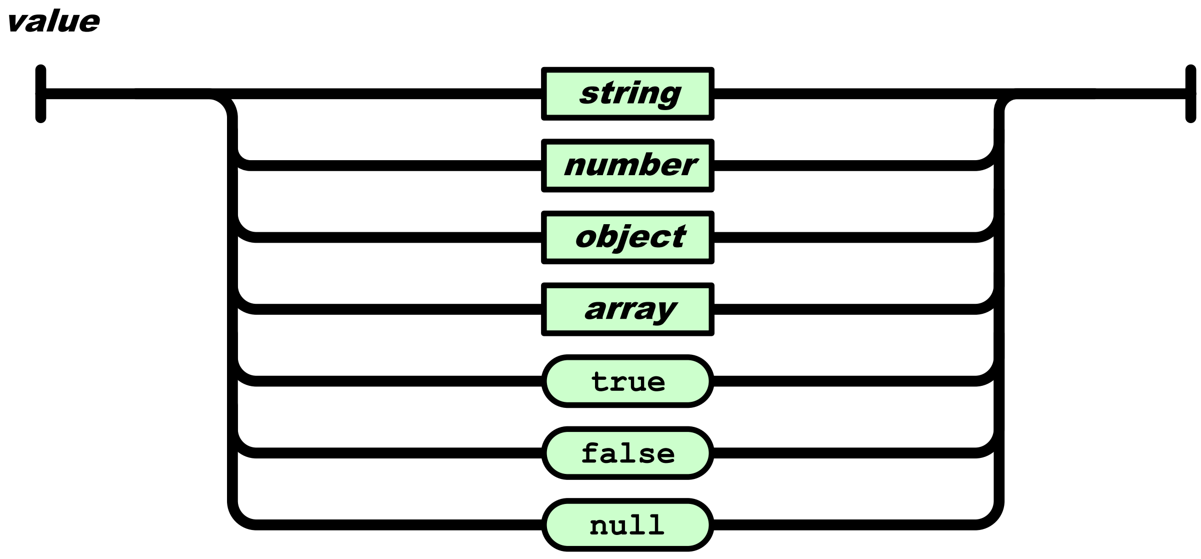


Figure 1 — value

6 Objects

An object structure is represented as a pair of curly brackets surrounding zero or more name/value pairs. A name is a string. A single colon comes after each name, separating the name from the value. A single comma separates a value from a following name.

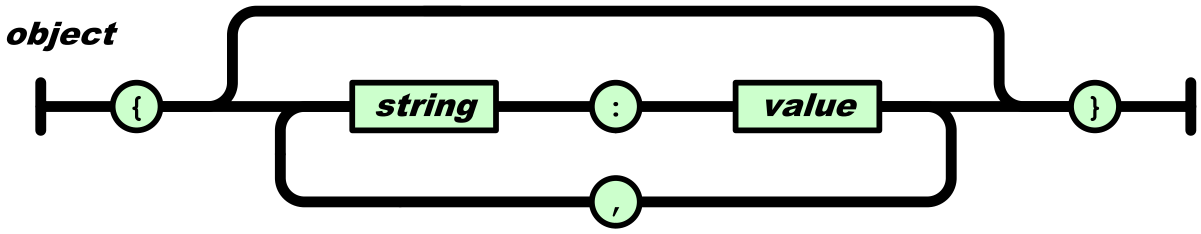


Figure 2 — object

7 Arrays

An array structure is represented as square brackets surrounding zero or more values. Elements are separated by commas.

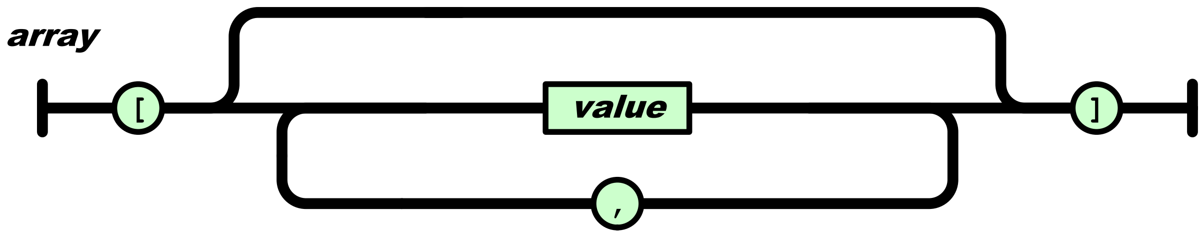


Figure 3 — array

8 Numbers

A number is represented in base 10 with no superfluous leading. It may have a preceding minus sign. It may have a `"."`-prefixed fractional part. It may have an exponent, prefixed by `e` or `E` and optionally `+` or `–`.

