Панель инструментов

Панель инструментов делает DokuWiki простым в использовании даже для новичков. Иногда вам может понадобиться расширить панель инструментов еще одной кнопкой. Эта страница расскажет вам немного о том, как работает панель инструментов и как ее расширить.

How the toolbar is built (and cached)

The toolbar is completely built through JavaScript. Which buttons the toolbar contains, is read from an array called `toolbar`. This array is initialized in `/lib/exe/js.php`. This is simply done by converting the PHP array structure defined in **`inc/toolbar.php`** to JavaScript using JSON.

This means the toolbar gets cached like any other JavaScript in DokuWiki. Whenever you do changes affecting the toolbar you should invalidate DokuWiki's cache (eg. `touch conf/local.php`) and refresh your browser cache while editing a page (Hit Shift/Ctrl-Reload and ignore the post warning).

The toolbar array

The mentioned `toolbar` array contains one entry per button. The order of the entries defines the position of the button, with the first entry defining the most left button.

Each entry is an associative array itself, defining the behavior of the button. The following table explains the available fields for the entries.

Name	**Description**
`type`	button action type: `format` ⇒ wiki markup `mediapopup` ⇒ popup window `picker` ⇒ picker menu `signature` ⇒ signature generation and insertion e.g. `yourtype` ⇒ looks for your custom action function `addBtnActionYourtype`
`title`	title of the button, displayed on mouseover
`icon`	icon to use for the button (based in `DOKUBASE/lib/images/toolbar/`)
`key`	(optional) Hotkey for the button, a value of n here would result in the button being pressed when the user presses `ALT + n` in his browser. (See also the access keys used by DokuWiki)
`class`	(optional) classname for styling buttons
`id`	(optional) id added to button, and with a postfix to icon e.g. `<id>_ico`
`format` type specific parameters	
`open`	opening tag of wiki markup to insert, cursor will be placed between opening and closing tag
`sample`	example input to be placed between opening and closing tag, will be automatically selected to be overwritten on first input
`close`	closing tag of the wiki markup to insert, cursor will be placed between opening and closing tag
`insert`	wiki markup to insert (for full substitutions)
`mediapopup` type specific parameters	

Name	Description
`url`	URL of the popup to open
`name`	internal name of the popup to open
`options`	additional options for the popup window (comma-separated name-value-pairs valid as parameters for the JS function `window.open`
`picker` type specific parameters	
`list`	list of insertable items. can either be an associative array (with the item to insert as the key and an image located in `icobase` to use as the value) or just a simple array (with the text-items to insert directly displayed)
`icobase`	subdirectory for item images (based in `DOKUBASE/lib/images/`).

A look at inc/toolbar.php should give you enough examples of what can be defined. Because the array is simply translated from PHP to JavaScript, the above description applies to the PHP array as well as to the resulting JavaScript array from which the toolbar is finally built.

Extending the Toolbar...

The toolbar can be customized in two ways - via JavaScript or PHP. Which method you choose basically depends on your skills and familiarity with the language at hand. For the PHP approach, you will always need to write a plugin, the JavaScript method can be used from a userscript as well.

...using PHP

To extend the toolbar from your plugin you need to write a Action Plugin which registers as a handler for the TOOLBAR_DEFINE event with an AFTER advice. Your handler function will receive the toolbar array as described earlier.

Create your button structure according to the above information and add it to the received event data.-function in `inc/toolbar.php`.

Example:

```
function register(Doku_Event_Handler $controller) {
    $controller->register_hook('TOOLBAR_DEFINE', 'AFTER', $this,
'insert_button', array ());
}

// ...

/**
 * Inserts a toolbar button
 */
function insert_button(Doku_Event $event, $param) {
    $event->data[] = array (
        'type' => 'format',
        'title' => $this->getLang('abutton'),
        'icon' => '../../plugins/example/abutton.png',
        'open' => '<abutton>',
```

```
        'close' => '</abutton>',
        'block' => false,
    );
}
```

Try to use custombuttons

...using JavaScript

Adding a button using JavaScript is similar to doing it in PHP. You just extend the `toolbar` array.

