Inhaltsverzeichnis

1. Modul:Category handler/Doku 2
2. Modul:Category handler 12
3. Modul:Namespace detect 26
4. Modul:Yesno 32

Modul:Category handler/Doku

Dies ist die Dokumentationsseite für Modul:Category handler

This module implements the Vorlage:Tl template. The category handler template helps other templates to automate both categorization and category suppression. For information about using the category handler template in other templates, please see the **template documentation**. Keep reading for information about using the category handler module in other Lua modules, or for information on exporting this module to other wikis.

Inhaltsverzeichnis

1 Use from other Lua modules 2
1.1 When not to use this module 2
1.2 Namespaces 3
1.3 Basic usage 4
1.4 Advanced usage 5
1.5 Subpages 7
1.6 Blacklist 7
1.7 The "nocat" parameter 7
1.8 The "categories" parameter 8
1.9 The "category2" parameter 8
1.10 Categories and text 8
1.11 The "page" parameter 9
1.12 Parameters 10
2 Exporting to other wikis 10
3 See also 10

Use from other Lua modules

When not to use this module

For cases where a module only needs to categorise in one of the namespaces main (articles), file (images) or category, then using this module is overkill. Instead, you can simply get a title object using mw.title.getCurrentTitle and check the `nsText` field. For example:

```
local title = mw.title.getCurrentTitle()
if title.nsText == 'File' then
    -- do something
end
```

However, if your module needs to categorize in any other namespace, then we recommend you use this module, since it provides proper category suppression and makes it easy to select how to categorize in the different namespaces.

Namespaces

This module detects and groups all the different namespaces used on Wikipedia into several types. These types are used as parameter names in this module.

main = Main/article space, as in normal Wikipedia articles.
talk = Any talk space, such as page names that start with "Talk:", "User talk:", "File talk:" and so on.
user, wikipedia, file ... = The other namespaces except the talk pages. Namespace aliases are also accepted. See the table below for the full list.
other = Any namespaces that were not specified as a parameter to the template. See examples below.

List of possible namespace parameters

(excluding *talk* and *other*)

Namespace	Aliases
main	
benutzer	user, benutzerin
projekt	project
datei	file, bild, image
mediawiki	
vorlage	template
hilfe	help
kategorie	category
attribut	property
formular	form
konzept	concept
smw/schema	smw/schema
rule	
widget	
campaign	
timedtext	
modul	module
blog	
socialentity	
gadget	
gadget-definition	gadget definition
copikiblog	

Namespace	Aliases
deutsch	
english	
français	
italiano	
español	
nederlands	
slovenská	
dansk	
polska	
suomi	

Basic usage

This module takes two or more parameters. Here's an example using a hello world program:

```
p = {}
local categoryHandler = require( 'Module:Category handler' ).main

function p.main( frame )
    local result = 'Hello world!'
    local category = categoryHandler{
        '[[Category:Somecat]]',
        nocat = frame.args.nocat -- So "nocat=true/false" works
    }
    category = category or '' -- Check that we don't have a nil value for the cat
    return result .. category
end

return p
```

The above example uses the default settings for the category handler module. That means the example module will categorize on pages in the following namespaces:

main, **file**, **help**, **category**, **portal** and **book**

But it will *not* categorize in any other namespaces, e.g.:

talk, **user**, **wikipedia**, **mediawiki**, **template** ...

And it will *not* categorize on blacklisted pages. (See section blacklist below.)

The reason the category handler module does not categorize in some of the namespaces is that in those namespaces most modules and templates are just demonstrated or listed, not used. Thus most modules and templates should not categorize in those namespaces.

Any module or template that is meant for one or more of the namespaces where this module categorizes can use the basic syntax as shown above.

Advanced usage

This module takes one or more parameters named after the different page types as listed in section namespaces above. By using those parameters you can specify exactly in which namespaces your template should categorize. Like this:

```
p = {}
local categoryHandler = require( 'Module:Category handler' ).main

function p.main( frame )
    local result = 'This is a module meant for articles and talk pages.'
    local category = categoryHandler{
        main = '[[Category:Somecat1]]', -- Categorize in main (article) space
        talk = '[[Category:Somecat2]]', -- Categorize in talk space
        nocat = frame.args.nocat -- So "nocat=true/false" works
    }
    category = category or '' -- Check that we don't have a nil value for the cat
    return result .. category
end

return p
```

The above module will only categorize in main and talk space. But it will not categorize on /archive pages since they are blacklisted. (See section blacklist below.) And if you need to demonstrate (discuss) the module on a talkpage, then you can feed `"nocat='true'"` to prevent that template from categorizing. (See section nocat below.) Like this:

```
== My new module ==
Hey guys, have you seen my new module?
{{#invoke:mymodule|main|nocat=true}}
Nice, isn't it?
--~~~~
```

Sometimes we want to use the same category in several namespaces, then do like this:

```
p = {}
local categoryHandler = require( 'Module:Category handler' ).main

function p.main( frame )
    local result = 'This is a module used in several namespaces.'
    local category = categoryHandler{
        main = '[[Category:Somecat1]]',
        [ 1 ] = '[[Category:Somecat2]]', -- For help and user space
        help = 1,
        user = 1,
        talk = '', -- No categories on talk pages
        other = '[[Category:Somecat3]]', -- For all other namespaces
        nocat = frame.args.nocat -- So "nocat=true/false" works
    }
    category = category or '' -- Check that we don't have a nil value for the cat
    return result .. category
end

return p
```

In the above example we use a numbered parameter to feed one of the categories, and then we tell this module to use that numbered parameter for both the help and user space.

The category handler module understands an unlimited number of numbered parameters.

The **other** parameter defines what should be used in the remaining namespaces that have not explicitly been fed data.

Note the empty but defined **talk** parameter. That stops this module from showing what has been fed to the **other** parameter, when in talk space.

