CirJSON

The CirJSON Data Interchange Syntax

June 3, 2023

Contents

List of Figures

Introduction

CirJSON[1], is a text syntax that facilitates structured data interchange between all programming languages. CirJSON is a syntax of braces, brackets, colons, and commas that is useful in many contexts, profiles, and applications. CirJSON stands for Circular JavaScript Object Notation and was inspired by the original JSON syntax. However, it does not attempt to impose JSON's internal data representations on other programming languages. Instead, it shares a subset of JSON's syntax with all other programming languages. The CirJSON syntax is not a specification of a complete data interchange. Meaningful data interchange requires agreement between a producer and consumer on the semantics attached to a particular use of the CirJSON syntax. What CirJSON does provide is the syntactic framework to which such semantics can be attached, and the notion of reoccurring objects and arrays, and sometimes even circular reference in this data.

CirJSON syntax describes a sequence of Unicode code points. CirJSON also depends on Unicode in the hex numbers used in the `\u` escapement notation.

CirJSON is agnostic about the semantics of numbers. In any 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. CirJSON instead 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.

Programming languages vary widely on whether they support objects, and if so, what

[1]Pronounced /ˈsɜˑ ʤeɪ sən/, as in the first syllable of "circular" and "Jason and The Argonauts".

characteristics and constraints the objects offer. The models of object systems can be wildly divergent and are continuing to evolve. CirJSON instead provides a simple notation for expressing collections of name/value pairs, as well as a unique ID for each different object, to provide the ability to have circular relationships. Most programming languages will have some feature for representing such collections, which can go by names like `record`, `struct`, `dict`, `map`, `hash`, or `object`.

CirJSON also provides support for ordered lists of values, that, just like CirJSON's objects, receive a unique ID. 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 CirJSON's simple convention, complex data structures can be easily interchanged between incompatible programming languages.

CirJSON directly supports cyclic graphs, since this feature is the reason why it was created in the first place. CirJSON is not indicated for applications requiring binary data.

It is expected that other standards will refer to this one, strictly adhering to the CirJSON syntax, while imposing semantics interpretation and restrictions on various encoding details. Such standards may require specific behaviours. CirJSON itself specifies the behaviour of unique IDs.

Because it is so simple, it is not expected that the CirJSON grammar will ever change. This gives CirJSON, as a foundational notation, tremendous stability.

JSON was first presented to the world at the `CirJSON.org` website in 2023.

1 The CirJSON Data Interchange Syntax

1.1 Scope

CirJSON is a lightweight, text-based, language-independent syntax for defining data interchange formats. It was derived from the JSON data interchange format, and is programming language independent. CirJSON defines a small set of structuring rules for the portable representation of structured data.

The goal of this specification is only to define the syntax of valid CirJSON texts. Its intent is not to provide any semantics or interpretation of text conforming to that syntax. It also intentionally does not define how a valid CirJSON text might be internalized into the data structures of a programming language. There are many possible semantics that could be applied to the CirJSON syntax and many ways that a CirJSON text can be processed or mapped by a programming language. Meaningful interchange of information using CirJSON requires agreement among the involved parties on the specific semantics to be applied. Defining specific semantic interpretations of CirJSON is potentially a topic for other specifications.

1.2 Conformance

A conforming CirJSON text is a sequence of Unicode code points that strictly conforms to the CirJSON grammar defined by this specification. A conforming processor of CirJSON texts should not accept any inputs that are not conforming CirJSON texts. A conforming

processor may impose semantic restrictions that limit the set of conforming CirJSON texts that it will process.

1.3 CirJSON Text

A CirJSON text is a sequence of tokens formed from Unicode code points that conforms to the CirJSON value grammar. The set of tokens includes six structural tokens, strings, numbers, and three literal name tokens.

The six structural tokens:

`[`	U+005B	left square bracket
`{`	U+007B	left curly bracket
`]`	U+005D	right square bracket
`}`	U+007D	right curly bracket
`:`	U+003A	colon
`,`	U+002C	comma

These are the three literal name tokens:

`true`	U+0074	U+0072	U+0075	U+0065	
`false`	U+0066	U+0061	U+006C	U+0073	U+0065
`null`	U+006E	U+0075	U+006C	U+006C	

Insignificant whitespace is allowed before or after any token. Whitespace is any sequence of one or more of the following code points:

character tabulation	U+0009
line feed	U+000A
carriage return	U+000D
space	U+0020
line separator	U+2028
paragraph separator	U+2029

Whitespace is not allowed within any token, except that space is allowed in strings.

1.4 CirJSON Values

A CirJSON value can be an *object*, *array*, *number*, *string*, *ID*, `true`, `false`, or `null`.

