struct Plugin

Why “struct” instead of “data”?

This plugin draws heavy inspiration from the data plugin. It basically tries to solve the same problem of assigning structured data to pages and build automatic aggregations from it.

So why another plugin? The data plugin proved to be very useful and versatile but had a few shortcomings:

- each page defined its own set of structured data
- there was no central way to ensure the structured data was consistent over multiple pages
- there was no easy way to modify the structured data set for multiple pages
- there was no validation for the data entered

The bureaucracy plugin helped with some of the points but not all of them. So struct tries to rethink the data plugin:

- structured data is classified in *schemas*
- a schema holds a set of *fields*
- fields have a specific *type*
- types control how the data is displayed and validated
- types have specific configuration within the schema
- schemas are managed centrally in an admin interface
- pages are assigned schemas through *namespace patterns*
- changing a schema changes it for all associated pages
- the structured data is no longer part of the page syntax
- data is only edited via a dedicated form within the standard editor or through inline editing
- it is possible to create *lookup* schemas where data is not attached to any page
- lookup schemas can be used as data source for lookup dropdowns
- all crucial code is covered by automated tests

This allows for central management of wanted structured data while keeping the functionality of the data plugin.

struct Plugin

Installation

External requirements: This plugin requires the following additional components that must be installed separately:

- SQLite Plugin

Please also note that this plugin requires **PHP 5.6 or higher**.

Install the plugin using the Extension Manager and the download above, which points to latest version of the plugin. Refer to Plugins on how to install plugins manually.

Usage

Please refer to the following pages to learn how to use the plugin:

- Schema Editing
    - Configuration
    - Types
    - Import/Export
        - CSV Import
        - CSV Export
    - Delete
    - Rename
- Assignments
- Editing Lookups
- Aggregation
    - Filters
    - Inline Editing
    - CSV Export
- Permissions
- Control-Cloud
- Bureaucracy Support
- Developer Support
- remote API

struct Plugin

Schema Editing

Schemas are used to manage the structured data that can be assigned to pages. You can think of them as a table definition. A schema defines which fields will be available and what type they have.

To create a new schema, log in as a Manager and access the “Admin” page. Select the “Struct Schema Editor” from the “Additional Plugins” section. Use the form to create a new schema.

There are two kinds of Schemas: Page Schemas and Lookup Schemas.

Page Schemas define data structures that can be assigned to pages via Assignments. Each set of data is for exactly one page it was entered for. This is what you may know from the old data plugin.

Lookup Schemas can not be assigned to pages. They store arbitrary amounts of data sets (rows) independent from any page. This is useful for data that you want to use as a Dropdown source or for data that doesn't justify a page per entry. You edit them by adding a Lookup Table Editor to a page. You can think of Lookup Schemas as a table in page where you can easily reference values from.

Existing schemas can be selected in the table of contents and can be edited there.

Important: Fields can not be deleted, only be disabled!

struct Plugin

Configuration

Each schema has some configuration that is described here:

Config	Format	Description
label	array of strings	This array contains keys for all languages available in the wiki (as defined by the translation plugin). You can use this to set translated labels for the name of the schema. These are used when showing the data or displaying the data editor. It's recommended to use short, rarely changing field names and set speaking names through this mechanism.
allowed editors	string	Comma separated list of users and @groups who may edit this schema's data (empty for all) User who want to edit Page Schemas also need at least write permission for the associated page. Superuser can always edit Struct data.

struct Plugin

Types

Types are used when defining schemas. Each field in the schema has to have a type. It configures what data is accepted in the field and how it will be displayed later on.

Available Types

- Checkbox
- Color
- Decimal
- Tag
- Text
- Page
- Date
- DateTime
- Dropdown
- Lookup
- Mail
- Media
- User
- Url
- Wiki

Configuration

Each type has it's own specific configuration options described on the type's own page linked above. In addition there's some configuration available to each type described here:

Config	Format	Description
label	array of strings	This array contains keys for all languages available in the wiki (as defined by the translation plugin). You can use this to set translated labels for the filed. These are used when showing the data or displaying the data editor. It's recommended to use short, rarely changing field names and set speaking names through this mechanism.
hint	array of strings	Similar to the label field but can be used to give some more detailed info about how to fill the field in question. Shown as tooltip.

Import/Export

You can export or import schema definitions using the Export/Import tab in the schema editor.

