JSON Data Plugin

Совместим с «Докувики»

- 2024-02-06 "Kaos" **да**
- 2023-04-04 "Jack Jackrum" **да**
- 2022-07-31 "Igor" **да**
- 2020-07-29 "Hogfather" **да**

Build JSON database inside DokuWiki page and use the data in the page

Последнее обновление:
: 2024-03-11

Предоставляет
: Syntax, Helper, Action, Remote

Репозиторий
: исходный код

Похож на data, jsoneditor, jsongendoc, jsontable, strata, struct

Теги: data, database, json, listing, tables, template, xmlrpc

- Janez Paternoster

Installation

- Search and install the plugin using the Extension Manager. Refer to Plugins on how to install plugins manually.

You may want to install other related plugins:

- JSON editor plugin
- JSON table plugin
- JSON generate document plugin

Description

With JSON Data Plugin you can build JSON database inside DokuWiki page. JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is a Document-oriented database, so no SQL is used.

To add JSON database into the wiki page write:

```
<json path=person>{
  "firstName": "James",
  "lastName": "Smith"
}</json>
```

With multiple `<json>` elements inside the page database is generated internally. There may also be external data sources.

Data from internal database can be used on the page, for example:

```
Person described here is %$person.firstName% %$person.lastName%.
```

will produce: «Person described here is James Smith.»

JSON data can be used in the following ways:

- Extract specific JSON **basic** element.
- Extract specific JSON object element as a **list**.
- Extract specific JSON array element as a **table**.
- Print specific part of data as JSON code.
- Show or hide sections of DokuWiki page, based on value of specific JSON element.
- As a datasource for a javascript widget.
- JSON data can also be edited and safely stored back into the dokuwiki page via ajax call.

Demo

There is a DokuWiki JSON Demo Server with JSON database integrated into DokuWiki. Also source code of this plugin contains demo for JSON Data Definition and Usage. You can copy the contents of the demo files into your DokuWiki and experiment with them.

Support

For issues or just questions use Issues on Gitlab. Please don't email directly.

JSON Data Definition Syntax

Each page must define it's JSON data. Data are inline or are loaded from external (text) files on each page refresh. Only those data are loaded, which are defined inside a page. Data are defined with one or more '<json>' elements:

`<json` *attributes*`>`*inline_json*`</json>`

When the DokuWiki page is served, there is internally a variable, which is an empty object on the beginning. For example `json_database = {}`. After the above first `<json>` element is rendered, variable is something like this: `json_database = {"person": {"firstname": "James", "lastName": "Smith"}}`. After each subsequent `<json>` element data is added to that database. `path` attribute specifies, where in the `json_database` data are appended or replaced.

Inline JSON

Valid JSON data can be located between <json ...> and </json> tags. If both, inline data and data referenced by 'src' attribute are specified, then inline data have precedence. Inline data will replace overlapped data referenced by 'src' attribute.

Attribute 'id'

`id` attribute must be specified, when we want to access this json element from somewhere else or if we want to write inline JSON data via ajax call. It must be unique on one page.

Attribute 'path'

`path` attribute specifies, where in the 'json_database' data will be added. For example `path=person1.address` will write data into `json_database.person1.address`. `.` (dot) is used as a delimiter between tokens. If tokens contains spaces, then they must be inside quotes.

If token is number, then it points to n-th array member. `0` points to first array member and so on. Special token `_FIRST_` or `_LAST_` points to the first or to the last member.

If there are already data on the specified path, then new data will recursively replace overlapped original data. (For detailed rules see php function array_replace_recursive.) This is true, if the following two modifiers are not used.

Modifier **-**: if `-` (minus) is set in the beginning of the path attribute, then original data on the path will be erased before new data will be written. For example `path=-person1.address`.

Modifier **[]**: open square bracket immediately followed by closing square bracket at the end of the path attribute means, that data will be appended (pushed) to the specified path on the 'json_database'. For example `path=persons[]` will put the data in the `json_database.persons`

array in the next free index.

If path attribute is not specified, data is combined with the 'json_database' directly.