Because user and plugin scripts will be loaded on all requests, but the toolbar is initialized on editing only, you need to make sure the toolbar array does exist first. Do this by checking if `window.toolbar` is defined. If it is, you can add your button at the end of the array.

```
if (typeof window.toolbar !== 'undefined') {
    toolbar[toolbar.length] = {
        type: "format",
        title: "A Button",
        icon: "abutton.png", // located in lib/images/toolbar/
        key: "",
        open: "<abutton>",
        sample: "Text between the tags",
        close: "</abutton>"
    };
}
```

Here is another example defining a picker dropdown (assumes plugin context):

```
if (typeof window.toolbar !== 'undefined') {
    var notes_arr = {
        //'insertion string as key' : '[path/]filename.extension of the
icon'
        '<note></note>\\n':           'note.png',
        '<note tip></note>\\n':       'tip.png',
        '<note important></note>\\n': 'important.png',
        '<note warning></note>\\n':   'warning.png'
    };

    toolbar[toolbar.length] = {
        type: "picker",
        title: LANG.plugins.note.note_tb_title, // localisation
        icon: '../../plugins/note/images/toolbar.png',  //where in
lib/images/toolbar/ the images are located
        key: "n", //access key
        list: notes_arr,
        icobase: "../plugins/note/images/toolbar" // subdir of lib/images/
where images can be found.
    };
```

```
}
```

You can register a simple global onclick function for your button by sticking to the naming convention `tb_<yourButtonType>`. Here is an example for the `insert` type from lib/scripts/toolbar.js

```
function tb_insert(btn, props, edid) {
    insertAtCarret(edid,fixtxt(props.insert));
    pickerClose();
    return false;
}
```

Some notes:

- Be aware of the default locations of the images: `lib/images/toolbar/` and `lib/images/`
- The available options for access keys depends on the access keys used by DokuWiki and the whether they are reserved by a browser or operation system as short cut. There are many differences.
- More about javascript localization

Dynamic Data

The above methods work well for adding static buttons that add static data. When you need to do something more dynamic like adding the current date, you can not simply extend the `toolbar` array. Instead, you need to dynamically add your button into the DOM using JavaScript.

The files lib/scripts/edit.js, lib/scripts/toolbar.js and lib/scripts/linkwiz.js contains some useful functions to help you with this.

When adding to the `toolbar` array you must set a custom value for `type`. You must use characters that are valid in a function name and should capitalize on the first character. When the toolbar is constructed it looks at the value of `type` and appends it to the string `addBtnAction` before calling your code. So if you set `type` to `Click` the toolbar will call `addBtnActionClick`. This is an important note because if two plugins use the same `type` they will conflict with each other and cause strange behavior (such as both buttons doing the exact same thing).

The returned string is «the id for in aria-control». This is explained further in Accessible Rich Internet Applications.

Here is an example partly from the toolbar picker:

```
/**
 * With the first function we create a new button type called Click
 *
 * the function name must be addBtnAction<Your type name>
 * in our case it is addBtnActionClick
 *
 * in the other function we simply use the simple toolbar method with the
new type
 *
 * you can easily extend it to complex scripts like the link wizard etc
```

```
 */

/**
 * Add button action for your toolbar button
 *
 * @param  {jQuery}   $btn  Button element to add the action to
 * @param  {Array}    props Associative array of button properties
 * @param  {string}   edid  ID of the editor textarea
 * @return {string}   If button should be appended return the id for in
aria-controls,
 *                    otherwise an empty string
 */
function addBtnActionClick($btn, props, edid) {
    // initialize stuff if required
    // ...

    $btn.click(function() {
        // your click handler
        alert('hey you clicked me');
        return false;
    });

    return 'click';
}

// add a new toolbar button, but first check if there is a toolbar
if (typeof window.toolbar !== 'undefined') {
    window.toolbar[window.toolbar.length] = {
        type: "Click", // we have a new type that links to the function
        title: "Hey Click me!",
        icon: "../../plugins/click/clickme.png"
    };
}
```

Tips

- If you don't want to append your button at the end of the toolbar, you just need to insert your data into the `toolbar` array at the position where you want your button, shifting the other buttons to the right.
- There are some predefined global variables available which you can access from your Javascript. For example JSINFO contains the page id and namespace, usage: `JSINFO.namespace`. This might be handy when doing some more dynamic stuff.
- **Important**: If the button does not appear, expire the cache of your DokuWiki by saving your config in the Config Manager or touching the `conf/local.php` of your wiki AND delete your browser cache (Ctrl+F5 or Ctrl+R)! Then reload the editor page and your button should appear.