Export files are JSON files representing the whole schema data. When importing a schema into an existing schema that already has columns, those columns will be overwritten! To match columns between the existing schema and the ones in the JSON, the internal `colref` IDs are used (those reflect the order in which columns where originally created).

When importing a schema with localized labels into a wiki that does not have the same languages, all labels for unknown languages will be ignored.

See also:

- CSV Import
- CSV Export

CSV Import

The Schema Editor has a Import/Export tab offering support for importing raw data into a lookup schema (page schemas are not supported currently) using a CSV file.

- The first row has to contain headers matching the field names of the lookup schema you're importing into
- Columns with headers not matching any field name are ignored
- Only commas are supported as separator
- Column contents should be enclosed with double quotes
- Double quotes can be escaped with a backslash (`\`) or using standard doubling of the double quotes (`""`)
- For multi fields, the column content will be split at commas (no support for any escaping)
- Content has to be in UTF-8 encoding

The imported data is added to whatever data is already stored in the lookup schema.

You can use the CSV Export on the same tab to get an example CSV file.

struct Plugin

CSV Export

Aggregations

All Aggregations do show a small "Export as CSV" link below the result table.

Clicking it will export all data of that aggregation, applying the current filters and columns. Limits and offsets will be ignored and all matching data will be exported.

CSV-Export can be disabled with the `csv: 0` option.

Exports use the following CSV format:

- Separator: `,`
- Text Delimiter: `"`
- Line-Endings: CRLF (Windows)
- Charset: UTF-8

Admin

The CSV export above will export exactly what a (possibly filtered) aggregation shows. The Schema Editor Import/Export tab offers a second way to export all raw data of a schema as CSV.

Data exported here can also be reimported into lookup schemas.

Delete Schemas

Schemas can be deleted through the appropriate tab on the Schema Editor admin page. This will delete all existing data associated with the schema including historic data. It's as if the schema never existed.

To avoid accidental deletion, the name of the schema has to be typed out.

struct Plugin

Rename

Schemas can be renamed by using the following code-snippet and replacing `OLD` with the old name of the schema and `NEW` with the new name of the schema and then executing it the struct-section of the administrative menu of the sqlite plugin.

```
UPDATE schemas SET tbl = 'NEW' WHERE tbl = 'OLD';
UPDATE schema_assignments SET tbl = 'NEW' WHERE tbl = 'OLD';
UPDATE schema_assignments_patterns SET tbl = 'NEW' WHERE tbl = 'OLD';
ALTER TABLE 'data_OLD' RENAME TO 'data_NEW';
ALTER TABLE 'multi_OLD' RENAME TO 'multi_NEW';
```

Please be aware that you still have to manually update all your aggregations.

Assignments

What structured data can be added to a page depends on the schema assigned to that page. The assignment is done through patterns - if a page matches a configured pattern, the appropriate schema will be used.

To manage assignment patterns, log in as a Manager and access the "Admin" page. Select the "Struct Schema Assignments" from the "Additional Plugins" section.

The assignment manager shows a list of currently active patterns and the schemas they assign. A form at the bottom allows to add a new pattern to assign a selected schema to all matching pages. Assignments can be done on a page or namespace basis or using a regular expression.

A page can have multiple schemas assigned.

Only page schemas, not lookups can be assigned to pages.