The category handler module also has a parameter called **all**. It works like this:

```
p = {}
local categoryHandler = require( 'Module:Category handler' ).main

function p.main( frame )
    local result = 'This is a module used in all namespaces.'
    local category = categoryHandler{
        all = '[[Category:Somecat1]]', -- Categorize in all namespaces
        nocat = frame.args.nocat -- So "nocat=true/false" works
    }
    category = category or '' -- Check that we don't have a nil value for the cat
    return result .. category
end

return p
```

The above example will categorize in all namespaces, but not on blacklisted pages. If you want to demonstrate that module on a page, then use "`nocat=true`" to prevent the template from categorizing.

We suggest avoiding the **all** parameter, since modules and templates should preferably only categorize in the namespaces they need to.

The all parameter can also be combined with the rest of the parameters. Like this:

```
p = {}
local categoryHandler = require( 'Module:Category handler' ).main

function p.main( frame )
    local result = 'This is a module used in all namespaces.'
    local category = categoryHandler{
        all = '[[Category:Somecat1]]', -- Categorize in all namespaces
        main = '[[Category:Somecat2]]', -- And add this in main space
        other = '[[Category:Somecat3]]', -- And add this in all other namespaces
        nocat = frame.args.nocat -- So "nocat=true/false" works
    }
    category = category or '' -- Check that we don't have a nil value for the cat
    return result .. category
end

return p
```

If the above module is placed on an article, then it will add the categories "Somecat1" and "Somecat2". But on all other types of pages it will instead add "Somecat1" and "Somecat3". As the example shows, the all parameter works independently of the rest of the parameters.

Subpages

The category handler module understands the **subpage** parameter. Like this:

```
p = {}
local categoryHandler = require( 'Module:Category handler' ).main

function p.main( frame )
    local result = 'This is a module used in all namespaces.'
    local category = categoryHandler{
        subpage = 'no' -- Don't categorize on subpages
        wikipedia = '[[Category:Somecat]]',
        nocat = frame.args.nocat -- So "nocat=true/false" works
    }
    category = category or '' -- Check that we don't have a nil value for the cat
    return result .. category
end

return p
```

If "`subpage='no'`" then this template will *not* categorize on subpages. For the rare occasion you *only* want to categorize on subpages, then use "`subpage='only'`". If **subpage** is empty or undefined then this template categorizes both on basepages and on subpages.

Blacklist

This module has a blacklist of the pages and page types where templates should not auto-categorize. Thus modules that use this meta-template will for instance not categorize on /archive pages and on the subpages of Wikipedia:Template messages.

If you want a template to categorize on a blacklisted page, then feed "`nocat = false`" to the module when you place it on the page, thus skipping the blacklist check. Note that this module only categorizes if it has data for the namespace. For instance, if the basic syntax is used (see basic usage above), then even if you set "`nocat = false`" the template will not categorize on a talk page, since it has no data for talk pages. But it has data for help space, so on a blacklisted help page it will categorize.

The blacklist is located in the configuration table `cfg.blacklist` near the top of the module code.

The "nocat" parameter

This module understands the **nocat** parameter:

- If "`nocat = true`" then this template does *not* categorize.
- If **nocat** is `nil` then this template categorizes as usual.

- If "`nocat = false`" this template categorizes even when on blacklisted pages. (See section blacklist above.)
- The nocat parameter also accepts aliases for `true` and `false` as defined by Module:Yesno, e.g. "yes", "y", "true", and 1 for `true`, and "no", "n", "false", and 0 for `false`.

Modules and templates that use Vorlage:Tlf should forward **nocat**, so they too understand **nocat**. The code "`nocat = frame.args.nocat`" shown in the examples on this page does that.

The "categories" parameter

For backwards compatibility this module also understands the **categories** parameter. It works the same as **nocat**. Like this:

- If "`categories = false`" then this template does *not* categorize.
- If **categories** is empty or undefined then this template categorizes as usual.
- If "`categories = true`" this template categorizes even when on blacklisted pages.
- The categories parameter also accepts aliases for `true` and `false` as defined by Module:Yesno , e.g. "yes", "y", "true", and 1 for `true`, and "no", "n", "false", and 0 for `false`.

The "category2" parameter

For backwards compatibility this template kind of supports the old "category =" parameter. But the parameter name "category" is already used in this module to feed category data for when in category space. So instead this template uses **category2** for the usage similar to **nocat**. Like this:

- If "`category2 = ""`" (empty but defined), or "`category2 = 'no'`", or if **category2** is fed any other data (except as described in the next two points), then this template does *not* categorize.
- If **category2** is undefined or if "`category2 = '¬'`", then this template categorizes as usual.
- If "`category2 = 'yes'`" this template categorizes even when on blacklisted pages.

Categories and text

Besides from categories, you can feed anything else to this module, for instance some text. Like this:

```
p = {}
local categoryHandler = require( 'Module:Category handler' ).main

function p.main( frame )
    local result = 'This is a module used on talk pages.'
    local category = categoryHandler{
        talk = '[[Category:Somecat]]',
        other = '<p class="error">This module should only be used on talk pages.<
        nocat = frame.args.nocat -- So "nocat=true/false" works
```

```
    }
    category = category or '' -- Check that we don't have a nil value for the cat
    return result .. category
end

return p
```

When the module code above is used on anything other than a talk page, it will look like this:

> This is a module used on talk pages.**Lua-Fehler in package.lua, Zeile 80: module 'Module:Category handler/data' not found**

That text will not show on blacklisted pages, so don't use this method to show any important information. Feeding "`nocat = 'true'`" to the template hides the text, just as it suppresses any categories.

The "page" parameter

For testing and demonstration purposes this module can take a parameter named **page**. Like this:

```
p = {}
local categoryHandler = require( 'Module:Category handler' ).main

function p.main( frame )
    local category = categoryHandler{
        main = 'Category:Some cat',
        talk = 'Category:Talk cat',
        nocat = frame.args.nocat, -- So "nocat=true/false" works
        page = 'User talk:Example'
    }
    return category
end

return p
```

In the above code we on purpose left out the brackets around the category names so we see the output on the page. No matter on what kind of page the code above is used it will return this:

> **Lua-Fehler in package.lua, Zeile 80: module 'Module:Category handler/data' not found**

The **page** parameter makes this module behave exactly as if on that page. Even the blacklist works. The pagename doesn't have to be an existing page.

If the **page** parameter is empty or undefined, the name of the current page determines the result.