Attribute 'src'

`src` attribute contains a DokuWiki link to the external data. External data is a text file. It may be document from the same dokuwiki or it can be located anywhere on the net.

```
src=path:to:remote_document
src=https://www.example.com/file.json
src=path:to:remote_document#specific_element
src=persons:person*
src='{"JSON": "data", "can clone": %$pre-defined.data%}'
```

External data may be a pure JSON file, it begins with `[` or `{`.

External data may be a text file with <json> elements inside - remote DokuWiki page. Remote DokuWiki page then first loads data according to the rules inside its <json> elements, builds its own database and then passes the generated database into our document. Please note, that <json> elements inside remote dokuwiki page may also contain 'src' attribute. In this case data from that 'src' are also loaded. Data are loaded recursively. How deep recursion goes is controlled in configuration setting 'src_recursive'.

External data may be a specific <json> element from remote dokuwiki page. In that case <json> element must have an 'id' attribute. It is referenced as in third example above. If target <json> element have a 'src' attribute, then data is loaded recursively.

Wildcards may be may be used in 'src' attribute. See fourth example above. In that case multiple files are read and integrated into our database. See glob() function.

Fifth example above shows, that 'src' attribute can not only be used for the file path, it can also contain JSON data. This way some already defined data from page can be «cloned» and reused. Similar as in Data snippets inside inline JSON. JSON data are automatically recognized, if 'src' attribute contains: `[`, `{` or `%` on the beginning and `]`, `]` or `%` on the end.

Attribute 'src_ext'

This attribute is similar to 'src'. But the link to remote data is defined externally. Value of 'src_ext' contains a name, which must begin with `json_`. For example, 'our_document' contains:

```
<json src_ext=json_my_data></json>
```

In some other page we have a link to 'our_document' with one (or more) extra argument, like this:

```
[[somewhere:our_document?json_my_data=path:to:remote_document]]
```

'our_document' will be loaded with data as specified in Query string. Rules for the file link from Query string are the same as for the 'src' attribute. We can put multiple arguments separated by & in

Query string. If we want to specify specific <json> element, then we can't use #, because it is reserved. We can use %23 instead.

If besides 'src_ext' also 'src' attribute is specified, then 'src_ext' has precedence. If link for 'src_ext' is not defined in query string, then link from 'src' is used.

Attribute 'src_path'

If attribute `src_path` is specified, then only part of database referenced by 'src' attribute is used. Path on 'src' specified by 'src_path' must exist. Rules for 'src_path' are the same as for 'path', except modifiers are not used.

Attribute 'src_path_ext'

This attribute is similar to 'src_path', but the value is defined externally in query string. It works the same way as attribute 'src_ext'.

Attribute 'display'

This attribute controls, which part of <json> element is rendered on html page. Multiple data can be displayed with jQuery UI Tabs widget. Attribute 'display' is a comma separated list of tokens. Default value for this attribute is specified in configuration setting 'json_display'. Tokens:

- 'original' - display data already in database combined with data from 'src' attribute. But not yet combined with 'inline' data.
- 'inline' - display inline_data from <json>inline_data</json> element in textarea.
- 'combined' - display original data combined with 'inline' data.
- 'log' - display log of data sources.
- 'error' - display log of data sources only, of there are errors.

There are also tokens 'orig-hidden', 'inl-hidden' and 'comb-hidden', which are similar to above three tokens. Each renders a hidden html 'div' element, which can be used for passing data to javascript widgets.

By default, if no 'display' attribute, only tabs menu will be shown with two tabs: 'inline' and 'log'. Both tabs will be collapsed. This can be changed in configuration settings.

If we want to add some tokens to default 'display' options, we can use comma in front of our token(s) in 'display' attribute.

If we want only to embed data into document and hide the element, we may set display='error' to show element only if there are errors or set display='' to unconditionally hide the element.

If we want to show contents of specific tab, then asterisk (*) should be added after the corresponding token. If 'display' contains only one token, followed by asterisk (*), then tabs menu will be hidden. For example 'display=combined*' will only show JSON data without the tabs menu.