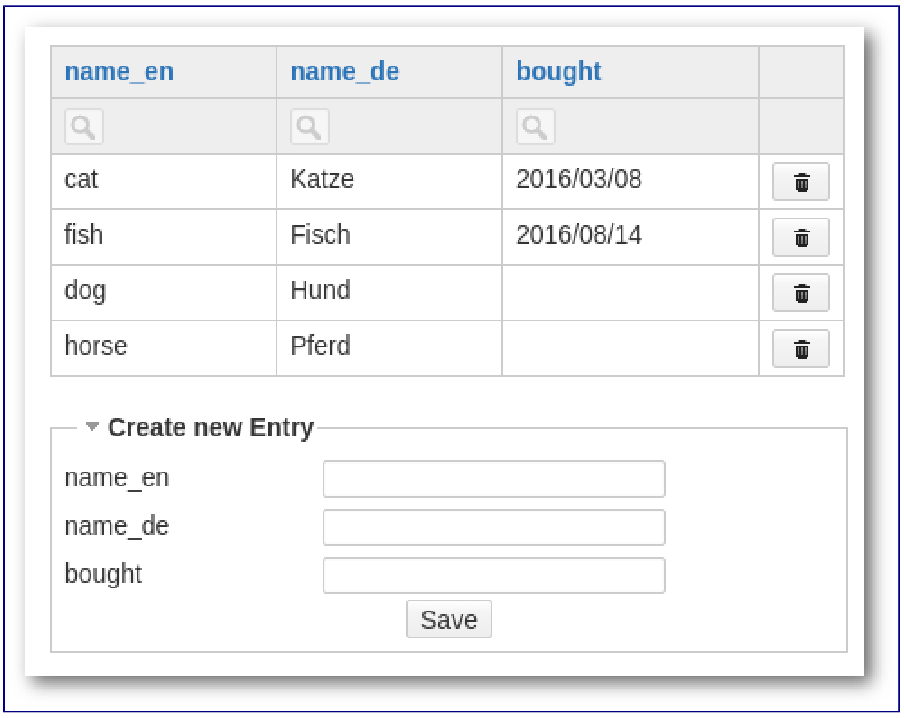
struct Plugin

Editing Lookups

Lookup Schemas store arbitrary data not tied to any page. To create such data you need to create a lookup editor through a special syntax:

```
---- struct lookup ----
schema: someschema
----
```

This creates a list of all data currently stored in the schema. Existing data can be edited through Inline Editing or deleted through the trash can button. New data rows can be added through the form below the table.



name_en	name_de	bought	
cat	Katze	2016/03/08	🗑
fish	Fisch	2016/08/14	🗑
dog	Hund		🗑
horse	Pferd		🗑

Create new Entry

name_en	
name_de	
bought	

The `struct lookup` syntax accepts most of the Aggregation syntax (summarize/row numbers is disabled for lookup). However you can only specify the one lookup schema you want to edit and you can not select columns. All of this can still be done in normal aggregations of course (without the add and delete feature then).

struct Plugin

Aggregation

Table

Aggregation of structured data is the main purpose of the plugin. Aggregation is done through a simple syntax which will result in a dynamic table listing all pages and associated data matching certain criteria.

The basic syntax looks like this:

```
---- struct table ----
schema: schema_name
cols: %pageid%, other, cols
----
```

The aggregation-syntax is very similar to that of the data plugin. The main difference is that you have to specify a schema name (or more than one).

The keyword before the colon is a configuration option and the value after it is the actual setting. To make it more fault tolerant, multiple option names are frequently possible. Here is a list of all available options:

struct Plugin

Option(s)	Required ?	Description
schema from	yes	These are the schemas from which you want to display the data.
cols select	yes	These are the attributes you want to display. These are the same *names* you used in the Data Entry part
head header headers	no	If specified, these names will be used in the table headers instead of the column names
max limit	no	How many rows should be displayed. If more rows are available the table will be made browsable. If not given all matching rows are shown
sort order	no	By what column should the table be sorted initially? Prepend a ^ to reverse the sorting
filter where filterand and	no	Filter by a column value. You may specify this more than once, multiple filters will be ANDed. (see also filters)
filteror or	no	Like filter, but multiple instances will be ORed (see also filters)
dynfilters	no	Set to `1` to enable a row of input fields for dynamically filtering the table
summarize	no	Set to `1` to calculate sum of columns
align	no	List of column alignments. The alignments can be `left` (`l`), `center` (`c`) or `right` (`r`)
rownumbers	no	Set to `1` to show row numbers
width widths	no	Comma-separated list of column widths to use. units and percentages can be given
csv	no	Set to `0` to disable the CSV export for this aggregation

Please note that you can join data from multiple page schemas (as long as they are assigned on the same pages), but can only run aggregations on a single lookup schema.

struct Plugin

Aliases

When selecting data from multiple schemas which contain the same field names, you have to prefix them with the schema name:

```
---- struct table ----
schema: schema1, schema2
cols: %pageid%, schema1.name, schema2.name
----
```

You can use aliases to reference schemas to type less. Aliases are separated by a space from the schema name:

```
---- struct table ----
schema: schema1 A, schema2 B
cols: %pageid%, A.name, B.name
----
```

Please see filters to learn more about filtering.