You can make it so your module also understands the **page** parameter. That means you can test how your template will categorize on different pages, without having to actually edit those pages. Then do like this:

```
p = {}
local categoryHandler = require( 'Module:Category handler' ).main

function p.main( frame )
    local category = categoryHandler{
        main = 'Category:Some cat',
        talk = 'Category:Talk cat',
        nocat = frame.args.nocat, -- So "nocat=true/false" works
        page = frame.args.page -- For testing
    }
    return category
end

return p
```

Parameters

List of all parameters:

- First positional parameter - for default settings
- subpage = 'no' / 'only'
- 1, 2, 3 ...
- all = '[[Category:Somecat]]' / 'Text'
- main = 1, 2, 3 ... / '[[Category:Somecat]]' / 'Text'
- ...
- other = 1, 2, 3 ... / '[[Category:Somecat]]' / 'Text'
- nocat = frame.args.nocat / true / false / 'yes' / 'no' / 'y' / 'n' / 'true' / 'false' / 1 / 0
- categories = frame.args.categories / false / true / 'no' / 'yes' / 'n' / 'y' / 'false' / 'true' / 0 / 1
- category2 = frame.args.category or '¬' / 'no' / 'not defined' / '¬' / 'yes'
- page = frame.args.page / 'User:Example'

Note that empty values to the "main" ... "other" parameters have special meaning (see examples above). The "all" parameter doesn't understand numbered parameters, since there should never be a need for that.

Exporting to other wikis

This module can be exported to other wikis by changing the configuration values in the `cfg` table. All the variable values are configurable, so after the configuration values have been set there should be no need to alter the main module code. Details of each configuration value are included in the module code comments. In addition, this module requires Module:Namespace detect to be available on the local wiki.

See also

- Vorlage:Tl – for using this module with templates, rather than Lua modules.
- Wikipedia:Category suppression – The how-to guide.
- Wikipedia:WikiProject Category Suppression – The WikiProject.

- Wikipedia:Namespace – Lists all the namespaces.

Modul:Category handler

This module implements the Vorlage:Tl template. The category handler template helps other templates to automate both categorization and category suppression. For information about using the category handler template in other templates, please see the **template documentation**. Keep reading for information about using the category handler module in other Lua modules, or for information on exporting this module to other wikis.

Inhaltsverzeichnis

1 Use from other Lua modules 12
1.1 When not to use this module 12
1.2 Namespaces 13
1.3 Basic usage 14
1.4 Advanced usage 15
1.5 Subpages 17
1.6 Blacklist 17
1.7 The "nocat" parameter 17
1.8 The "categories" parameter 18
1.9 The "category2" parameter 18
1.10 Categories and text 18
1.11 The "page" parameter 19
1.12 Parameters 20
2 Exporting to other wikis 20
3 See also 20

Use from other Lua modules

When not to use this module

For cases where a module only needs to categorise in one of the namespaces main (articles), file (images) or category, then using this module is overkill. Instead, you can simply get a title object using mw.title.getCurrentTitle and check the `nsText` field. For example:

```
local title = mw.title.getCurrentTitle()
if title.nsText == 'File' then
    -- do something
end
```

However, if your module needs to categorize in any other namespace, then we recommend you use this module, since it provides proper category suppression and makes it easy to select how to categorize in the different namespaces.

Namespaces

This module detects and groups all the different namespaces used on Wikipedia into several types. These types are used as parameter names in this module.

main = Main/article space, as in normal Wikipedia articles.
talk = Any talk space, such as page names that start with "Talk:", "User talk:", "File talk:" and so on.
user, wikipedia, file ... = The other namespaces except the talk pages. Namespace aliases are also accepted. See the table below for the full list.
other = Any namespaces that were not specified as a parameter to the template. See examples below.

List of possible namespace parameters

(excluding *talk* and *other*)

Namespace	Aliases
main	
benutzer	user, benutzerin
projekt	project
datei	file, bild, image
mediawiki	
vorlage	template
hilfe	help
kategorie	category
attribut	property
formular	form
konzept	concept
smw/schema	smw/schema
rule	
widget	
campaign	
timedtext	
modul	module
blog	
socialentity	
gadget	
gadget-definition	gadget definition
copikiblog	

Namespace	Aliases
deutsch	
english	
français	
italiano	
español	
nederlands	
slovenská	
dansk	
polska	
suomi	

Basic usage

This module takes two or more parameters. Here's an example using a hello world program:

```
p = {}
local categoryHandler = require( 'Module:Category handler' ).main

function p.main( frame )
    local result = 'Hello world!'
    local category = categoryHandler{
        '[[Category:Somecat]]',
        nocat = frame.args.nocat -- So "nocat=true/false" works
    }
    category = category or '' -- Check that we don't have a nil value for the cat
    return result .. category
end

return p
```

The above example uses the default settings for the category handler module. That means the example module will categorize on pages in the following namespaces:

main, **file**, **help**, **category**, **portal** and **book**

But it will *not* categorize in any other namespaces, e.g.:

talk, **user**, **wikipedia**, **mediawiki**, **template** ...

And it will *not* categorize on blacklisted pages. (See section blacklist below.)

The reason the category handler module does not categorize in some of the namespaces is that in those namespaces most modules and templates are just demonstrated or listed, not used. Thus most modules and templates should not categorize in those namespaces.

Any module or template that is meant for one or more of the namespaces where this module categorizes can use the basic syntax as shown above.

