



Inhaltsverzeichnis

Modul:Ustring

This module directly imports all functions from the **Vorlage:Luaref** library. Documentation for each function can be found there.

The module takes an indefinite number of arguments. Arguments given as **Vorlage:Para**, **Vorlage:Para**, etc, are read first, and are used as strings. All remaining numerical arguments are coerced to number type if possible, and remain strings if not. If you wish for a numerical (i.e. unnamed or "**Vorlage:Para**", "**Vorlage:Para**") to remain a string, you can simply escape it by inserting `\` at the beginning of the string.

Note that MediaWiki will always trim whitespace from named arguments; to give arguments with surrounding whitespace you must use unnamed parameters starting with `\`.

How to pass inconvenient strings

To pass this...	Write this	Explanation
" 0123 "	Vorlage:Para	To get surrounding whitespace must use unnamed, but must precede with <code>\</code> to indicate that it isn't a number.
"0123"	Vorlage:Para	If you don't need to preserve whitespace use Vorlage:Para etc.
"0123"	Vorlage:Para	If you don't need to preserve whitespace explicitly use Vorlage:Para etc.
In another template, to pass its parameter <code>{{{1}}}</code> , preserving whitespace	Vorlage:Para	Must provide the <code>\</code> with unknown string input.
In another template, to pass its parameter <code>{{{1}}}</code> , stripping whitespace	Vorlage:Para	
In another template, to pass its parameter <code>{{{1}}}</code> , stripping whitespace	Vorlage:Para	

You can also wrap results in tags. All other unused arguments will be passed to **Vorlage:Luaref**

Inhaltsverzeichnis	
1 Usage	3
1.1 Example using <code>mw.ustring.sub</code>	3
1.2 Example using <code>mw.ustring.gsub</code>	3



1.3 Example using mw.ustring.char	3
1.4 Example using mw.ustring.match	3
1.5 Example using tag arguments	4
2 Errors	4
3 See also	5

Usage

`{{#invoke:Ustring|function_name|arg1|arg2|...}}` is equivalent to [Vorlage:Luaref](#)

Example using mw.ustring.sub

```
{{#invoke:Ustring|sub|s1=abcde|2|4}}
```

produces:

bcd

Example using mw.ustring.gsub

```
{{#invoke:Ustring|gsub|s1=1234|23|}}
```

produces:

14

Example using mw.ustring.char

```
&#amp;#{{#invoke:ustring|char|49|48|59}}
```

produces:

Example using mw.ustring.match

```
{{#invoke:Ustring|match|s1=abcde|s2=(c%w)}}
```

produces:

cd

Note: Only the first match is returned. Additional returns are omitted because mw.ustring.gsub's second return value is generally undesirable.



Example using tag arguments

```

{{#invoke:Ustring|match
|s1={{Module:Ustring}}|%s%s%sif%not%[^%s]+%sthen.+%
<!--enter an actual newline character to match '\n'-->%s%s%send
|tag=syntaxhighlight|lang=lua}}

```

produces:

```

if not fargs.tag then
    return (what(unpack(args))) -- Out
end
local tagargs = {}
for x, y in pairs(fargs) do
    if not fargsused[x] then tagargs[x] = y end
end

```

Note that:

```

<syntaxhighlight lang="lua">{{#invoke:Ustring|match
|s1={{Module:Ustring}}|%s%s%sif%not%[^%s]+%sthen.+%
<!--enter an actual newline character to match '\n'-->%s%s%send}}</syntaxhighlight

```

produces:

```

{{#invoke:Ustring|match
|s1={{Module:Ustring}}|%s%s%sif%not%[^%s]+%sthen.+%
<!--enter an actual newline character to match '\n'-->%s%s%send}}

```

Errors

Errors from accessing [Vorlage:Luaref](#) should be maintained, e.g.:

```

{{#invoke:Ustring|xyzyzy}}

```

should produce:

[Vorlage:Script error](#)

and

```

{{#invoke:Ustring|maxPatternLength}}

```

should produce:

[Vorlage:Script error](#)

See also

Vorlage:String handling templates

```
require('Module:No globals')
return setmetatable({}, {
    __index = function(t, k)
        local what = mw.usttring[k]
        if type(what) ~= "function" then
            return what
        end
        return function(frame)
            local fargs = frame.args
            local fargsused = { tag = true }
            local args = {}
            local str_i = 1
            while fargs['s' .. str_i] do
                fargsused['s' .. str_i] = true
                args[str_i] = fargs['s' .. str_i]
                str_i = str_i + 1
            end
            for i, v in ipairs(fargs) do
                fargsused[i] = true
                args[i + str_i - 1] = tonumber(v) or v:gsub("^\\")
            end
            if not fargs.tag then
                return (what(unpack(args))) -- Out
            end
            local tagargs = {}
            for x, y in pairs(fargs) do
                if not fargsused[x] then tagargs[x] = y end
            end
            return frame:extensionTag{name = fargs.tag, content = wha
        end
    end
})
```