Special Columns

The following special column names can be used to access non-field columns:

Special Name	Description
`%pageid%`	The page name of the page holding the matching struct data
`%title%`	The title (first heading) of the above page
`%lastupdate%`	The date and time when the page or data was last updated
`%lasteditor%`	The user that last changed the page

List

As an alternative to the table, you can also aggregate the data in list form. This is largely analogous to Data Lists of the data-plugins

The basic syntax for list aggregations is very similar to table aggregations and looks like this:

```
---- struct list ----
schema: schema_name
cols: %pageid%, other, cols
----
```

struct Plugin

Lists support a new option: `sepbyheaders`. If you set this option to `1` and define headers for the columns, than each column will be prefaced with their header.

Several options from the table-aggregation are not supported for the list-aggregation:

Option	supported for lists
schema from	yes
cols select	yes
head header headers	with `sepbyheaders` option
max limit	yes
sort order	yes
filter where filterand and	yes
filteror or	yes
dynfilters	no
summarize	no
align	no
rownumbers	no
width widths	no
csv	no

struct Plugin

Filters

Filters are used to filter aggregations.

Fixed Filters

Fixed filters are added through the `filter` and `filteror` options. They expect a column name, a comparator and a value to compare with.

Example:

```
---- struct table ----
schemas  : projects A
cols     : %pageid%, product, budget, team
filter   : product = Fantastico Basic
filteror : product = Fantastico Professional
----
```

Column names can of course reference the full column name (eg. `projects.product`) or use defined aliases (eg. `A.product`).

Comparators

For filtering, multiple comparators are possible:

Comparator	Meaning
=	Exact match
!= or <>	Does not exactly match
<	Less than
<=	Less or equal than
>	Greater than
>=	Greater or equal than
~	Wildcard match. Use a `*` as wildcard. Like `Apple*` to match `Apple Pie` and `Apple Computer`; e.g. `dessert~ *Pie`. Case insensitive.
`*~`	Wildcard match. Look for matches containing search term; e.g. `dessert*~ Pi` match `Apple Pie`
`!~`	Negative Wildcard match. Select everything that does not match the expression.
`=*`	Regular expression search

struct Plugin

Value Placeholders

When defining fixed filters it is sometimes useful to compare against semi-dynamic values. This is where value placeholders come in handy. You simply use the placeholder instead of a value when creating the filter.

The following example prints all projects the current user is a team member of:

```
---- struct table ----
schemas  : projects A
cols     : %pageid%, product, budget, team
filter   : team = $USER$
----
```

Placeholder	Description
`$USER$`	currently logged in user
`$TODAY$`	Today's date in `Y-m-d` format
`$ID$`	The page's full page ID (of the page the aggregation is defined on)
`$PAGE$`	The page's page name without a namespace
`$NS$`	The page's namespace
`$STRUCT.`*`<schema>.<field>`*`$`	The struct data saved for the current page in given field.

struct Plugin

Dynamic Filters

Aggregations can be "live" filtered by either adding the `dynfilters` option or passing the right parameters.

The following example adds filter input fields for all selected columns.

```
---- struct table ----
schema     : projects A
cols       : %pageid%, product, budget, team
dynfilters : 1
----
```

Filters created through this method always use the `*~` comparator, eg. look for the search term anywhere in the column's data.

Dynamic filters can also be added through the `flt` parameter. The parameter is an array type, with the column name and comparator combined in the key and the comparison value in the value part.