Advanced usage

This module takes one or more parameters named after the different page types as listed in section namespaces above. By using those parameters you can specify exactly in which namespaces your template should categorize. Like this:

```
p = {}
local categoryHandler = require( 'Module:Category handler' ).main

function p.main( frame )
    local result = 'This is a module meant for articles and talk pages.'
    local category = categoryHandler{
        main = '[[Category:Somecat1]]', -- Categorize in main (article) space
        talk = '[[Category:Somecat2]]', -- Categorize in talk space
        nocat = frame.args.nocat -- So "nocat=true/false" works
    }
    category = category or '' -- Check that we don't have a nil value for the cat
    return result .. category
end

return p
```

The above module will only categorize in main and talk space. But it will not categorize on /archive pages since they are blacklisted. (See section blacklist below.) And if you need to demonstrate (discuss) the module on a talkpage, then you can feed `"nocat='true'"` to prevent that template from categorizing. (See section nocat below.) Like this:

```
== My new module ==
Hey guys, have you seen my new module?
{{#invoke:mymodule|main|nocat=true}}
Nice, isn't it?
--~~~~
```

Sometimes we want to use the same category in several namespaces, then do like this:

```
p = {}
local categoryHandler = require( 'Module:Category handler' ).main

function p.main( frame )
    local result = 'This is a module used in several namespaces.'
    local category = categoryHandler{
        main = '[[Category:Somecat1]]',
        [ 1 ] = '[[Category:Somecat2]]', -- For help and user space
        help = 1,
        user = 1,
        talk = '', -- No categories on talk pages
        other = '[[Category:Somecat3]]', -- For all other namespaces
        nocat = frame.args.nocat -- So "nocat=true/false" works
    }
    category = category or '' -- Check that we don't have a nil value for the cat
    return result .. category
end

return p
```

In the above example we use a numbered parameter to feed one of the categories, and then we tell this module to use that numbered parameter for both the help and user space.

The category handler module understands an unlimited number of numbered parameters.

The **other** parameter defines what should be used in the remaining namespaces that have not explicitly been fed data.

Note the empty but defined **talk** parameter. That stops this module from showing what has been fed to the **other** parameter, when in talk space.

The category handler module also has a parameter called **all**. It works like this:

```
p = {}
local categoryHandler = require( 'Module:Category handler' ).main

function p.main( frame )
    local result = 'This is a module used in all namespaces.'
    local category = categoryHandler{
        all = '[[Category:Somecat1]]', -- Categorize in all namespaces
        nocat = frame.args.nocat -- So "nocat=true/false" works
    }
    category = category or '' -- Check that we don't have a nil value for the cat
    return result .. category
end

return p
```

The above example will categorize in all namespaces, but not on blacklisted pages. If you want to demonstrate that module on a page, then use "`nocat=true`" to prevent the template from categorizing.

We suggest avoiding the **all** parameter, since modules and templates should preferably only categorize in the namespaces they need to.

The all parameter can also be combined with the rest of the parameters. Like this:

```
p = {}
local categoryHandler = require( 'Module:Category handler' ).main

function p.main( frame )
    local result = 'This is a module used in all namespaces.'
    local category = categoryHandler{
        all = '[[Category:Somecat1]]', -- Categorize in all namespaces
        main = '[[Category:Somecat2]]', -- And add this in main space
        other = '[[Category:Somecat3]]', -- And add this in all other namespaces
        nocat = frame.args.nocat -- So "nocat=true/false" works
    }
    category = category or '' -- Check that we don't have a nil value for the cat
    return result .. category
end

return p
```

If the above module is placed on an article, then it will add the categories "Somecat1" and "Somecat2". But on all other types of pages it will instead add "Somecat1" and "Somecat3". As the example shows, the all parameter works independently of the rest of the parameters.

Subpages

The category handler module understands the **subpage** parameter. Like this:

```
p = {}
local categoryHandler = require( 'Module:Category handler' ).main

function p.main( frame )
    local result = 'This is a module used in all namespaces.'
    local category = categoryHandler{
        subpage = 'no' -- Don't categorize on subpages
        wikipedia = '[[Category:Somecat]]',
        nocat = frame.args.nocat -- So "nocat=true/false" works
    }
    category = category or '' -- Check that we don't have a nil value for the cat
    return result .. category
end

return p
```

If "`subpage='no'`" then this template will *not* categorize on subpages. For the rare occasion you *only* want to categorize on subpages, then use "`subpage='only'`". If **subpage** is empty or undefined then this template categorizes both on basepages and on subpages.

Blacklist

This module has a blacklist of the pages and page types where templates should not auto-categorize. Thus modules that use this meta-template will for instance not categorize on /archive pages and on the subpages of Wikipedia:Template messages.

If you want a template to categorize on a blacklisted page, then feed "`nocat = false`" to the module when you place it on the page, thus skipping the blacklist check. Note that this module only categorizes if it has data for the namespace. For instance, if the basic syntax is used (see basic usage above), then even if you set "`nocat = false`" the template will not categorize on a talk page, since it has no data for talk pages. But it has data for help space, so on a blacklisted help page it will categorize.

The blacklist is located in the configuration table `cfg.blacklist` near the top of the module code.

The "nocat" parameter

This module understands the **nocat** parameter:

- If "`nocat = true`" then this template does *not* categorize.
- If **nocat** is `nil` then this template categorizes as usual.

- If "`nocat = false`" this template categorizes even when on blacklisted pages. (See section blacklist above.)
- The nocat parameter also accepts aliases for `true` and `false` as defined by Module:Yesno, e.g. "yes", "y", "true", and 1 for `true`, and "no", "n", "false", and 0 for `false`.

Modules and templates that use Vorlage:Tlf should forward **nocat**, so they too understand **nocat**. The code "`nocat = frame.args.nocat`" shown in the examples on this page does that.

The "categories" parameter

For backwards compatibility this module also understands the **categories** parameter. It works the same as **nocat**. Like this:

- If "`categories = false`" then this template does *not* categorize.
- If **categories** is empty or undefined then this template categorizes as usual.
- If "`categories = true`" this template categorizes even when on blacklisted pages.
- The categories parameter also accepts aliases for `true` and `false` as defined by Module:Yesno , e.g. "yes", "y", "true", and 1 for `true`, and "no", "n", "false", and 0 for `false`.

The "category2" parameter

For backwards compatibility this template kind of supports the old "category =" parameter. But the parameter name "category" is already used in this module to feed category data for when in category space. So instead this template uses **category2** for the usage similar to **nocat**. Like this:

- If "`category2 = ""`" (empty but defined), or "`category2 = 'no'`", or if **category2** is fed any other data (except as described in the next two points), then this template does *not* categorize.
- If **category2** is undefined or if "`category2 = '¬'`", then this template categorizes as usual.
- If "`category2 = 'yes'`" this template categorizes even when on blacklisted pages.

