The `jsonparse` package

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

Jasper Habicht [*]

Version 0.9.1, released on 21 September 2024

1 Introduction

Hello guys, I am Jason, the JSON parsing horse. JSON data is my favorite thing to parse! But I found that converting JSON to TeX can be a bit tricky. Therefore, I created this package which I am happy to introduce to you.

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.

This key can be set either as package option or using `\JSONParseSet`. It can also be set locally as option to the commands `\JSONParse` and `\JSONParseFromFile`.

---

3 Escaping and special treatment of the input

In general, the package reads the JSON source as string, which means that all characters have category code 12 (other), except for spaces which have category code 10 (space). The `\endlinechar` value is set to -1. Furthermore, if PDFLaTeX is used, the upper-half of the 8-bit range is set to "active". JSON, however, defines a small set of escape sequences and in order to be able to process these, the category code of the backslash is set to 0 (escape).

JSON strings cannot contain the two characters `"` and `\`. These two characters need to be escaped with a preceding backslash (`\`). This package therefore redefines locally the TeX control symbols `\"`, `\/`, `\\`, `\b`, `\f`, `\n`, `\r`, `\t` and `\u`. These control symbols are prevented from expanding during parsing. For example, `\"` is first defined as `\exp_not:N \"` and only when typeset, `\"` is expanded to `"`, which ensures that strings are parsed properly.

Similarly, the control symbol `\/` expands eventually to `/` and `\\` to `\c_backslash_str` (i. e. a backslash with category code 12). 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 JSON escape sequences `\b`, `\f`, `\n`, `\r` and `\t` eventually expand to token variables of which the contents can be set using the relevant `replacement` key. See more on setting options below in section 4.2.

It is possible to insert TeX macros to the JSON source that will eventually be parsed when typesetting. Backslashes of TeX macros need to be escaped by another backslash. The TeX macros `\"` and `\\` must be escaped twice in the JSON source so that they become `\\\"` and `\\\\` respectively.

`\x`{‹token variable name›}{‹key›}

Using the control sequence `\x`, it is possible to nest JSON strings into each other. The control sequence takes two arguments delimited by curly braces. The first argument represents the name of the token variable that holds the parsed JSON data where the inserted JSON string should be taken from. The second argument sets the key that should be selected. The following example shows a simple use case:

```
                          \JSONParse{\myJSONdataA}{
                              { "a" : { "b" : "c" } }
                          }

c                         \JSONParse{\myJSONdataB}{
                              { "d" : \x{myJSONdataA}{a} }
                          }

                          \JSONParseValue{\myJSONdataB}{d.b}
```

Note that the control sequence `\x` is replaced by the value exactly. Therefore, if the value happens to be a string, the control sequence `\x` should be placed between quotation marks (`"`) in order for the resulting string to be valid JSON.

---

escape={all}
escape={none}
escape={number sign}
escape={dollar sign}
escape={percent sign}
escape={ampersand}
escape={circumflex accent}
escape={low line}
escape={tilde}

The key `escape` can be used to convert characters that don't require escaping in JSON but in TeX into the relevant TeX escape sequences. Apart from the backslash and curly braces that need to be escaped anyways, these are the number sign, the dollar sign, the percent sign, the ampersand, the circumflex accent, the low line and the tilde. The characters can be selected individually separated by a comma (for example `escape={dollar sign, circumflex accent, low line}`. With `escape={all}`, all escaping sequences are selected, with `escape={none}`, none is selected.

The naming of the relevant characters follows their Unicode names. However, `hash` exists as alias for `number sign`, `dollar` as alias for `dollar sign`, `percent` for `percent sign`, `circumflex` for `circumflex accent` and `underscore` for `low line`.

This key can be set using `\JSONParseSet`. It can also be set locally as option to the commands `\JSONParseValue`, `\JSONParseArrayValues` and `\JSONParseArrayValuesMap`.

---

rescan
rescan={‹boolean›}

The key `rescan` can be used to activate and deactivate rescanning of the output. This key is active per default. Rescanning converts all tokens to their default category codes and TeX control sequences are expanded before typesetting. Further, during the rescanning process, JSON escape sequences are replaced and characters that don't require escaping in JSON but in TeX are replaced by the relevant TeX escape sequences.

This key can be set using `\JSONParseSet`. It can also be set locally as option to the commands `\JSONParseValue`, `\JSONParseArrayValues` and `\JSONParseArrayValuesMap`.

4   Main user commands

\JSONParse[‹options›]{‹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 second argument takes the name of the token variable that is created by the command. The third 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).

The first optional argument can be used to pass options to the command that are then applied locally.

\JSONParseFromFile[‹options›]{‹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 third argument the path to the JSON file relative to the working directory.

\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[‹options›]{‹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 first optional argument can be used to pass options to the command, such as `escape` or `rescan`, that are then applied locally. When the option `rescan` is used, the token list is rescanned before it is typeset (which means that all category codes that may have been changed before are set to the default values). This is the default behaviour. If rescanning is not desired, pass the option `rescan=false` to the command.

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.

Whole objects or arrays can also be output as JSON string for further use in other macros using the expandable command `\JSONParseExpandableValue`.

\JSONParseArrayValues[‹options›]{‹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 first optional argument can be used to pass options to the command, such as `escape` or `rescan`, that are then applied locally.