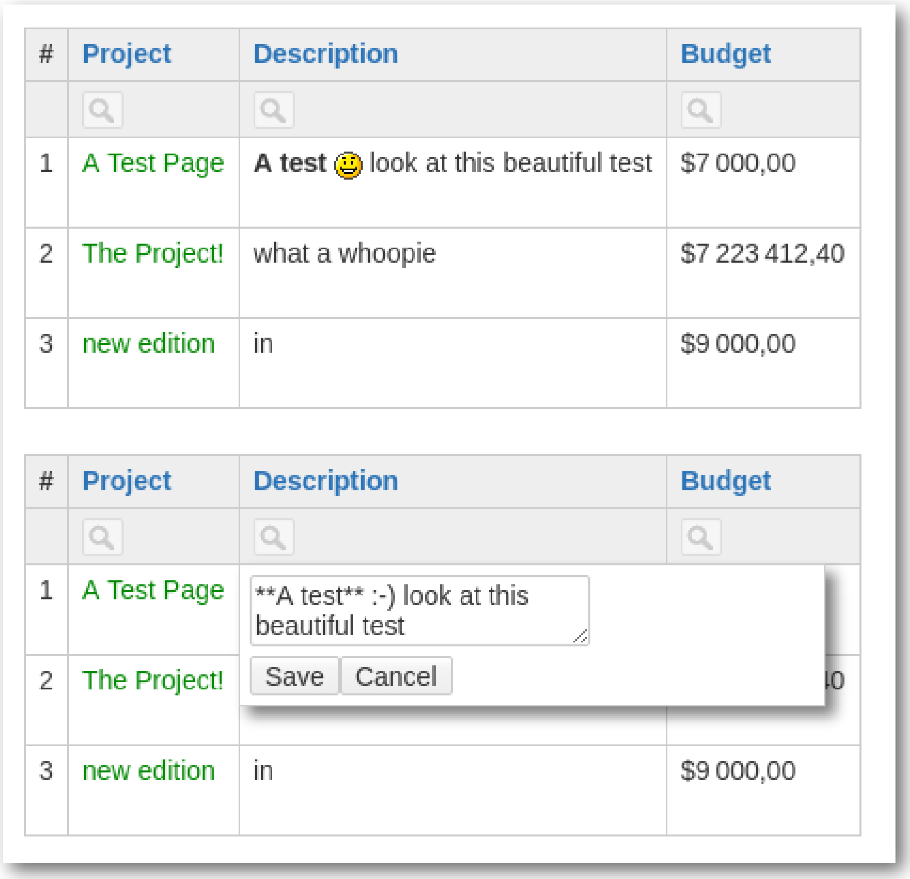
Example: `?flt[projects.budget>]=10000`

struct Plugin

Inline Editing

When using an Aggregation Table to show data from different pages or a Lookup Editor, the Struct plugin allows you to edit the displayed data directly in the displayed Table.

To edit the data in a cell, simply double click the cell. If a cell contains linked content (links, images, etc) you need to double click the cell itself, not the content. Eg. click on the free space.



#	Project	Description	Budget
1	A Test Page	**A test** 🙂 look at this beautiful test	$7 000,00
2	The Project!	what a whoopie	$7 223 412,40
3	new edition	in	$9 000,00

#	Project	Description	Budget
1	A Test Page	**A test** :-) look at this beautiful test	
2	The Project!		0
3	new edition	in	$9 000,00

Restrictions:

The editor will only pop up if

- you have write permissions to the page the data is attached to
- that page isn't locked
- it is actually struct data (you can't edit `pageid` or `title` columns)

struct Plugin

CSV Export

Aggregations

All Aggregations do show a small "Export as CSV" link below the result table.

Clicking it will export all data of that aggregation, applying the current filters and columns. Limits and offsets will be ignored and all matching data will be exported.

CSV-Export can be disabled with the `csv: 0` option.

Exports use the following CSV format:

- Separator: `,`
- Text Delimiter: `"`
- Line-Endings: CRLF (Windows)
- Charset: UTF-8

Admin

The CSV export above will export exactly what a (possibly filtered) aggregation shows. The Schema Editor Import/Export tab offers a second way to export all raw data of a schema as CSV.

Data exported here can also be reimported into lookup schemas.

Permissions

Read permissions to data stored in schemas depends on the type of schema:

- Data in page-schemas is metadata associated with the specific page and hence shares the page's access rules.
- Data in lookup-schemas is not associated with any page and therefore is basically always public.

You can restrict who can edit data in lookup-schemas in the schema editor.

struct Plugin

Control Clouds

Analogous to the Tag Cloud of the data-plugin, there is a syntax to create tag-clouds to control aggregations.

This syntax will display the values of a given struct field as a tag cloud. Each value will link back to the current page (unless configured otherwise by `target` option). The page should also contain a struct table - this table will then be filtered for all entries matching the selected tag.

Example:

```
---- struct cloud ----
schema: project
field: employees
min: 2
limit: 20
----
```

The above code would display a cloud of employees assigned to at least two different projects. A maximum of the 20 most busiest employees are shown.

These are the possible options for the cloud:

Option(s)	Required?	Description
schema from tables	yes	What Schema is the source of the data?
field select cols col	yes	What attribute is used to build the cloud?
limit max	no	Maximum number of tags to display. If not given all will be displayed
min	no	Minimum count a tag must have. If not given all will be shown
page target	no	Give a page which contains the Data Table to control. If not given the current page is used
summarize	no	Set to 1 to show sum of field

struct Plugin

Bureaucracy Support

The struct plugin supports integration with the Bureaucracy Plugin. This allows reusing the input mechanisms of the different types in bureaucracy forms and creation of structured data when pages are created through Bureaucracy's template action.