Categories and text

Besides from categories, you can feed anything else to this module, for instance some text. Like this:

```
p = {}
local categoryHandler = require( 'Module:Category handler' ).main

function p.main( frame )
    local result = 'This is a module used on talk pages.'
    local category = categoryHandler{
        talk = '[[Category:Somecat]]',
        other = '<p class="error">This module should only be used on talk pages.<
        nocat = frame.args.nocat -- So "nocat=true/false" works
```

```
    }
    category = category or '' -- Check that we don't have a nil value for the cat
    return result .. category
end

return p
```

When the module code above is used on anything other than a talk page, it will look like this:

> This is a module used on talk pages.**Lua-Fehler in package.lua, Zeile 80: module 'Module:Category handler/data' not found**

That text will not show on blacklisted pages, so don't use this method to show any important information. Feeding "`nocat = 'true'`" to the template hides the text, just as it suppresses any categories.

The "page" parameter

For testing and demonstration purposes this module can take a parameter named **page**. Like this:

```
p = {}
local categoryHandler = require( 'Module:Category handler' ).main

function p.main( frame )
    local category = categoryHandler{
        main = 'Category:Some cat',
        talk = 'Category:Talk cat',
        nocat = frame.args.nocat, -- So "nocat=true/false" works
        page = 'User talk:Example'
    }
    return category
end

return p
```

In the above code we on purpose left out the brackets around the category names so we see the output on the page. No matter on what kind of page the code above is used it will return this:

> **Lua-Fehler in package.lua, Zeile 80: module 'Module:Category handler/data' not found**

The **page** parameter makes this module behave exactly as if on that page. Even the blacklist works. The pagename doesn't have to be an existing page.

If the **page** parameter is empty or undefined, the name of the current page determines the result.

You can make it so your module also understands the **page** parameter. That means you can test how your template will categorize on different pages, without having to actually edit those pages. Then do like this:

```
p = {}
local categoryHandler = require( 'Module:Category handler' ).main

function p.main( frame )
    local category = categoryHandler{
        main = 'Category:Some cat',
        talk = 'Category:Talk cat',
        nocat = frame.args.nocat, -- So "nocat=true/false" works
        page = frame.args.page -- For testing
    }
    return category
end

return p
```

Parameters

List of all parameters:

- First positional parameter - for default settings
- subpage = 'no' / 'only'
- 1, 2, 3 ...
- all = '[[Category:Somecat]]' / 'Text'
- main = 1, 2, 3 ... / '[[Category:Somecat]]' / 'Text'
- ...
- other = 1, 2, 3 ... / '[[Category:Somecat]]' / 'Text'
- nocat = frame.args.nocat / true / false / 'yes' / 'no' / 'y' / 'n' / 'true' / 'false' / 1 / 0
- categories = frame.args.categories / false / true / 'no' / 'yes' / 'n' / 'y' / 'false' / 'true' / 0 / 1
- category2 = frame.args.category or '¬' / 'no' / 'not defined' / '¬' / 'yes'
- page = frame.args.page / 'User:Example'

Note that empty values to the "main" ... "other" parameters have special meaning (see examples above). The "all" parameter doesn't understand numbered parameters, since there should never be a need for that.

Exporting to other wikis

This module can be exported to other wikis by changing the configuration values in the `cfg` table. All the variable values are configurable, so after the configuration values have been set there should be no need to alter the main module code. Details of each configuration value are included in the module code comments. In addition, this module requires Module:Namespace detect to be available on the local wiki.

See also

- Vorlage:Tl – for using this module with templates, rather than Lua modules.
- Wikipedia:Category suppression – The how-to guide.
- Wikipedia:WikiProject Category Suppression – The WikiProject.