If 'inline' token is specified, then inline data are shown in textarea. That data can be edited and saved

to the document. When data is first changed, then 'Save' button is shown in tab area. There are some rules, which may prevent the saving. First, <json> element must have unique 'id' specified. Second, if someone else changed the data since last page reload, then data can not be written. It is necessary to reload the page first (in another browser tab?) and re-enter our data. If data is successfully saved, then 'Save' button disappears from tab area.

Attribute 'archive'

This attribute enables us to make archive of the JSON database loaded by 'src' or 'src_ext' attribute. Archive is stored into DokuWiki page itself as JSON string. This action is triggered, when user presses a button.

We can specify 'archive' attribute for each <jsonxxx> element on the page. We can specify 'archive=make' or 'archive=disable'. If at least one <jsonxxx> element has 'archive' attribute equal to 'make' or 'disable', then button 'Archive JSON database' appears on the top of DokuWiki page. When user presses that button, following procedure is triggered:

1. Verify for errors: user permission, file unmodified, lock, etc.
2. Search DokuWiki page and find all `<jsonxxx archive=make ...>` elements. Replace them with `<jsonxxx archive='/JSON_encoded_string/' ...>`.
3. Find all `<jsonxxx archive=disable src=... src_ext=...>` elements and replace them with `<jsonxxx archive_disabled=disable src_disabled=... src_ext_disabled=...>`. This disables 'src' and 'src_ext' attributes.
4. Save the DokuWiki page.

When DokuWiki page is archived, then 'archive' attribute of some or all <jsonxxx> elements contains JSON database. Data is then read from 'archive' attribute. 'src' and 'scr_ext' attributes are then ignored. Inline JSON data remain unchanged.

JSON Data Usage Syntax

Basic pattern for render JSON data in dokuwiki is: `%$ ... %`. JSON data can be displayed in multiple different ways, for example: simple string, as link, list of properties, table, json code, etc. Detailed description of pattern:

```
%$path [(table_row_filter)] {header} #format# (filter)%
```

Each of: `path`, `table_row_filter`, `header`, `format`, `filter` is optional. Order of the brackets is important. If square brackets are used, then table will be rendered. Else if curly brackets are used, then list will be rendered. Otherwise single variable will be rendered. If special character must be used inside the pattern, then use HTML code. Use `&#37;` instead of %, for example.

path

'path' specifies part of the 'json_database', which will be rendered. `.` (dot) is used as a delimiter between tokens. Path may contain spaces, no quotes should be used. Following characters are not allowed inside path: '[]{}#()'. There are two special tokens, which may be used as part of the path:

FIRST selects the first element inside the array and LAST selects the last element inside he array.

table_row_filter

Square brackets will render table. *table_row_filter* inside brackets is optional. It will display each row of the table, if filter is matched. Rules are the same as for filter below.

header

header is a comma separated list of `key:value` pairs, where `key` is a header description and `value` is a path to variable. It is used to render a header in a table or in a list of properties.

In *header* it is also possible to define tooltips, which can be displayed on mouse hover on specific table row or table cell. Tooltip is defined, if `key` is prepended with special string `_tooltip_`.

Example header for two column table and with tooltip on second column: `"Header 1":data.1, "Header 2":data.2, "_tooltip_Header 2":data.2.tooltip`. This example can also be used for a list of two properties with tooltip on second property.

If we want to use one tooltip for whole table row, then we use just `_tooltip_` string for the `key`.

format

Format may be applied on any variable, list item or table column. It can render value of the variable to format other than text. Supported formats are:

- `code` - render variable as json code.
- `headern` - render header, where n is the number from 1-5 for header level.
- `link` - render as internal or external or windows share link. If variable is external link, it must have protocol specified, for example 'http://...'. Variable may also have title specified, for example 'some:link|Some Title'.
- `email` - render as email address. Variable may have title specified.
- `media?L200x300` - render as internal or external media file. ? and parameters are optional. First letter for parameter must be 'l' for left, 'c' for center or 'r' for right position of the media file. Other part of parameter is width x height. Parameter may also be just 'linkonly'. Variable may have title specified.
- `rss n nosort reverse author date details` - Render as rss. Rules are the same as for DokuWiki syntax.
- `ejs?`*`template`* - Use Embedded JavaScript templating. EJS will render data according to the *`template`* with usage of the powerful javaScript language. This option must be enabled in configuration settings, it is disabled by default. *`template`* is a string designed according to the rules for ejs, for example `<$=d.toUpperCase()$>`. Variable (data) is passed to javaScript as `d`. Because of string limitations not all characters may be passed to the template directly: characters %, #, :, ,, <%= and %> must be written as `&percnt;`, `&num;`, `&colon;`, `&comma;`, `<$=` and `$>`.