Single Field

You can use any field from any defined schema as a a field in a bureaucracy form. To do so simply use the type `struct_field` and specify the field name in the form of `<`*`schema`*`>.<`*`name`*`>`.

```
<form>
Action template templates:product product :
struct_field "products.product" @
submit "Create new product"
</form>
```

The defined (translated) label will automatically be used for the field as well as all the validation rules configured in the schema.

You can use the value entered in a struct field as any other value for templating. Eg. you can use `@@products.product@@` to access the value of the example above.

All struct values will be assigned to the resulting pages of (when using the bureaucracy template action), provided the schema assignments match.

Complete Schema

Instead of adding individual fields to a bureaucracy form as described above, you can also add a whole schema to the form. To do so, simply use the `struct_schema` command.

```
<form>
Action template templates:product product :
textbox name @
struct_schema "product" !
submit "Create new product"
</form>
```

Please note that all additional parameters given will be added to each individual field. Eg. you can only make all fields of the schema mandatory or optional.

struct Plugin

Note

At the Moment it isn't possible to pre-fill fields in the Bureaucracy - fieldset.

Development Support

The Struct Plugin allows other plugin developers to hook into it's functionality, adding additional features.

Registering new Types

The plugin signals the `PLUGIN_STRUCT_TYPECLASS_INIT` event which can be intercepted by Action Plugins. There is no default action. The passed `$data` is an associative array listing the available Types and their respective classes. It looks like this:

```
array(
    'Checkbox' => 'dokuwiki\\plugin\\struct\\types\\Checkbox',
    'Date' => 'dokuwiki\\plugin\\struct\\types\\Date',
    'DateTime' => 'dokuwiki\\plugin\\struct\\types\\DateTime',
    'Decimal' => 'dokuwiki\\plugin\\struct\\types\\Decimal',
    'Dropdown' => 'dokuwiki\\plugin\\struct\\types\\Dropdown',
    'Lookup' => 'dokuwiki\\plugin\\struct\\types\\Lookup',
    'Mail' => 'dokuwiki\\plugin\\struct\\types\\Mail',
    'Media' => 'dokuwiki\\plugin\\struct\\types\\Media',
    'Page' => 'dokuwiki\\plugin\\struct\\types\\Page',
    'Tag' => 'dokuwiki\\plugin\\struct\\types\\Tag',
    'Text' => 'dokuwiki\\plugin\\struct\\types\\Text',
    'Url' => 'dokuwiki\\plugin\\struct\\types\\Url',
    'User' => 'dokuwiki\\plugin\\struct\\types\\User',
    'Wiki' => 'dokuwiki\\plugin\\struct\\types\\Wiki',
);
```

Plugins may add their own `Type` ⇒ `class` pairs here. The class name has to be fully qualified and needs to be loadable by DokuWiki's Autoloader. The class needs to inherit from AbstractBaseType or one of its subclasses.

Please refer to the existing types' source code to see how to implement your own type.

Examples of plugins implenting this are: structstatus Plugin

struct Plugin

Allow Additional Config Keys

When you write a plugin doing it's own aggregation, you might want to reuse the ConfigParser class. By default this class will throw an Exception when a config key is encountered that it does not understand. You can handle the `PLUGIN_STRUCT_CONFIGPARSER_UNKNOWNKEY` event and prevent the default (of throwing the exception). You can add your own config to the `config` array in the event.

```
array(
  'config' => &$this->config,
  'key' => 'the unknown config key',
  'val' => 'the value for that key'
)
```

Examples of plugins implenting this are: structgantt Plugin

struct Plugin

remote API

The struct plugin exposes several methods that can be accessed via the XML-RPC API. You can find the current code in the remote.php