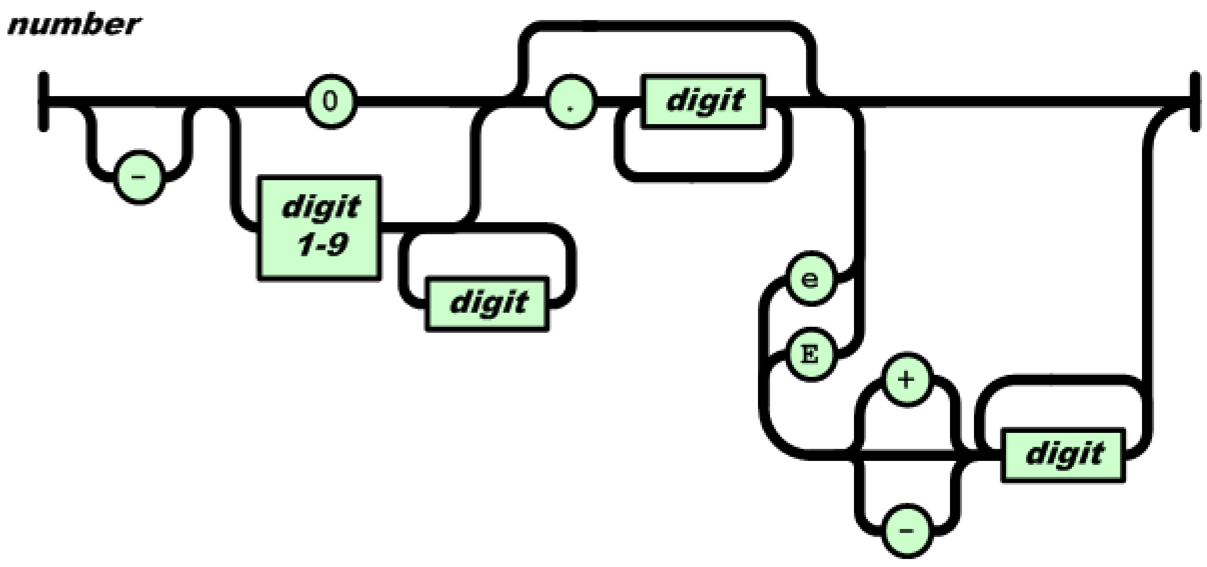


Figure 4 — number

Numeric values that cannot be represented as sequences of digits (such as `Infinity` and `NaN`) are not permitted.

9 String

The representation of strings is similar to conventions used in the C family of programming languages, a family that includes ECMAScript. A string is a sequence of characters wrapped with quotation marks. All characters may be placed within the quotation marks except for the characters that must be escaped: quotation mark, reverse solidus, and invisible control characters.

There are two-character sequence escape representations of some characters. So, for example, a string containing only a single reverse solidus character may be represented as `"\\"`.

Any character may be represented as a hexadecimal number. The meaning of such a number is determined by the Unicode Standard. If the character is in the Basic Multilingual Plane (U+0000 through U+FFFF), then it may be represented as a six-character sequence: a reverse solidus, followed by the lowercase letter u, followed by four hexadecimal digits that encode the character's Unicode code point. The hexadecimal letters A though F can be upper or lowercase. So, for example, a string containing only a single reverse solidus character may be represented as `"\u005C"`.

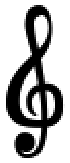
The following four cases all produce the same result:

```
"\u002F"

"\u002f"

        "\/"

        "/"
```



To escape an extended character that is not in the Basic Multilingual Plane, the character is represented as a twelve-character sequence, encoding the UTF-16 surrogate pair. So for example, a string containing only the G clef character (U+1D11E) may be represented as `"\uD834\uDD1E"`.