Format may be used to render single variable or specific member of list or specific table column. For list or table, format must be a comma separated list of key:value pairs, where key is header

description (same as in *header*) and value is format.

filter

Filter may be applied inside brackets on any variable, list or table. If it evaluates to true, contents of the variable will be shown, otherwise not. Filter consists of one or multiple conditions separated by keywords `or` or `and`. Condition is a path compared to some value(string, numeric, boolean or null). Comparison operators are: ==, !=, <, >, <= or >=. Comparison operator and value may also be omitted. There is no precedence or brackets possible. Quotes should not be used.

Examples

Type	Example	Description
Basic variable	`%$path.to.var%`	If type of variable is string or number or boolean, it just prints it out. If it is null, it prints '(null)' by default. This can be changed in Configuration Settings. If type of the variable is array or object, this syntax prints printable members of the array or object. If there is nothing printable it prints '(array)'.
Use filter	`%$path.to.var (path.to.cond1 >= a and path.to.cond1 <b)%`	Print variable only, if filter evaluates to true.
Print JSON data	`%$path.to.var #code#%`	Print JSON raw data.
Variable is a link	`%$path.to.var #link#%`	Variable will render as link with optional title.
List all properties	`%$path.to.var {}%`	All properties of a variable will be displayed in two column table.
List specific properties	`%$path.to.var {"Property 1": subvar.path, "Property 2": subvar2.path2}%`	Similar as above, but only specific properties with custom name will be printed
List specific properties with format	`%$path.to.var { ... } #"Property 1": email#%`	Similar as above, but «Property 1» will be rendered as email.
Simple table	`%$path.to.var []%`	If path.to.var is well formed double array, then table without header will be rendered. If path.to.var is more complex, then result may be mixed.
Table with header	`%$path.to.var [] {"Column 1": subvar.path, "Column 2": subvar2.path2}%`	path.to.var should be array with similar members. *header* describes header names and members to be rendered in the table. If only blank `{}` is used, then header will be generated automatically.
Table with format	`%$path.to.var [] { ... } #"Column 2": link}#%`	Same as above, but «Column 2» will be rendered as a link.

Type	Example	Description
Table with filter	`%$path.to.var [(subvar.in.row == something)] { ... }%`	Same as above, but only those rows will be rendered, which match the filter.
Conditionally hide section	`%$-start (path.to.cond1 == true)%`*`dokuwiki-contents`*`%$end%`	Evaluate *filter* inside brackets. If true, then any *dokuwiki-contents* will be normally rendered. If false, then *dokuwiki-contents* will be hidden and if *dokuwiki-contents* contains title, it won't be shown in TOC.
Conditionally hide inline text	`%$-starti ( ... )% ... %$end%`	Same as above, but for inline DokuWiki contents. Mind extra 'i' in 'starti'.

Data snippets inside inline JSON

```
<json path=renault>{
    "cars": %$cars[(row_filter)](filter)%
}</json>
```

Inside inline json we can use `%$ path [( row_filter )]( filter )%` syntax, which will be replaced by data from existing JSON database. `[( row_filter )]` and `( filter )` are optional and can be set according to the same rules, as described above.

If data is not defined, then `null` will be used.

If data is string, then double quotes must be used. Strings can also be concatenated. For example, `"%$path.name%, %$path.age% years"` can be used for json string concatenated from two predefined strings.