Methods

- **`getData()`** *Get the structured data of a given page*
  1. parameter: **string** *The page to get data for*
  2. parameter: **string** *The schema to use, empty for all*
  3. parameter: **int** *A timestamp if you want historic data (0 for now)*
  4. returns: **array** `('schema' ⇒ ( 'fieldlabel' ⇒ 'value', …))`
- **`saveData()`** *Saves data for a given page (creates a new revision)*
  If this call succeeds you can assume your data has either been saved or it was not necessary to save it because the data already existed in the wanted form or the given schemas are no longer assigned to that page.
  1. parameter: **string** *page*
  2. parameter: **array** `('schema' ⇒ ( 'fieldlabel' ⇒ 'value', …))`
  3. parameter: **string** *summary*
  4. return **bool** *returns always true*
- **`getSchema()`** *Get info about existing schemas columns*
  Returns only current, enabled columns
  1. parameter: **string** *the schema to query, empty for all*
  2. return **array**
- **`getAggregationData()`** *Get the data that would be shown in an aggregation*
  1. parameter: **array** *array of strings with the schema-names*
  2. parameter: **array** *array of strings with the columns*
  3. parameter: **array** *array of arrays with* `['logic'⇒ 'and'|'or', 'condition' ⇒ 'your condition']`
  4. parameter: **string** *string indicating the column to sort by*
  5. return **array** *array of rows, each row is an array of the column values*

struct Plugin

Type Checkbox

The `Checkbox` type allows to specify a value that can be selected or not. When the type is set to `Multi` multiple checkboxes are displayed.

Configuration

The following configuration is available in addition to the basic type configuration:

Config	Format	Description
values	string	A comma separated list of values to pick from. If the type is not set to `Multi` only the first value is used.

Type Color

The `Color` type stores a color as RGB hex value. This type is probably not very useful on it's own but can be used by other plugins.

The editor relies on the input[type=color] feature in modern . On browsers not supporting it (Internet Explorer) a simple text field will be displayed.

Configuration

The following configuration is available in addition to the basic type configuration:

Config	Format	Description
default	string	The default color. This color will not be saved - it's considered to be an empty selection. Defaults to `#ffffff` (white)

struct Plugin

Type Decimal

The `Decimal` type holds decimal numbers also known as float or double. Upper and lower bounds can be set through the config. The decimal point and thousands separator can be configured. Values can be rounded and trailing zeros be trimmed.

Internally values are always stored with a dot as decimal point.

Values can be entered with a dot or comma, but may not contain thousand separators.

Configuration

The following configuration is available in addition to the basic type configuration:

Config	Format	Description
min	integer	The number has to be at least this value. Be sure to specify decimals with a `.` as decimal point!
max	integer	The number may be at most this value. Be sure to specify decimals with a `.` as decimal point!
roundto	integer	The number of decimals the values should be rounded to. Use `-1` for no rounding.
decpoint	char	The character to use as a decimal point in output. You probably want to use either `.` or `,`. Regardless of this setting, users may always enter values with either of the two.
thousands	char	The character to use as a thousands separator. The default is a thin non-breaking space.
trimzeros	boolean	If true, trailing zeros are stripped from the decimals.
prefix	string	This text will be prepended to the saved value - useful for currency symbols
postfix	string	This text will be appended to the saved value - useful for currency symbols

struct Plugin

Type Tag

The `Tag` type accepts “tag” names - arbitrary strings to categorize a page. Each tag will be linked to the same configured page containing an aggregation with a parameter filtering the aggregation to the given tag. Autocompletion completes with existing tags. When filtering by tags, comparisons are made case insensitive and without spaces.

Configuration

The following configuration is available in addition to the basic type configuration:

Config	Format	Description
page	string	The page containing an aggregation displaying data from the same schema as the tag. Leaving this empty defaults to a page named after the field.
autocomplete.mininput	integer	Autocompletion will only return results when at least this many character have been entered
autocomplete.maxresult	integer	The maximum number of suggestions shown by the autocompletion

Type Text

The `Text` type is the most basic field type. It allows arbitrary (one line) textual input. The output will not contain any formatting.

Configuration

The following configuration is available in addition to the basic type configuration:

Config	Format	Description
prefix	string	This text will be prepended to the saved value
postfix	string	This text will be appended to the saved value

struct Plugin

Type Page

The `Page` type accepts a wiki page. It will be linked in the output. Autocompletion helps with data entry.

Configuration

The following configuration is available in addition to the basic type configuration:

Config	Format	Description
usetitles	boolean	When enabled, the title (first heading) of the page will be shown and also be used to filter against
autocomplete.namespace	string	Autocomplete searches in this namespace and below only. Can be a relative namespace
autocomplete.postfix	string	Autocomplete looks for pages ending with this postfix only. Useful if you want people to specify a namespace and then link to a certain page within.
autocomplete.mininput	integer	Autocompletion will only return results when at least this many character have been entered
autocomplete.maxresult	integer	The maximum number of suggestions shown by the autocompletion

Type Date

The `Date` type accepts a date in the format YYYY-MM-DD. The output format can be controlled through a config. Data entry is helped through a date picker.

