The `jsonparse` package

An easy way to parse, store and access JSON data from files or strings in LaTeX documents

Jasper Habicht *

Version 0.7.1, released on 20 April 2024

---

1 Introduction

The `jsonparse` package provides a handy way to read in JSON data from files or strings in LaTeX documents, parse the data and store it in a user-defined token variable. The package allows accessing the stored data via a JavaScript-flavored syntax.

This package is still in a beta stage and not thoroughly tested. Bugs or improvements can be issued via GitHub at `https://github.com/jasperhabicht/jsonparse/issues`.

2 Loading the package

To install the package, copy the package file `jsonparse.sty` into the working directory or into the `texmf` directory. After the package has been installed, the `jsonparse` package is loaded by calling `\usepackage{jsonparse}` in the preamble of the document.

The package does not load any dependencies.

debug

The package can be loaded with the option `debug`. It will then output to the log file every instance of a string, a boolean (true or false) value, a null value, a number as well as the start and end of every object and the start and end of every array that is found while parsing the JSON string or JSON file.

3 Escaping and special treatment of the input

escape={‹choice›}

In general, characters in the JSON source that are special to TeX are not handled in a special way and will be treated by TeX the same way as if the user had input them in the document. However, certain escaping procedures are available to conform with the way JSON treats certain escape sequences.

The key `escape` takes one of the values `false`, `basic` or `full`. If no value is given, `basic` is assumed and `basic` is also the default setting if the key is not set at all. See more on setting keys below in section 4.1. The three values select different escape modes that are designed to have as little influence as possible to how TeX normally acts. While it would surely be possible to parse and rescan the input by changing category codes and rescanning the output per default, such practice might result un unintended output in special cases. Therefore, the current approach was chosen

* E-mail: mail@jasperhabicht.de

where changes to category codes are only introduced with the most most extensive escaping mode and rescanning of token sequences can be controlled by the user.

Setting the key to `false` disables the treatment of the input as described in the following of this section. The source code will be read and parsed as is. If the source does not contain any escape sequences, empty lines or TeX macro, this choice should be used.

If the key `escape` is set to `basic`, the package allows for empty lines in JSON strings. During parsing, every instance of the TeX macro `\par` is replaced by a space.

JSON strings cannot contain the two characters `"` and `\`. These two characters need to be escaped with a preceding backslash (`\`). If the key `escape` is set to `basic`, this package therefore redefines locally the TeX control symbols `\"`, `\/` and `\\`. During parsing, `\"` expands to `\exp_not:N \"` (i. e. it is prevented to expand during parsing) and only when typeset, `\"` is expanded to `"`, which ensures that strings are parsed properly. Similarly, the control symbol `\/` expands to `\exp_not:N \/` and finally to `/` while `\\` expands to `\exp_not:N \\` and finally to `\c_backslash_str` (i. e. a backslash with category code 12). Due to this procedure, the TeX macros `\"` and `\\` must be escaped twice in the JSON source, so that they become `\\\"` and `\\\\` respectively.

With the key `escape` set to `basic`, other escape sequences defined by JSON, such as `\b`, `\f`, `\n`, `\r`, `\t` or `\u` (the latter followed by a hex value) are not escaped.

If the key `escape` is set to `full`, apart from the above, the JSON escape sequences `\b`, `\f`, `\n`, `\r`, `\t` or `\u` (followed by a hex value) are parsed if the JSON source is read in as file using `\JSONParseFromFile` (in other words, full escaping functionality is not supported for the command `\JSONParse`). During parsing, these escape sequences are not expanded and only when being typeset expand to their relevant replacement. The escape sequence `\u` followed by a hex value consisting of four digits eventually expands to `\char"` followed by the relevant four hex digits. The escape sequences `\b` (backspace), `\f` (formfeed), `\n` (linefeed), `\r` (carriage return) and `\t` (horizontal tab) expand to token variables of which the contents can be set using the relevant `replacement` key. See more on setting keys below in section 4.1.

Note that if the key `escape` is set to `full`, the category code of `b`, `f`, `n`, `r`, `t` and `u` will be set to 12 (other). Changing the category codes is necessary to be able to define single-letter control sequences in TeX. The category codes of these characters are changed back to 10 (letter) when stored in the property list that contains the parsed JSON string.

```
replacement/backspace={⟨string⟩}
replacement/formfeed={⟨string⟩}
replacement/linefeed={⟨string⟩}
replacement/carriage return={⟨string⟩}
replacement/horizontal tab={⟨string⟩}
```

These keys can be used to set the replacement text for the JSON escape sequences `\b` (backspace), `\f` (formfeed), `\n` (linefeed), `\r` (carriage return) and `\t` (horizontal tab). The default replacement string is a space. Only strings can be used as replacement.