Insert referenced data into array elements

This is advanced feature. Let's say, we have two data definitions: one for items database and one for item purchase history. For example:

```
<json path=items_db>{
    "apple": { "Description": "Apple", "type": "fruit" },
    "salad": { "Description": "Salad", "type": "vegetable" }
}</json>
```

```
<json path=items_purchased>[
    { "item": "apple", "quantity": 5, "date":"2019-06-22" },
    { "item": "salad", "quantity": 1, "date":"2019-06-28" },
    { "item": "apple", "quantity": 3, "date":"2019-07-10" }
]</json>
```

We want to extend each item in *items_purchased* data with item type, which is available in *items_db* data definition. We can make this:

```
<json path=items_purchased_extended>
    %$items_purchased[{item_type: items_db.{item}.type}]%
</json>
```

And get this:

```
[{ "item": "apple", "quantity": 5, "date":"2019-06-22", "item_type": "fruit"
},
 { "item": "salad", "quantity": 1, "date":"2019-06-28", "item_type":
"vegetable" },
 { "item": "apple", "quantity": 3, "date":"2019-07-10", "item_type": "fruit"
}]
```

Filters can also be used. Full definition for data snippet is:

```
%$path[(row_filter){property.path: data.path.1.{reference.path}.data.path.2,
...}](filter)%
```

Text macros

JSON define `<json ...> ... </json>` or extract `%$ ... %` patterns can be quite verbose and some lengthy attributes may repeat across multiple pages.

It is possible to use macros defined by textinsert Plugin. First install the plugin, then define some macros in it. Use `#@macro_name@#` patterns inside json define or extract patterns. JSON define or extract patterns are preprocessed for simple replacement of macros defined globally by textinsert Plugin.

For example, if macro `table_header` is defined as `{"Part description": part, "Quantity of parts": quantity}`, then you can use `%$data [] #@table_header@# %`.

Use JavaScript widgets with JSON data

JSON plugin has interface for other sub-plugins, which can use JSON data inside JavaScript widgets for example. It is relative simple to write such a plugin. For example JSONEditor plugin uses nice JSON Schema based Editor JavaScript library, which generates forms based on JSON Schema. It uses `<jsoneditor>` element, which is first rendered by JSON plugin. JSONeditor plugin has only helper.php script, which renders some additional html code into page. Data from the JSON database are already available for the widget as well as data saving mechanism.

Remote access to JSON data

DokuWiki has a XML-RPC API which can be used to access/interact with your wiki from other applications.

Available Functions

plugin.json.get

Name	plugin.json.get
Description	Generate JSON database on page and return data from the JSON_path.
Parameter (string) pagename	DokuWiki page name.
Parameter (string) JSON_path	Path on the JSON database, see attribute_path. Optional parameter, empty by default.
Parameter (boolean) addLog	If true, additional information about JSON generation will be returned. Optional parameter, false by default.
Return (object)	{'status': 'OK_or_error_description', 'data': 'JSON_data', 'log': 'JSON_loading_log' }

plugin.json.set

Name	plugin.json.set
Description	Find <json id=… element inside page and set its inline data.
Parameter (string) pagename	DokuWiki page name.
Parameter (string) json_id	Id of the json element, see attribute_id.
Parameter (array|string) data	Data to be put inside json element. Must be an array, empty string or valid JSON string.
Parameter (boolean) overwrite	If true, existing data will be overwritten. Optional parameter, false by default.
Return (string)	'OK' on success or error description.

plugin.json.append

Name	plugin.json.append
Description	Find <json id=… element inside page and append data to its inline database. Inline JSON data must be an array or empty. If empty, new array will be initialized.
Parameter (string) pagename	DokuWiki page name.
Parameter (string) json_id	Id of the json element, see attribute_id. If empty, complete page will be treated as JSON database (page must be a JSON array, empty or non-existent).
Parameter (array|string) data	JSON Data to be appended inside json element. Must be an array or valid JSON string.
Return (string)	'OK' on success or error description.

Example usage with Python

First enable DokuWiki's XML-RPC API as described. See also python-dokuwiki.

```
#pip install dokuwiki
import dokuwiki
wiki = dokuwiki.DokuWiki('https://path/to/dokuwiki', 'user', 'password')
```

```
wiki.send("plugin.json.get", "path:to:page")

person = {'firstName': 'James', 'lastName': 'Smith', 'age': 40}
wiki.send("plugin.json.set", "path:to:page", "id_attr", person)
```