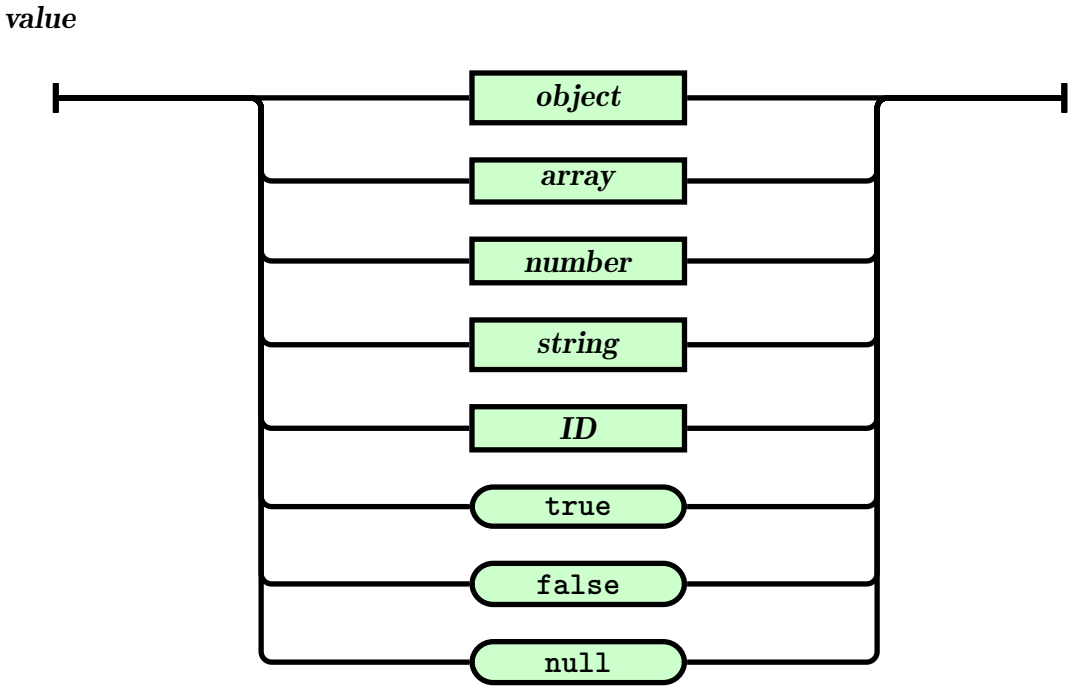

Figure 1.1: CirJSON Values

1.5 Objects

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

After the opening curly bracket, there must be the name `"__cirJsonId__"`/*ID* pair.

The CirJSON syntax has only one restriction on the strings used as names. Except for defining the ID at the start of the object, the string `"__cirJsonId__"` cannot be used as a name. Other than that, it does not require that name strings be unique, and does not assign any significance to the ordering of name/value pairs. These are all semantic considerations that may be defined by CirJSON processors or in specifications defining specific uses of CirJSON for data interchange.

object

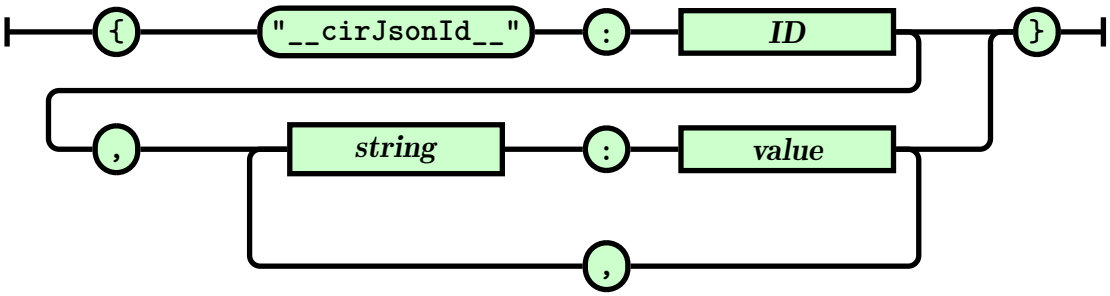

Figure 1.2: object

1.6 Arrays

An array structure is a pair of square bracket tokens surrounding its ID and zero or more *values*. The *values* and the ID are separated by commas. The first value is the array's ID. The CirJSON syntax does not define any specific meaning to the ordering of the following

values. However, the CirJSON array structure is often used in situations where there is some semantics to the ordering.

