The **`JSON`** object contains methods for parsing JavaScript Object Notation (JSON) and converting values to JSON. It can't be called or constructed, and aside from its two method properties, it has no interesting functionality of its own.

Description

JavaScript and JSON differences

JSON is a syntax for serializing objects, arrays, numbers, strings, booleans, and `null`. It is based upon JavaScript syntax but is distinct from it: some JavaScript is *not* JSON.

Objects and Arrays
Property names must be double-quoted strings; trailing commas are forbidden.

Numbers
Leading zeros are prohibited. A decimal point must be followed by at least one digit. `NaN` and `Infinity` are unsupported.

Any JSON text is a valid JavaScript expression...
...But only in JavaScript engines that have implemented the proposal to make all JSON text valid ECMA-262. In engines that haven't implemented the proposal, U+2028 LINE SEPARATOR and U+2029 PARAGRAPH SEPARATOR are allowed in string literals and property keys in JSON; but their use in these features in JavaScript string literals is a `SyntaxError`.

Consider this example where `JSON.parse()` parses the string as JSON and `eval` executes the string as JavaScript:

```
let code = '"\u2028\u2029"'
JSON.parse(code)  // evaluates to "\u2028\u2029" in all engines
eval(code)        // throws a SyntaxError in old engines
```

Other differences include allowing only double-quoted strings and having no provisions for `undefined` or comments. For those who wish to use a more human-friendly configuration format based on JSON, there is JSON5, used by the Babel compiler, and the more commonly used YAML.

Full JSON syntax

The full JSON syntax is as follows:

```
JSON = null
    or true or false
    or JSONNumber
    or JSONString
    or JSONObject
    or JSONArray

JSONNumber = - PositiveNumber
          or PositiveNumber
PositiveNumber = DecimalNumber
              or DecimalNumber . Digits
              or DecimalNumber . Digits ExponentPart
              or DecimalNumber ExponentPart
DecimalNumber = 0
             or OneToNine Digits
ExponentPart = e Exponent
            or E Exponent
Exponent = Digits
        or + Digits
        or - Digits
Digits = Digit
      or Digits Digit
Digit = 0 through 9
OneToNine = 1 through 9

JSONString = ""
          or " StringCharacters "
StringCharacters = StringCharacter
                or StringCharacters StringCharacter
StringCharacter = any character
                  except " or \ or U+0000 through U+001F
               or EscapeSequence
EscapeSequence = \" or \/ or \\ or \b or \f or \n or \r or \t
              or \u HexDigit HexDigit HexDigit HexDigit
HexDigit = 0 through 9
```

```
        or A through F
        or a through f

JSONObject = { }
          or { Members }
Members = JSONString : JSON
       or Members , JSONString : JSON

JSONArray = [ ]
         or [ ArrayElements ]
ArrayElements = JSON
             or ArrayElements , JSON
```

Insignificant whitespace may be present anywhere except within a *JSONNumber* (numbers must contain no whitespace) or *JSONString* (where it is interpreted as the corresponding character in the string, or would cause an error). The tab character (U+0009), carriage return (U+000D), line feed (U+000A), and space (U+0020) characters are the only valid whitespace characters.

Static methods

`JSON.parse(text[, reviver])`

Parse the string *`text`* as JSON, optionally transform the produced value and its properties, and return the value. Any violations of the JSON syntax, including those pertaining to the differences between JavaScript and JSON, cause a `SyntaxError` to be thrown. The *`reviver`* option allows for interpreting what the *`replacer`* has used to stand in for other datatypes.

`JSON.stringify(value[, replacer[, space]])`

Return a JSON string corresponding to the specified value, optionally including only certain properties or replacing property values in a user-defined manner. By default, all instances of `undefined` are replaced with `null`, and other unsupported native data types are censored. The *`replacer`* option allows for specifying other behavior.

Examples

Example JSON

```
{
  "browsers": {
    "firefox": {
      "name": "Firefox",
      "pref_url": "about:config",
      "releases": {
        "1": {
          "release_date": "2004-11-09",
          "status": "retired",
          "engine": "Gecko",
          "engine_version": "1.7"
        }
      }
    }
  }
}
```