- Wikipedia:Namespace – Lists all the namespaces.

```
--------------------------------------------------------------------------------
--                                                                            --
--                              CATEGORY HANDLER                              --
--                                                                            --
--      This module implements the {{category handler}} template in Lua,      --
--      with a few improvements: all namespaces and all namespace aliases     --
--      are supported, and namespace names are detected automatically for     --
--      the local wiki. This module requires [[Module:Namespace detect]]      --
--      and [[Module:Yesno]] to be available on the local wiki. It can be     --
--      configured for different wikis by altering the values in              --
--      [[Module:Category handler/config]], and pages can be blacklisted      --
--      from categorisation by using [[Module:Category handler/blacklist]].   --
--                                                                            --
--------------------------------------------------------------------------------

-- Load required modules
local yesno = require('Module:Yesno')

-- Lazily load things we don't always need
local mShared, mappings

local p = {}

--------------------------------------------------------------------------------
-- Helper functions
--------------------------------------------------------------------------------

local function trimWhitespace(s, removeBlanks)
        if type(s) ~= 'string' then
                return s
        end
        s = s:match('^%s*(.-)%s*$')
        if removeBlanks then
                if s ~= '' then
                        return s
                else
                        return nil
                end
        else
                return s
        end
end

--------------------------------------------------------------------------------
-- CategoryHandler class
--------------------------------------------------------------------------------

local CategoryHandler = {}
CategoryHandler.__index = CategoryHandler

function CategoryHandler.new(data, args)
        local obj = setmetatable({ _data = data, _args = args }, CategoryHandler)

        -- Set the title object
        do
                local pagename = obj:parameter('demopage')
                local success, titleObj
                if pagename then
                        success, titleObj = pcall(mw.title.new, pagename)
                end
```

```
                if success and titleObj then
                        obj.title = titleObj
                        if titleObj == mw.title.getCurrentTitle() then
                                obj._usesCurrentTitle = true
                        end
                else
                        obj.title = mw.title.getCurrentTitle()
                        obj._usesCurrentTitle = true
                end
        end

        -- Set suppression parameter values
        for _, key in ipairs{'nocat', 'categories'} do
                local value = obj:parameter(key)
                value = trimWhitespace(value, true)
                obj['_' .. key] = yesno(value)
        end
        do
                local subpage = obj:parameter('subpage')
                local category2 = obj:parameter('category2')
                if type(subpage) == 'string' then
                        subpage = mw.ustring.lower(subpage)
                end
                if type(category2) == 'string' then
                        subpage = mw.ustring.lower(category2)
                end
                obj._subpage = trimWhitespace(subpage, true)
                obj._category2 = trimWhitespace(category2) -- don't remove blank
        end
        return obj
end

function CategoryHandler:parameter(key)
        local parameterNames = self._data.parameters[key]
        local pntype = type(parameterNames)
        if pntype == 'string' or pntype == 'number' then
                return self._args[parameterNames]
        elseif pntype == 'table' then
                for _, name in ipairs(parameterNames) do
                        local value = self._args[name]
                        if value ~= nil then
                                return value
                        end
                end
                return nil
        else
                error(string.format(
                        'invalid config key "%s"',
                        tostring(key)
                ), 2)
        end
end

function CategoryHandler:isSuppressedByArguments()
        return
                -- See if a category suppression argument has been set.
                self._nocat == true
                or self._categories == false
                or (
                        self._category2
                        and self._category2 ~= self._data.category2Yes
                        and self._category2 ~= self._data.category2Negative
                )
```

```
                -- Check whether we are on a subpage, and see if categories are
                -- suppressed based on our subpage status.
                or self._subpage == self._data.subpageNo and self.title.isSubpage
                or self._subpage == self._data.subpageOnly and not self.title.isS
end

function CategoryHandler:shouldSkipBlacklistCheck()
        -- Check whether the category suppression arguments indicate we
        -- should skip the blacklist check.
        return self._nocat == false
                or self._categories == true
                or self._category2 == self._data.category2Yes
end

function CategoryHandler:matchesBlacklist()
        if self._usesCurrentTitle then
                return self._data.currentTitleMatchesBlacklist
        else
                mShared = mShared or require('Module:Category handler/shared')
                return mShared.matchesBlacklist(
                        self.title.prefixedText,
                        mw.loadData('Module:Category handler/blacklist')
                )
        end
end

function CategoryHandler:isSuppressed()
        -- Find if categories are suppressed by either the arguments or by
        -- matching the blacklist.
        return self:isSuppressedByArguments()
                or not self:shouldSkipBlacklistCheck() and self:matchesBlacklist(
end

function CategoryHandler:getNamespaceParameters()
        if self._usesCurrentTitle then
                return self._data.currentTitleNamespaceParameters
        else
                if not mappings then
                        mShared = mShared or require('Module:Category handler/sha
                        mappings = mShared.getParamMappings(true) -- gets mapping
                end
                return mShared.getNamespaceParameters(
                        self.title,
                        mappings
                )
        end
end

function CategoryHandler:namespaceParametersExist()
        -- Find whether any namespace parameters have been specified.
        -- We use the order "all" --> namespace params --> "other" as this is wha
        -- the old template did.
        if self:parameter('all') then
                return true
        end
        if not mappings then
                mShared = mShared or require('Module:Category handler/shared')
                mappings = mShared.getParamMappings(true) -- gets mappings with n
        end
        for ns, params in pairs(mappings) do
                for i, param in ipairs(params) do
                        if self._args[param] then
                                return true
                        end
```

```
                end
        end
        if self:parameter('other') then
                return true
        end
        return false
end

function CategoryHandler:getCategories()
        local params = self:getNamespaceParameters()
        local nsCategory
        for i, param in ipairs(params) do
                local value = self._args[param]
                if value ~= nil then
                        nsCategory = value
                        break
                end
        end
        if nsCategory ~= nil or self:namespaceParametersExist() then
                -- Namespace parameters exist - advanced usage.
                if nsCategory == nil then
                        nsCategory = self:parameter('other')
                end
                local ret = {self:parameter('all')}
                local numParam = tonumber(nsCategory)
                if numParam and numParam >= 1 and math.floor(numParam) == numPara
                        -- nsCategory is an integer
                        ret[#ret + 1] = self._args[numParam]
                else
                        ret[#ret + 1] = nsCategory
                end
                if #ret < 1 then
                        return nil
                else
                        return table.concat(ret)
                end
        elseif self._data.defaultNamespaces[self.title.namespace] then
                -- Namespace parameters don't exist, simple usage.
                return self._args[1]
        end
        return nil
end

--------------------------------------------------------------------------------
-- Exports
--------------------------------------------------------------------------------

local p = {}

function p._exportClasses()
        -- Used for testing purposes.
        return {
                CategoryHandler = CategoryHandler
        }
end

function p._main(args, data)
        data = data or mw.loadData('Module:Category handler/data')
        local handler = CategoryHandler.new(data, args)
        if handler:isSuppressed() then
                return nil
        end
        return handler:getCategories()
end
```

```
function p.main(frame, data)
	data = data or mw.loadData('Module:Category handler/data')
	local args = require('Module:Arguments').getArgs(frame, {
		wrappers = data.wrappers,
		valueFunc = function (k, v)
			v = trimWhitespace(v)
			if type(k) == 'number' then
				if v ~= '' then
					return v
				else
					return nil
				end
			else
				return v
			end
		end
	})
	return p._main(args, data)
end

return p
```

Modul:Namespace detect

This module allows you to output different text depending on the namespace that a given page is in. It is a Lua implementation of the Vorlage:Tl template, with a few improvements: all namespaces and all namespace aliases are supported, and namespace names are detected automatically for the local wiki.

Inhaltsverzeichnis

1 Usage 26
2 Parameters 26
2.1 Namespace parameters 27
3 Table function 28
4 Porting to different wikis 28
5 Technical details 28

Usage

```
{{#invoke: Namespace detect | main
| page              = <!-- page to detect namespace for, if not the current page
| main              = <!-- text to return for the main namespace -->
| talk              = <!-- text to return for talk namespaces -->

<!-- text to return for specific subject namespaces -->
| portal            =
| category          =
| user                    =
| wikipedia         =
| mediawiki         =
| book              =
| timedtext         =
| template          =
| special           =
| media             =
| file              =
| image             =
| help                    =
| module            =

| other             = <!-- text to return for unspecified namespaces -->
| demospace         = <!-- namespace to display text for -->

| subjectns         = <!-- set to "yes" to treat talk pages as the corresponding
}}
```

Parameters

- **main** - text to return if the page is in the main namespace.
- **talk** - text to return if the page is in a talk namespace. This can be any talk namespace - it will match any of "Talk:", "Wikipedia talk:", "User talk:", etc.