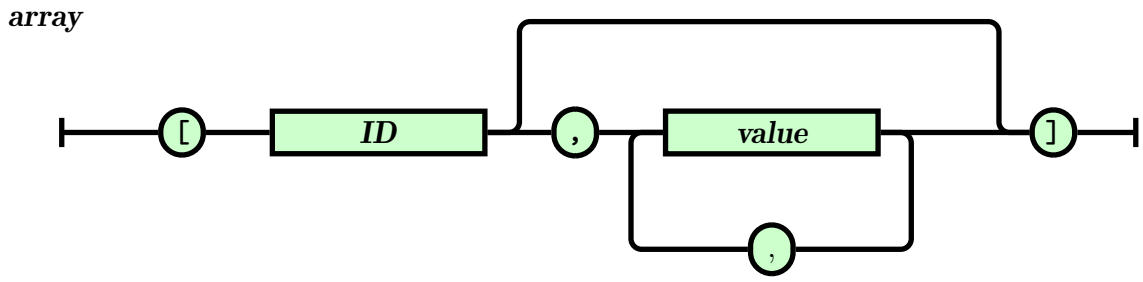

Figure 1.3: array

1.7 Numbers

A number is a sequence of decimal digits with no superfluous leading zero. It may have a preceding minus sign `-` (U+002D). It may have a fractional part prefixed by a decimal point `.` (U+002E). It may have an exponent, prefixed by `e` (U+0065) or `E` (U+0045) and optionally `+` (U+002B) or `-` (U+002D). The digits are the code points U+0030 through U+0039.

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

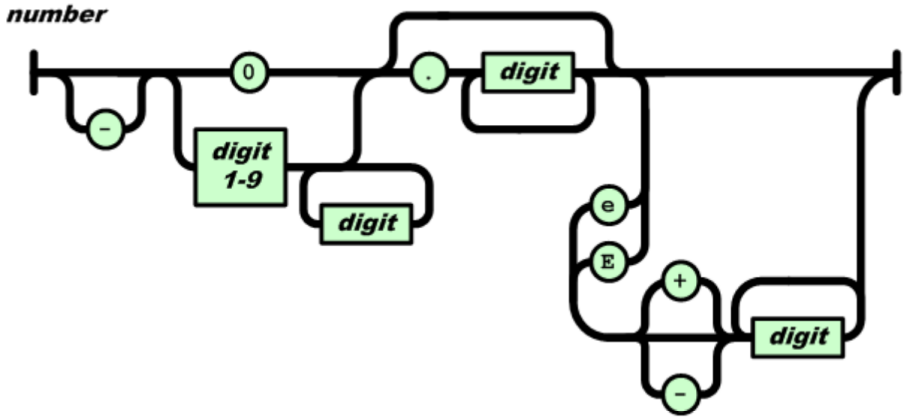

Figure 1.4: number

1.8 String

A string is a sequence of Unicode code points wrapped with quotation marks (U+0022). All code points may be placed within the quotation marks except for the code points that must be escaped: quotation mark (U+0022), reverse solidus (U+005C), and the control characters (U+0000 to U+001F).

There are two-character escape sequence representations of some characters:

- `\"` represents the quotation mark character (U+0022).
- `\\` represents the reverse solidus character (U+005C)..
- `\/` represents the solidus character (U+002F).
- `\b` represents the backspace character (U+0008).
- `\f` represents the form feed character (U+000C).
- `\n` represents the line feed character (U+000A).
- `\r` represents the carriage return character (U+000D).
- `\t` represents the character tabulation character (U+0009).

So, for example, a string containing only a single reverse solidus character may be represented as `"\\"`.

Any code point may be represented as a hexadecimal escape sequence. The meaning of such a hexadecimal number is determined by ISO/IEC 10646. If the code point 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 code point. Hexadecimal digits can be digits (U+0030 through U+0039) or the hexadecimal letters `A` through `F` in uppercase (U+0041 through U+0046) or lowercase (U+0061 through U+0066). 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 a code point that is not in the Basic Multilingual Plane, the character may be represented as a twelve-character sequence, encoding the UTF-16 surrogate pair corresponding to the code point. So for example, a string containing only the G clef character (U+1D11E) may be represented as `"\uD834\uDD1E"`. However, whether a processor of JSON texts interprets such a surrogate pair as a single code point or as an explicit surrogate pair is a semantic decision that is determined by the specific processor.

Note that the CirJSON grammar permits code points for which Unicode does not currently provide character assignments.

