



# Inhaltsverzeichnis

---

1. Modul:Demo/Doku .....	2
2. Modul:Automarkup .....	4
3. Modul:Demo .....	5

# Modul:Demo/Doku

Dies ist die Dokumentationsseite für **Modul:Demo**

Inhaltsverzeichnis	
1 Usage .....	2
1.1 Usage via templates .....	2
1.2 Usage in a module .....	2
2 See also .....	3

## Usage

### Usage via templates

This module supports **Vorlage:T**

```
{{#invoke:Demo|main}}
```

and **Vorlage:T**

```
{{#invoke:Demo|inline}}
```

The input must be wrapped in **Vorlage:Tag** tags or else it may be processed before the module can read it.

### Usage in a module

If you want to use this in another module (such as to make the output prettier), you can get values like so:

```
require('Module:demo').get(frame)
```

Function `get()` returns a table containing:

- `source` = the source code (without **Vorlage:Tag** wrappers, characters substituted with html entities)
- `output` = the execution result of the source.
- `frame` = the frame from which this template took the parameter.

By default, `get()` takes the first parameter of frame. If the frame uses a different parameter name for the nowiki-wrapped source, then place that name (as a string) as the second parameter, like so `require('Module:demo').get(frame, 'alternate_name')`



### Example:

```
p = {}  
  
function p.main(frame)  
  local parts = require('Module:demo').get(frame)  
  return '<Pretty html><pre>' .. parts.source .. '</pre><more pretty html>' .. pa  
end  
  
return p
```

### See also

---

- [Template:Nowiki template demo](#) which uses [Module:Template test case](#)
- [Template:Automarkup](#) which uses [Module:Automarkup](#)



## Modul:Automarkup

---

Implements [Vorlage:TI](#).

### See also

---

- [Template:Demo](#) which uses [Module:Demo](#)
- [Template:Nowiki template demo](#) which uses [Module:Template test case](#)

```
local p = {}
function p.main(frame)
    local args = frame:getParent().args

    local templateArgs = { }
    for key, value in pairs(args) do
        if type(key) == "number" then
            templateArgs[2 * key - 1] = value
            templateArgs[2 * key] = frame:preprocess(value)
        else
            templateArgs[key] = value
        end
    end

    return frame:expandTemplate{ title = "Markup", args = templateArgs }
end
return p
```

# Modul:Demo

## Inhaltsverzeichnis

1 Usage .....	5
1.1 Usage via templates .....	5
1.2 Usage in a module .....	5
2 See also .....	6

## Usage

### Usage via templates

This module supports **Vorlage:T**

```
{{#invoke:Demo|main}}
```

and **Vorlage:T**

```
{{#invoke:Demo|inline}}
```

The input must be wrapped in **Vorlage:Tag** tags or else it may be processed before the module can read it.

### Usage in a module

If you want to use this in another module (such as to make the output prettier), you can get values like so:

```
require('Module:demo').get(frame)
```

Function `get()` returns a table containing:

- `source` = the source code (without **Vorlage:Tag** wrappers, characters substituted with html entities)
- `output` = the execution result of the source.
- `frame` = the frame from which this template took the parameter.

By default, `get()` takes the first parameter of `frame`. If the frame uses a different parameter name for the nowiki-wrapped source, then place that name (as a string) as the second parameter, like so `require('Module:demo').get(frame, 'alternate_name')`

Example:

```
p = {}

function p.main(frame)
  local parts = require('Module:demo').get(frame)
  return '<Pretty html><pre>' .. parts.source .. '</pre><more pretty html>' .. pa
end

return p
```