- Subject namespace parameters, e.g. **wikipedia**, **user**, **file**... - the text to return if the page is in the corresponding namespace. This module accepts all subject namespaces as parameters, including namespace aliases and virtual namespaces. See below for a list of supported values.
- **other** - text to return if no parameters for the page's namespace were specified. This text is also returned if Vorlage:Para is set to an invalid namespace value.
- **subjectns** - if on a talk page, use the corresponding subject page. Can be set with values of "yes", "y", "true" or "1".
- **demopage** - specifies a page to detect the namespace of. If not specified, and if the Vorlage:Para parameter is not set, then the module uses the current page.
- **demospace** - force the module to behave as if the page was in the specified namespace. Often used for demonstrations.

Namespace parameters

Possible values for subject namespace parameters are as follows:

Namespace	Aliases
main	
benutzer	user, benutzerin
projekt	project
datei	file, bild, image
mediawiki	
vorlage	template
hilfe	help
kategorie	category
attribut	property
formular	form
konzept	concept
smw/schema	smw/schema
rule	
widget	
campaign	
timedtext	
modul	module
blog	
socialentity	
gadget	
gadget-definition	gadget definition
copikiblog	

Namespace	Aliases
deutsch	
english	
français	
italiano	
español	
nederlands	
slovenská	
dansk	
polska	
suomi	

Table function

Use the following to display a table with the different possible namespace parameters:

```
{{#invoke:Namespace detect|table|talk=yes}}
```

To include the parameter for talk namespaces, use Vorlage:Para.

Porting to different wikis

This module is designed to be portable. To use it on a different wiki, all you need to do is to change the values in Module:Namespace detect/config. Instructions are available on that page.

Technical details

The module uses a data page at Module:Namespace detect/data. This page is loaded with mw.loadData, which means it is processed once per page rather than once per #invoke. This was done for performance reasons.

```
--[[
--------------------------------------------------------------------------------
--                                                                            --
--                            NAMESPACE DETECT                                --
--                                                                            --
-- This module implements the {{namespace detect}} template in Lua, with a    --
-- few improvements: all namespaces and all namespace aliases are supported,  --
-- and namespace names are detected automatically for the local wiki. The     --
-- module can also use the corresponding subject namespace value if it is     --
-- used on a talk page. Parameter names can be configured for different wikis --
-- by altering the values in the "cfg" table in                               --
-- Module:Namespace detect/config.                                            --
--                                                                            --
--------------------------------------------------------------------------------
--]]
```

```
local data = mw.loadData('Module:Namespace detect/data')
local argKeys = data.argKeys
local cfg = data.cfg
local mappings = data.mappings

local yesno = require('Module:Yesno')
local mArguments -- Lazily initialise Module:Arguments
local mTableTools -- Lazily initilalise Module:TableTools
local ustringLower = mw.ustring.lower

local p = {}

local function fetchValue(t1, t2)
	-- Fetches a value from the table t1 for the first key in array t2 where
	-- a non-nil value of t1 exists.
	for i, key in ipairs(t2) do
		local value = t1[key]
		if value ~= nil then
			return value
		end
	end
	return nil
end

local function equalsArrayValue(t, value)
	-- Returns true if value equals a value in the array t. Otherwise
	-- returns false.
	for i, arrayValue in ipairs(t) do
		if value == arrayValue then
			return true
		end
	end
	return false
end

function p.getPageObject(page)
	-- Get the page object, passing the function through pcall in case of
	-- errors, e.g. being over the expensive function count limit.
	if page then
		local success, pageObject = pcall(mw.title.new, page)
		if success then
			return pageObject
		else
			return nil
		end
	else
		return mw.title.getCurrentTitle()
	end
end

-- Provided for backward compatibility with other modules
function p.getParamMappings()
	return mappings
end

local function getNamespace(args)
	-- This function gets the namespace name from the page object.
	local page = fetchValue(args, argKeys.demopage)
	if page == '' then
		page = nil
	end
	local demospace = fetchValue(args, argKeys.demospace)
	if demospace == '' then
```

```
                demospace = nil
        end
        local subjectns = fetchValue(args, argKeys.subjectns)
        local ret
        if demospace then
                -- Handle "demospace = main" properly.
                if equalsArrayValue(argKeys.main, ustringLower(demospace)) then
                        ret = mw.site.namespaces[0].name
                else
                        ret = demospace
                end
        else
                local pageObject = p.getPageObject(page)
                if pageObject then
                        if pageObject.isTalkPage then
                                -- Get the subject namespace if the option is set
                                -- otherwise use "talk".
                                if yesno(subjectns) then
                                        ret = mw.site.namespaces[pageObject.names
                                else
                                        ret = 'talk'
                                end
                        else
                                ret = pageObject.nsText
                        end
                else
                        return nil -- return nil if the page object doesn't exist
                end
        end
        ret = ret:gsub('_', ' ')
        return ustringLower(ret)
end

function p._main(args)
        -- Check the parameters stored in the mappings table for any matches.
        local namespace = getNamespace(args) or 'other' -- "other" avoids nil tab
        local params = mappings[namespace] or {}
        local ret = fetchValue(args, params)
        --[[
        -- If there were no matches, return parameters for other namespaces.
        -- This happens if there was no text specified for the namespace that
        -- was detected or if the demospace parameter is not a valid
        -- namespace. Note that the parameter for the detected namespace must be
        -- completely absent for this to happen, not merely blank.
        --]]
        if ret == nil then
                ret = fetchValue(args, argKeys.other)
        end
        return ret
end

function p.main(frame)
        mArguments = require('Module:Arguments')
        local args = mArguments.getArgs(frame, {removeBlanks = false})
        local ret = p._main(args)
        return ret or ''
end

function p.table(frame)
        --[[
        -- Create a wikitable of all subject namespace parameters, for
        -- documentation purposes. The talk parameter is optional, in case it
        -- needs to be excluded in the documentation.
        --]]
```

```
        -- Load modules and initialise variables.
        mTableTools = require('Module:TableTools')
        local namespaces = mw.site.namespaces
        local cfg = data.cfg
        local useTalk = type(frame) == 'table'
                and type(frame.args) == 'table'
                and yesno(frame.args.talk) -- Whether to use the talk parameter.

        -- Get the header names.
        local function checkValue(value, default)
                if type(value) == 'string' then
                        return value
                else
                        return default
                end
        end
        local nsHeader = checkValue(cfg.wikitableNamespaceHeader, 'Namespace')
        local aliasesHeader = checkValue(cfg.wikitableAliasesHeader, 'Aliases')

        -- Put the namespaces in order.
        local mappingsOrdered = {}
        for nsname, params in pairs(mappings) do
                if useTalk or nsname ~= 'talk' then
                        local nsid = namespaces[nsname].id
                        -- Add 1, as the array must start with 1; nsid 0 would be
                        nsid = nsid + 1
                        mappingsOrdered[nsid] = params
                end
        end
        mappingsOrdered = mTableTools.compressSparseArray(mappingsOrdered)

        -- Build the table.
        local ret = '{| class="wikitable"'
                .. '\n|-'
                .. '\n! ' .. nsHeader
                .. '\n! ' .. aliasesHeader
        for i, params in ipairs(mappingsOrdered) do
                for j, param in ipairs(params) do
                        if j == 1 then
                                ret = ret .. '\n|-'
                                        .. '\n| <code>' .. param .. '</code>'
                                        .. '\n| '
                        elseif j == 2 then
                                ret = ret .. '<code>' .. param .. '</code>'
                        else
                                ret = ret .. ', <code>' .. param .. '</code>'
                        end
                end
        end
        ret = ret .. '\n|-'
                .. '\n|}'
        return ret
end

return p
```