Configuration

The following configuration is available in addition to the basic type configuration:

Config	Format	Description
format	string	A date format string for formatting the date in the output
prefilltoday	boolean	When set to true, the input will be automatically filled with the current date when empty. Useful to have something like a “creation date”.

struct Plugin

Type DateTime

The `DateTime` type accepts a date and time in the format `YYYY-MM-DD HH:MM:SS`. The output format can be controlled through a config. Data entry is helped through a date picker, the time has to be entered manually.

Configuration

The following configuration is available in addition to the basic type configuration:

Config	Format	Description
format	string	A date format string for formatting the date in the output
prefilltoday	boolean	When set to true, the input will be automatically filled with the current date and time when empty. Useful to have something like a “creation date”.

Type Dropdown

The `Dropdown` type allows you to specify a list of values that can be selected.

When the type is set to `Multi` multiple values can be selected.

If you want to use data from another schema as values, use the Lookup type.

Configuration

The following configuration is available in addition to the basic type configuration:

Config	Format	Description
values	string	A comma separated list of values to pick from. Ignored if `schema` and `field` below are defined.

struct Plugin

Type Lookup

The `Lookup` type works similar to the DropDown type. But values for the dropdown are selected from another schema (either lookup or page).

When the type is set to `Multi` multiple values can be selected.

You can reference different fields based on the language using the `$LANG` variable which might be useful for multilingual wikis using the Translation Plugin.

Configuration

The following configuration is available in addition to the basic type configuration:

Config	Format	Description
schema	string	A lookup schema to pull the values from
field	string	The field of the above schema to use as value display. Use `$LANG` placeholder to refer to a language dependent field

Type Mail

The `Mail` type accepts an email address. It will be obfuscated and linked as configured through the Configuration Setting: mailguard.

Configuration

The following configuration is available in addition to the basic type configuration:

Config	Format	Description
prefix	string	This text will be prepended to the saved value
postfix	string	This text will be appended to the saved value

struct Plugin

Type Media

The `Media` type accepts internal and external media files (like images, video or PDF).

Configuration

The following configuration is available in addition to the basic type configuration:

Config	Format	Description
mime	string	A comma separated list of MIME types to accept. MIME types can be specified as beginning part only (like `image/` to allow all images)
width	int	The pixel width to use when embedding media in the page output
height	int	The pixel height to use when embedding media in the page output. Will be auto computed when not specified
agg_width	int	The pixel width to use when embedding media in aggregations. Defaults to `width`
agg_height	int	The pixel height to use when embedding media in aggregations. Defaults to `height`

Type User

The `User` type accepts DokuWiki user names. They will be displayed according to Configuration Setting: showuseras. Autocompletion helps with input.

Configuration

The following configuration is available in addition to the basic type configuration:

Config	Format	Description
existingonly	boolean	When true, only existing users can be added. Otherwise any value is accepted.
autocomplete.mininput	integer	Autocompletion will only return results when at least this many character have been entered
autocomplete.maxresult	integer	The maximum number of suggestions shown by the autocompletion
autocomplete.fullname	boolean	Should the autocomplete search in full names as well (disable when too slow in your backend)

struct Plugin

Type Url

The `Url` type expects an external as value. It will be linked in the output. The final (including the configured `pre-` and postfixes) has to validate against the configured URL Schemes.

Configuration

The following configuration is available in addition to the basic type configuration:

Config	Format	Description
autoscheme	string	If the does not contain a scheme (like `http://`) this scheme will be prepended
prefix	string	This text will be prepended to the saved value
postfix	string	This text will be appended to the saved value

Type Wiki

The `Wiki` type accepts wiki syntax in a small text area. The output will be rendered just like wiki syntax is in normal pages.

Please note: you should avoid using this type when possible. A new Parser/Renderer process needs to be instantiated for each field of this type displayed. This is an expensive operation and might slow down the display of aggregations.

Configuration

The following configuration is available in addition to the basic type configuration:

Config	Format	Description
prefix	string	This text will be prepended to the saved value
postfix	string	This text will be appended to the saved value