Removing Buttons

If you want to remove some of the toolbar buttons, you can do so by adding the following code into

`/conf/userscript.js`. Note that this is not in any function, simply append this code to the end of the file:

/conf/userscript.js

```
if (typeof window.toolbar !== 'undefined') {
   var blacklist = ["Bold Text", "Italic Text", "Underlined Text",
"Monospaced Text", "Strike-through Text",
   "Same Level Headline", "Lower Headline", "Higher Headline", "Select
Headline", "Internal Link", "External Link",
   "Horizontal Rule", "Add Images and other files (opens in a new
window)", "Smileys", "Special Chars", "Insert Signature"];
    window.toolbar = window.toolbar.filter(function(elem){ return
jQuery.inArray(elem.title, blacklist) === -1; });
}
```

This is from my personal wiki where I removed almost all the buttons. You can figure out what the title of your button is by running the following snippet in a javascript console:

```
for (i in window.toolbar) { console.log(toolbar[i].title)}
```

Then take the string and append it to the 'blacklist' array in the code.

See also the toolbuttondel plugin for removing toolbar items.

CSS Spritemap

If you need to reduce the number of server requests for downloading micro-icons, you can use the following style fix.

1. 15 icons of the 20 in the `/lib/images/toolbar/` folder are used in a constant order and can be combined into a vertical ribbon named **spritemap**.
    1. You can take the finished spritemap here and put it in `/lib/images/toolbar/` named as `toolbar.png`.
2. CSS used to show a certain point on spritemap.
    1. Add the following CSS rules to your local style file `/conf/userstyle.less` or template's style file `/lib/tpl/<name>/css/design.less`:

```
#tool__bar {
    button:nth-of-type(1), button:nth-of-type(2), button:nth-of-type(3),
button:nth-of-type(4), button:nth-of-type(5),
    button:nth-of-type(6), button:nth-of-type(7), button:nth-of-type(8),
button:nth-of-type(9), button:nth-of-type(10),
    button:nth-of-type(11), button:nth-of-type(12), button:nth-of-type(13),
button:nth-of-type(14), button:nth-of-type(15) {
        background: url(/lib/images/toolbar/toolbar.png);
        height: 16px;
        margin: 4px 7px 3px 3px;
```

```
        vertical-align: middle;
        width: 16px;
        img {
            display: none;
        }
    }
    button:nth-of-type(1) { background-position: 0 0; }
    button:nth-of-type(2) { background-position: 0 -16px; }
    button:nth-of-type(3) { background-position: 0 -32px; }
    button:nth-of-type(4) { background-position: 0 -48px; }
    button:nth-of-type(5) { background-position: 0 -64px; }
    button:nth-of-type(6) { background-position: 0 -80px; }
    button:nth-of-type(7) { background-position: 0 -96px; }
    button:nth-of-type(8) { background-position: 0 -112px; }
    button:nth-of-type(9) { background-position: 0 -128px; }
    button:nth-of-type(10) { background-position: 0 -144px; }
    button:nth-of-type(11) { background-position: 0 -160px; }
    button:nth-of-type(12) { background-position: 0 -176px; }
    button:nth-of-type(13) { background-position: 0 -192px; }
    button:nth-of-type(14) { background-position: 0 -208px; }
    button:nth-of-type(15) { background-position: 0 -224px; }
}
```

Maybe one day this fix will be obsolete due to the introduction of spritemap in the Dokuwiki.
см. также https://www.dokuwiki.org/devel:toolbar

From:
https://wwoss.ru/ - **worldwide open-source software**

Permanent link:
https://wwoss.ru/doku.php?id=wiki:devel:toolbar

Last update: **2025/01/10 10:52**