JSON
Internet Draft
draft-jsonorg-json-00.txt
Intended status: Informational
Expires: June 10, 2006

D. Crockford
JSON.org
January, 2006

JSON

Status of this Memo

Copyright Notice

Abstract

JSON (JavaScript Object Notation) is a light-weight, text-based, language-independent, data interchange format. It was derived from ECMA 262 (The ECMAScript Programming Language Standard), Third Edition. JSON defines a small set of formatting rules for the portable representation of structured data.

Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",

"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119.

The syntax diagrams in this document are to be interpreted as described in RFC-2234.

1. Introduction

JSON, or JavaScript Object Notation, is a text format for the serialization of structured data. It is derived from the object literals of JavaScript, as defined in ECMA 262 (The ECMAScript Programming Language Standard), Third Edition (1999).

JSON can represent four primitive types (strings, numbers, booleans, and null) and two structured types (objects and arrays).

A string is a sequence of zero or more Unicode characters.

An object is an unordered collection of zero or more name/value pairs, where a name is a string, and a value is a string, number, boolean, null, object, or array.

An array is an ordered sequence of zero or more values.

The terms "object" and "array" come from the conventions of JavaScript.

2. JSON Grammar

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

These are the six structural characters:

```
<begin-object>    = %x7B      ; { left brace

<end-object>      = %x7D      ; } right brace

<begin-array>     = %x5B      ; [ left bracket

<end-array>       = %x5D      ; ] right brace

<name-separator>  = %x3A      ; : colon

<value-separator> = %x2C      ; , comma
```

2.1. Whitespace

The tokens may be separated by any combination of these whitespace characters:

```
   space U+0020     Space
```

```
   TAB   U+0009     Horizontal tab
   LF    U+000A     Line feed or New line
   CR    U+000D     Carriage return
```

Insignificant whitespace must not be placed within a multicharacter token (a literal name, number, or string). A space character in a string is significant.

2.2. Values

A JSON value can be a object, array, number, or string, or one of the literal names true, false, or null. The literal names must be in lower case. No other literal names are allowed.

```
   <value> = <string> / <number> / <object> / <array> /
             <true> / <false> / <null>

   <true>  = %x74.72.75.65    ; true

   <false> = %x66.61.6c.73.65 ; false

   <null>  = %x6e.75.6c.6c    ; null
```

2.3. Objects

An object structure is represented as a pair of curly braces surrounding zero or more name/value pairs (or members). 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.

```
   <object> = <begin-object> [ <member>
              *( <value-separator> <member> ) ] <end-object>

   <member> = <string> <name-separator> <value>
```

2.4. Arrays

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

```
   <array> = <begin-array> [ <value>
             *( <value-separator> <value> ) ] <end-array>
```

2.5. Numbers

The representation of numbers is similar to that used in programming languages. A number contains an integer component (which may be prefixed with an optional minus sign (U+002D)), which may be followed by a fraction part and/or an exponent part.

Octal and hex forms are not allowed. Leading zeros are not allowed as that could lead to confusion.

A fraction part is a decimal point (U+002E) followed by one or more digits.

An exponent part begins with the letter E in upper or lower case (U+0045 or U+0065), which may be followed by a plus (U+002B) or minus (U+002D). The E and optional sign are followed by one or more digits.

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

```
   <number> = [ "-" ] <int> [ <frac> ] [ <exp> ]

   <int> = "0" / ( <digit1-9> *<digit> )

   <frac> = "." 1*<digit>

   <exp> = ( "e" / "E" ) [ "-" / "+" ] 1*<digit>

   <digit> = "0" / "1" / "2" / "3" / "4" /
             "5" / "6" / "7" / "8" / "9"

   <digit1-9> = "1" / "2" / "3" / "4" /
                "5" / "6" / "7" / "8" / "9"
```

2.6. Strings

The representation of strings is similar to conventions used in the C family of programming languages. A string begins and ends with quotation marks (U+0022). All Unicode characters can be placed within the quotation marks except for the characters which must be escaped: quotation mark (U+0022), reverse virgule (U+005C), and the control characters (U+0000 through U+001F).

Any character may be escaped. 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 virgule followed by the lower case letter u (U+0075) followed by four hexadecimal digits which encode the character's code point. The hexadecimal letters a though f can be in upper or lower case. So, for example, a string containing only a single reverse virgule character may be represented as "\u005C".

Alternatively, there are two-character sequence escape representations of some popular characters. So, for example, a string containing only a single reverse virgule character may be represented more compactly as "\\".

Short	Long

```
  form    form
   \"    \u0022   quotation mark
   \\    \u005C   reverse virgule or backslash
   \/    \u002F   virgule or slash
   \b    \u0008   backspace
   \f    \u000C   form feed
   \n    \u000A   line feed or new line
   \r    \u000D   carriage return
   \t    \u0009   tab
```

To escape an extended character that is not in the Basic Multilingual Plane, then 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".

A space in a string is treated as a space character, not as insignificant whitespace.

```
   <string> = <quotation-mark> *<char> <quotation-mark>

   <quotation-mark> = %x22    ; "

   <escape> = %x5C            ; \

   <char> =
      <unescaped> /
      <escape> (
          %x22 /             ; " quotation mark
          %x5C /             ; \ reverse virgule
          %x2F /             ; / virgule
          %x62 /             ; b backspace
          %x66 /             ; f form feed
          %x6E /             ; n line feed
          %x72 /             ; r carriage return
          %x74 /             ; t tab
          %x75 4<hex-digit> ) ; uXXXX

   <hex-digit> = <digit> / "a" / "b" / "c" / "d" / "e" / "f" /
                           "A" / "B" / "C" / "D" / "E" / "F"

   <unescaped> = %x20-21 / %x23-5B / %x5D-10FFFF
```

3. Parsers

A JSON parser transforms a JSON text into another representation. A JSON parser MUST accept all texts that conform to the JSON grammar. A JSON parser MAY accept non-JSON forms or extensions.

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 of numbers.

An implementation may set limits on the length and character contents of strings.

4. Generators

A JSON generator produces JSON text. The resulting text MUST strictly conform to the JSON grammar.

5. IANA Considerations

The MIME media type for JSON text is text/json.

6. Security Considerations

Since JSON is a subset of JavaScript, the eval() function (which compiles and execute a text) can be used as a JSON parser. This should only done if the text is known to be safe. A regular expression can be used to prove that the text contains only JSON tokens. A text containing only JSON tokens is safe to eval because the JSON subset of JavaScript is safe.

Author's Address

Douglas Crockford
JSON.org
Contact Email: douglas@crockford.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

This Internet-Draft will expire on June 10, 2006.