## See also

- [Template:Nowiki template demo](#) which uses [Module:Template test case](#)
- [Template:Automarkup](#) which uses [Module:Automarkup](#)

```
local p = {}

--creates a frame object that cannot access any of the parent's args
--unless a table containing a list keys of not to inherit is provided
function disinherit(frame, onlyTheseKeys)
  local parent = frame:getParent() or frame
  local orphan = parent:newChild{}
  orphan.getParent = parent.getParent --returns nil
  orphan.args = {}
  if onlyTheseKeys then
    local family = {parent, frame}
    for f = 1, 2 do
      for k, v in pairs(family[f] and family[f].args or {}) do
        orphan.args[k] = orphan.args[k] or v
      end
    end
    parent.args = mw.clone(orphan.args)
    setmetatable(orphan.args, nil)
    for _, k in ipairs(onlyTheseKeys) do
      rawset(orphan.args, k, nil)
    end
  end
  return orphan, parent
end

function p.get(frame, arg, passArgs)
  local orphan, frame = disinherit(frame, passArgs and {arg or 1})
  local code, noWiki, preserve = frame.args[arg or 1] or ''
  if code:match'nowiki' then
    local placeholder, preserve = ('6'):char(), {}
    code = mw.text.unstripNoWiki(code)
    noWiki = code:gsub('%%', placeholder):gsub('&lt;', '<'):gsub('&gt;', '>')
    for k in noWiki:gmatch('&.-;') do
      if not preserve[k] then
        preserve[k] = true
        table.insert(preserve, (k:gsub('&', '&')))
        noWiki = noWiki:gsub('&.-;', '%%s')
      end
    end
    noWiki = mw.text.nowiki(noWiki):format(unpack(preserve)):gsub(pla
  end
  local kill_categories = frame.args.demo_kill_categories or frame.args.no
  return {
    source = noWiki or code,
```



```
        output = orphan:preprocess(code):gsub(kill_categories and '%[Ca
        frame = frame
    }
end

function p.main(frame, demoTable)
    local show = demoTable or p.get(frame)
    local args = show.frame.args
    args.br = tonumber(args.br or 1) and ('<br>'):rep(args.br or 1) or args.br
    if show[args.result_arg] then
        return show[args.result_arg]
    end
    return string.format('<pre>%s</pre>%s', args.style and string.format(
end

-- Alternate function to return an inline result
function p.inline(frame, demoTable)
    local show = demoTable or p.get(frame)
    local args = show.frame.args
    if show[args.result_arg] then
        return show[args.result_arg]
    end
    return string.format('<code>%s</code>%s', args.style and string.forma
end

--passing of args into other module without preprocessing
function p.module(frame)
    local orphan, frame = disinherit(frame, {
        'demo_template',
        'demo_module',
        'demo_module_func',
        'demo_main',
        'demo_br',
        'demo_result_arg',
        'demo_kill_categories',
        'nocat'
    })
    local template = frame.args.demo_template and 'Template:..'frame.args.dem
    local demoFunc = frame.args.demo_module_func or 'main\n'
    local demoModule = require('Module:..'frame.args.demo_module)[demoFunc
    frame.args.br, frame.args.result_arg = frame.args.demo_br, frame.args.dem
    local kill_categories = frame.args.demo_kill_categories or frame.args.noc
    if demoModule then
        local named = {insert = function(self, ...) table.insert(self, .
        local source = {insert = named.insert, '{{', frame.args.demo_tem
        if not template then
            source:insert(2, '#invoke:'):insert(4, '|'):insert(5, dem
        end
        local insertNamed = #source + 1
        for k, v in pairs(orphan.args) do
            local nan, insert = type(k) ~= 'number', {v}
            local target = nan and named or source
            target:insert'|'
            if nan then
                target:insert(k):insert'=':insert'\n'
                table.insert(insert, 1, #target)
            end
            target:insert(unpack(insert))
            local nowiki = v:match('nowiki')
            if nowiki or v:match('{{.-}}') then
                orphan.args[k] = frame:preprocess(nowiki and mw.t
            end
        end
        end
        source:insert'}}'
```



```
        table.insert(source, insertNamed, table.concat(named))
    return p.main(orphan, {
        source = mw.text.encode(table.concat(source), "<>'|=~"),
        output = tostring(demoModule(orphan)):gsub(kill_categorie
        frame = frame
    })
else
    return "ERROR: Invalid module function: "..demoFunc
end
end
return p
```