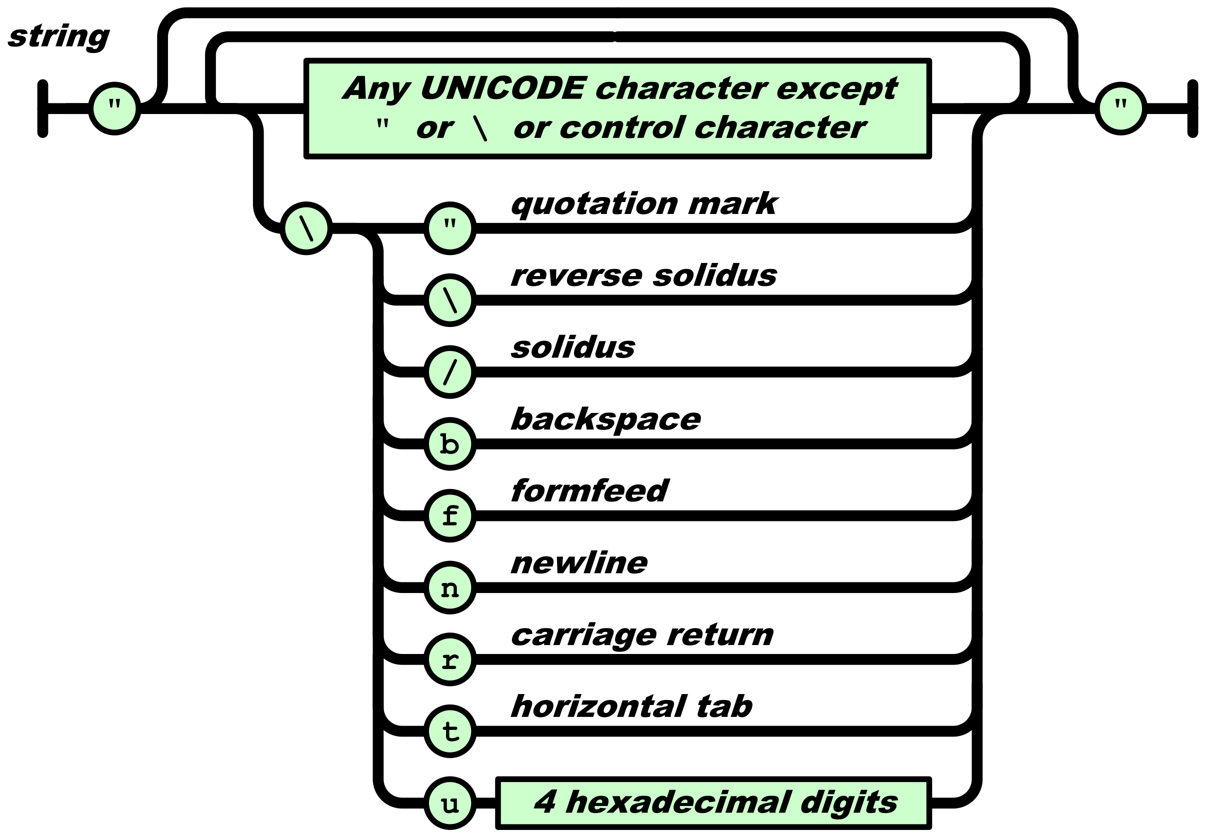


Figure 5 — string

10 Security Considerations

With any data format, it is important to encode correctly. Care must be taken when constructing JSON texts by concatenation. For example:

```
account = 4627;

comment = '","account":262';   // provided by attacker

json_text = '{"account":' + account + ',"comment":"' + comment + '"}';
```

The result will be

```
{"account":4627,"comment":"","account":262}
```

which in some situations might be seen as being the same as

```
{"comment":"","account":262}
```

This confusion allows an attacker to modify the `account` property or any other property.

It is much wiser to use JSON libraries, which are available in many forms for most programming languages, to do the encoding, avoiding the confusion hazard.

JSON is so similar to some programming languages that the native parsing ability of the language processors can be used to parse JSON texts. This should be avoided because the native parser will accept and execute code that is not JSON.

For example, ECMAScript's `eval()` function is able parse JSON text, but is can also parse programs. If an attacker can inject code into the JSON text (as we saw above), then it can compromise the system. JSON decoders should always be used instead. The web browser's `<script>` tag is an alias for the `eval()` function. It should not be used to deliver JSON text to web browsers.