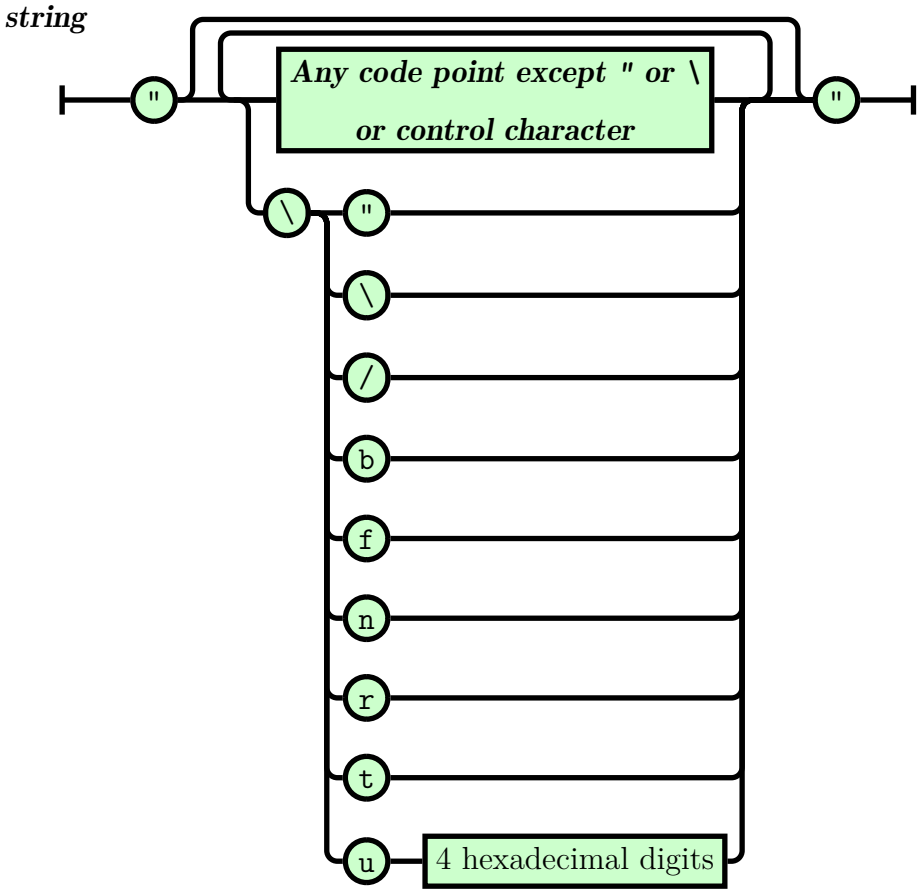

Figure 1.5: string

1.9 ID

An ID is a *string* that is not empty.

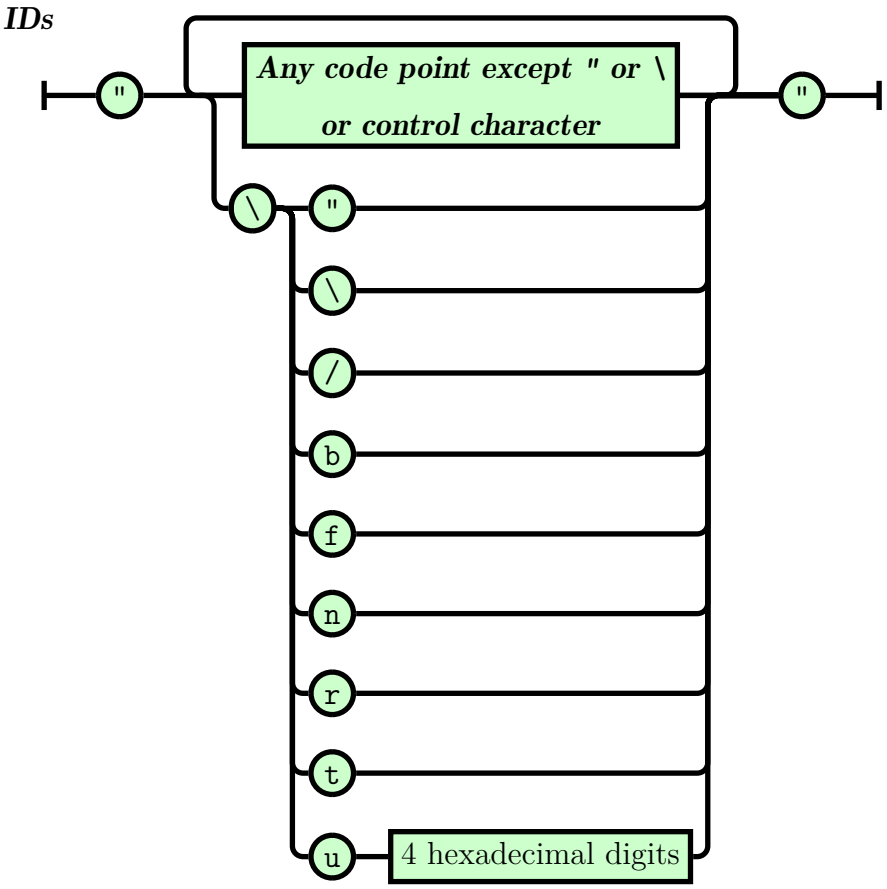

Figure 1.6: ID