4 Main user commands

```
\JSONParse{⟨token variable⟩}{⟨JSON string⟩}
```

The command `\JSONParse` is used to parse a JSON string and store the parsed result in a token variable (a property list). The first argument takes the name of the token variable that is created by the command. The second argument takes the JSON string to be parsed.

For example, using `\JSONParse{\myJSONdata}{ { "key" : "value" } }`, the relevant JSON string will be parsed and the result stored in the token variable `\myJSONdata` as property list. In this case, the property list only consists of one entry with the key `key` and the value `value`. The

command `\JSONParseValue{\myJSONdata}{key}`, for example, can then be used to extract the relevant value from this property list (see the description below).

\JSONParseFromFile{‹token variable›}{‹JSON file›}

The command `\JSONParseFromFile` is used to parse a JSON file and store the parsed result in a token variable (a property list). It works the same way as `\JSONParse`, but instead of a JSON string, it takes as second argument the path to the JSON file relative to the working directory.

This command will temporarily change the category code of `b`, `f`, `n`, `r`, `t` and `u` to 12 (other) if full escaping is activated. See more on escaping above in section 3.

\JSONParseKeys{‹token variable›}{‹token variable›}

The command `\JSONParseKeys` is used to store all top-level keys of a parsed JSON object as array into a token variable. The command takes as first argument the token variable that holds the parsed JSON data. The second argument takes the token variable that is assigned a JSON array containing the top-level keys of the object represented by the token variable in the first argument. The token variable to store the keys as array is created if it does not exist.

\JSONParseValue{‹token variable›}{‹key›}
\JSONParseValue*{‹token variable›}{‹key›}
\JSONParseExpandableValue{‹token variable›}{‹key›}

The command `\JSONParseValue` is used to select values from the token variable (property list) that has been created using the commands `\JSONParse` or `\JSONParseFromFile`. The first argument takes the token variable that holds the parsed JSON data. The second argument takes the key to select the relevant entry from the parsed JSON data using JavaScript syntax.

If the JSON string `{ "key" : "value" }` is parsed into the token variable `\myJSONdata`, using `\JSONParseValue{\myJSONdata}{key}` would extract the value associated with the key `key`, which in this case is `value`, and typeset it to the document.

Nested objects and arrays are assigned keys that adhere to JavaScript syntax. For example, if the JSON string `{ "outer_key" : { "inner_key" : "value" } }` is parsed into the token variable `\myJSONdata`, to select the value associated with the key `inner_key`, the command `\JSONParseValue{\myJSONdata}{outer_key.inner_key}` can be used. To give an example for an array, the command `\JSONParseValue{\myJSONdata}{key[0]}` selects the first value of the array associated with the key `key` in the JSON string `{ "key" : [ "one" , "two" ] }`.

The starred variant, `\JSONParseValue*`, rescans the token list before it is typeset (which means that all category codes that may have been changed before are set to the default values), making it possible to place TeX commands in the JSON file. The starred variants of this and similar commands should not be placed in a `\JSONParse` command. In order to adhere to proper JSON syntax, backslashes need to be escaped in the JSON source with another backslash.

When a key is associated with an object or array, the whole object or array is output as JSON string. The special key `.` (or the string defined using the key `child sep`) returns the whole JSON object as string.

If the output of whole objects or arrays is meant to be parsed again using `\JSONParse`, the expandable command `\JSONParseExpandableValue` is to be used.

\JSONParseArrayValues{‹token variable›}{‹key›}[‹subkey›]{‹string›}
\JSONParseArrayValues*{‹token variable›}{‹key›}[‹subkey›]{‹string›}

The command `\JSONParseArrayValues` is used to select all values from an array from a parsed JSON string or JSON file. The first argument takes the token variable that holds the parsed JSON

data. The second argument takes the key to select the relevant entry from the parsed JSON data using JavaScript syntax. The third argument is optional and can be used to pass a subkey, i. e. a key that is used to select a value for every item. The last argument takes a string that is inserted between all values when they are typeset.

For example, let us assume the following JSON data structure is parsed into the token variable `\myJSONdata`:

```
{
  "array" : [
    {
      "key_a" : "one" ,
      "key_b" : "two"
    } ,
    {
      "key_a" : "three" ,
      "key_b" : "four"
    }
  ]
}
```

Then, when using `\JSONParseArrayValues{\myJSONdata}{array}[key_a]{, }`, 'one, three' is typeset to the document.

The starred variant, `\JSONParseArrayValues*`, rescans the token lists before they are typeset.