Modul:Yesno

This module provides a consistent interface for processing boolean or boolean-style string input. While Lua allows the `true` and `false` boolean values, wikicode templates can only express boolean values through strings such as "yes", "no", etc. This module processes these kinds of strings and turns them into boolean input for Lua to process. It also returns `nil` values as `nil`, to allow for distinctions between `nil` and `false`. The module also accepts other Lua structures as input, i.e. booleans, numbers, tables, and functions. If it is passed input that it does not recognise as boolean or `nil`, it is possible to specify a default value to return.

Inhaltsverzeichnis

1 Syntax 32
2 Usage 32
2.1 Undefined input ('foo') 33
2.2 Handling nil results 34

Syntax

```
yesno(value, default)
```

`value` is the value to be tested. Boolean input or boolean-style input (see below) always evaluates to either `true` or `false`, and `nil` always evaluates to `nil`. Other values evaluate to `default`.

Usage

First, load the module. Note that it can only be loaded from other Lua modules, not from normal wiki pages. For normal wiki pages you can use Vorlage:Tl instead.

```
local yesno = require('Module:Yesno')
```

Some input values always return `true`, and some always return `false`. `nil` values always return `nil`.

```
-- These always return true:
yesno('yes')
yesno('y')
yesno('true')
yesno('t')
yesno('1')
yesno(1)
yesno(true)

-- These always return false:
yesno('no')
```

```
yesno('n')
yesno('false')
yesno('f')
yesno('0')
yesno(0)
yesno(false)

-- A nil value always returns nil:
yesno(nil)
```

String values are converted to lower case before they are matched:

```
-- These always return true:
yesno('Yes')
yesno('YES')
yesno('yEs')
yesno('Y')
yesno('tRuE')

-- These always return false:
yesno('No')
yesno('NO')
yesno('nO')
yesno('N')
yesno('fALsE')
```

Undefined input ('foo')

You can specify a default value if yesno receives input other than that listed above. If you don't supply a default, the module will return `nil` for these inputs.

```
-- These return nil:
yesno('foo')
yesno({})
yesno(5)
yesno(function() return 'This is a function.' end)
yesno(nil, true)
yesno(nil, 'bar')

-- These return true:
yesno('foo', true)
yesno({}, true)
yesno(5, true)
yesno(function() return 'This is a function.' end, true)

-- These return "bar":
yesno('foo', 'bar')
yesno({}, 'bar')
yesno(5, 'bar')
yesno(function() return 'This is a function.' end, 'bar')
```

Note that the empty string also functions this way:

```
yesno('')        -- Returns nil.
yesno('', true)  -- Returns true.
yesno('', 'bar') -- Returns "bar".
```

Although the empty string usually evaluates to false in wikitext, it evaluates to true in Lua. This module prefers the Lua behaviour over the wikitext behaviour. If treating the empty string as false is important for your module, you will need to convert empty strings to a value that evaluates to false before passing them to this module. In the case of arguments received from wikitext, this can be done by using Module:Arguments.

Handling nil results

By definition

```
yesno(nil)         -- Returns nil.
yesno('foo')       -- Returns nil.
yesno(nil, true)   -- Returns nil.
yesno(nil, false)  -- Returns nil.
yesno('foo', true) -- Returns true.
```

To get the binary `true/false`-only values, use code like:

```
myvariable = yesno(value) or false -- When value is nil, result is false.
myvariable = yesno(value) or true  -- When value is nil, result is true.
myvariable = yesno('foo') or false  -- Unknown string returns nil, result is fals
myvariable = yesno('foo', true) or false  -- Default value (here: true) applies,
```

```
-- Function allowing for consistent treatment of boolean-like wikitext input.
-- It works similarly to the template {{yesno}}.

return function (val, default)
        -- If your wiki uses non-ascii characters for any of "yes", "no", etc., y
        -- should replace "val:lower()" with "mw.ustring.lower(val)" in the
        -- following line.
        val = type(val) == 'string' and val:lower() or val
        if val == nil then
                return nil
        elseif val == true
                or val == 'yes'
                or val == 'y'
                or val == 'true'
                or val == 't'
                or val == 'on'
                or tonumber(val) == 1
        then
                return true
        elseif val == false
                or val == 'no'
                or val == 'n'
                or val == 'false'
                or val == 'f'
                or val == 'off'
                or tonumber(val) == 0
```

```
        then
                return false
        else
                return default
        end
end
```