\JSONParseArrayValuesMap[‹`options`›]{‹`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 first optional argument can be used to pass options to the command, such as `escape` or `rescan`, that are then applied locally.

\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 Externalising parsed JSON data

Parsing large JSON files can take quite a while. In order to speed up follow-up compilation runs, this package provides a way to store parsed JSON data for future use. Once a file for externalization has been created, the package will try to load the data from this file instead of parsing the JSON data again.

externalize
externalize={‹`boolean`›}

With the key `externalize` set (or set to true), a file will be created in the working directory that stores the externalization of the parsed JSON data. The file name gets the extension `.jsonparse`. The file name is created automatically and consists of the name of the current file followed by an underscore and the name of the token variable where the JSON data is stored into. If a file with the same name and file extension already exists, an error will be issued.

This key can be set using `\JSONParseSet`. It can also be set locally as option to the commands `\JSONParse` and `\JSONParseFromFile`.

```
externalize prefix={‹string›}
```

With the key `externalize prefix`, a prefix can be defined that is added to the file name. Per default this is an empty string.

This key can be set using `\JSONParseSet`. It can also be set locally as option to the commands `\JSONParse` and `\JSONParseFromFile`.

```
externalize file name={‹token list›}
```

The key `externalize file name` sets the schema for the file name. The default schema is as follows:

```
\l_jsonparse_externalize_prefix_str \c_sys_jobname_str
    \c_underscore_str \l_jsonparse_current_prop_str
```

The token variable `\l_jsonparse_externalize_prefix_str` contains the prefix that is set using the key `externalize prefix`. `\c_sys_jobname_str` holds the name of the current file (the current job name), `\c_underscore_str` is an underscore and the token variable `\l_jsonparse_current_prop_str` contains the name of the property list where the relevant JSON data is stored into.

This key can be set using `\JSONParseSet`. It can also be set locally as option to the commands `\JSONParse` and `\JSONParseFromFile`.

```
\JSONParsePut{‹token variable›}{‹key›}[‹JSON string›]
```

The command `\JSONParsePut` is used by the externalization procedure to re-read already parsed JSON data to the main file. It just adds a key-value pair to the property list (where the value part is read as string). Hence, it can also be used to append more entries to an already existing property list containing parsed JSON data.

4.2   Changing separators, output and other options

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{‹options›}
```

The commands `\JSONParseSet` can be used to specify options 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. Several user commands allow to pass keys directly which are then applied locally. The following keys are available:

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

With the key `separator/child`, 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.

With the keys `separator/array left` and `separator/array 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. Changing these separators to curly braces (`{}`) is not supported due to their grouping function in TeX.

These keys can be set using `\JSONParseSet`. They can also be set locally as option to the commands `\JSONParse` and `\JSONParseFromFile`.

---

zero-based
zero-based={‹boolean›}

If set (or explicitly set to `true`), the key `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.

This key can be set using `\JSONParseSet`. It can also be set locally as option to the commands `\JSONParse` and `\JSONParseFromFile`.

---

replace/**true**={‹string›}
replace/**false**={‹string›}
replace/**null**={‹string›}

With the keys `replace/true`, `replace/false` and `replace/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. These replacements take place already during parsing.

These keys can be set using `\JSONParseSet`. They can also be set locally as option to the commands `\JSONParse` and `\JSONParseFromFile`.

---

replace/**backspace**={‹string›}
replace/**formfeed**={‹string›}
replace/**linefeed**={‹string›}
replace/**carriage return**={‹string›}
replace/**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. These replacements take place only during typesetting.

These keys can be set using `\JSONParseSet`. They can also be set locally as option to the commands `\JSONParseValue`, `\JSONParseArrayValues` and `\JSONParseArrayValuesMap`.

---

check num
check num={‹boolean›}

If set to `false`, the key `check num` omits an internal check of numerical expressions against the JSON specification for numbers. Turning off this feature can increase the parsing speed.

This key can be set using `\JSONParseSet`. It can also be set locally as option to the commands `\JSONParse` and `\JSONParseFromFile`.

4.3   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.

`\jsonparse_if_num:nTF` {‹string›} {‹true code›} {‹false code›}
`\jsonparse_if_num:nT` {‹string›} {‹true code›}
`\jsonparse_if_num:nF` {‹string›} {‹false code›}
`\jsonparse_if_num_p:n` {‹string›}

The command `\jsonparse_if_num:nTF` checks whether a string is a valid JSON number according the relevant specification. It executes the true code if the string is a valid JSON number and the false code if not. The variants `\jsonparse_if_num:nT` and `\jsonparse_if_num:nF` work accordingly. The command `\jsonparse_if_num_p:n` returns a boolean true or false.

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 top-level keys of object added.

v0.8.0 (2024/04/24)
    Internal rewrite, escaping procedures changed.

v0.8.2 (2024/04/26)
    Bug fixes, externalizing parsed data.

v0.8.3 (2024/04/28)
    Escaping of characters with special meaning in TeX.

v0.8.5 (2024/05/05)
    Enhanced key management.

v0.8.6 (2024/05/09)
    Bug fix in nesting function.

v0.8.7 (2024/08/08)
    Corrections in documentation, error messages.

v0.9.0 (2024/08/27)
    Adaption to updated verbatim tokenization.

v0.9.1 (2024/09/21)
    Added functions to test for valid JSON numbers.