\JSONParseArrayValuesMap{‹token variable›}{‹key›}[‹subkey›]{‹command name›}
\JSONParseArrayValuesMap*{‹token variable›}{‹key›}[‹subkey›]{‹command name›}

The command `\JSONParseArrayValuesMap` takes the same first three arguments as the command `\JSONParseArrayValues` and works in a similar way. However, instead of a string that is added between the array items, it takes a command name as fourth argument. This command can be defined beforehand and will be called for every array item. Inside its definition, the commands `\JSONParseArrayIndex`, `\JSONParseArrayKey` and `\JSONParseArrayValue` can be used which are updated for each item and output the index, the key and the value of the current item respectively.

For example, let us assume the same JSON data structure as defined above parsed into the token variable `\myJSONdata`. Then, the following can be done:

- *one*
- *three*

```
\newcommand{\myJSONitem}{
  \item \emph{\JSONParseArrayValue}
}

\begin{itemize}
  \JSONParseArrayValuesMap{\myJSONdata}
    {array}[key_a]{myJSONitem}
\end{itemize}
```

The starred variant, `\JSONParseArrayValuesMap*`, rescans the token lists before they are typeset.

\JSONParseArrayCount{‹token variable›}{‹key›}

The command `\JSONParseArrayCount` takes as first argument a token variable holding a parsed

JSON string or JSON file and as second argument a key. It returns an integer representing the number of items contained in the selected array.

4.1 Changing separators, output and other settings

The package provides a set of keys can be set to change the separators used to select the relevant value in the JSON structure, the output that is generated from the JSON data as well as other things.

```
\JSONParseSet{‹key-value list›}
```

The commands `\JSONParseSet` can be used to specify settings via key-value pairs (separated by commas). Keys that are presented here as a subkey (i. e. preceded by another key and a slash) can also be set using the syntax `key={subkey}` and multiple subkeys belonging to one key can be combined using a comma as separator. The following keys are available:

```
separator/child={‹string›}
```

With the key `child sep`, the separator for child objects that is used in the syntax to select a specific value in the JSON data structure can be changed. Per default, the child separator is a dot (`.`). Changing the separator can be useful if keys in the JSON structure already use these characters.

```
separator/array left={‹string›}
separator/array right={‹string›}
```

With the keys `array sep left` and `array sep right`, the separators for arrays that are used in the syntax to select a specific value in the JSON data structure can be changed. Per default, the separators are square brackets (`[` and `]`). Changing the separators can be useful if keys in the JSON structure already use these characters.

```
replacement/true={‹string›}
replacement/false={‹string›}
replacement/null={‹string›}
```

With the keys `true`, `false` and `null`, the string that is typeset for true, false and null values can be changed. The default strings that are typeset are `true`, `false` and `null` respectively. Only strings can be used as replacement.

```
array index zero-based
array index zero-based={‹boolean›}
```

If set (or explicitly set to `true`), the key `array index zero-based` sets the numbering of the index of array items to zero-based. If set to false, the indexing starts with one instead. Per default, the package uses zero-based indexing to resemble JavaScript notation.

4.2 L3 commands

```
\jsonparse_parse:n {‹JSON string›}
```

The command `\jsonparse_parse:n` takes as argument a JSON string and populates the token variable (property list) `\g_jsonparse_entries_prop` with key-value pairs representing all elements of the JSON data structure represented by this string. This command does not escape the input in any way.

\jsonparse_parse_to_prop:Nn ‹token variable› {‹JSON string›}

The command `\jsonparse_parse_to_prop:Nn` creates the token variable given as the first arguments as property list and, after having called `\jsonparse_parse:n` using the second argument, sets this newly created property list equal to `\g_jsonparse_entries_prop`. If escaping is activated, this command will pre-process the input according to the selected escaping mode before forwarding it to `\jsonparse_parse:n`. See more on escaping above in section 3.

\jsonparse_filter:Nn ‹token variable› {‹key›}

The command `\jsonparse_parse_to_prop:Nn` processes the token variable given as the first arguments as property list and filters it according to the key given as second argument. Filtering means that for every entry in the property list, the key of this entry is compared against the key given to the command. If the key in the property list starts with the given key, the matching part is removed from the key in the property list. If the keys do not match, the entry is completely removed from the property list.

5 Changes

v0.3.0 (2024/04/08)
First public beta release.

v0.5.0 (2024/04/09)
Changed from string token variables to token lists to support Unicode.

v0.5.5 (2024/04/09)
Bug fixes, introduction and enhancement of user functions.

v0.5.6 (2024/04/11)
Bug fixes, escaping of special chars added.

v0.5.7 (2024/04/14)
Bug fixes, key-value option setting added.

v0.6.0 (2024/04/15)
Bug fixes, renaming of several commands.

v0.7.0 (2024/04/18)
Renaming and rearranging of keys, escaping of special JSON escape sequences added.

v0.7.1 (2024/04/20)
Access to first-level keys of object added.