

☐ Toggle menu
Blue Gold Program Wiki

Navigation

- [Main page](#)
- [Recent changes](#)
- [Random page](#)
- [Help about MediaWiki](#)

Tools

- [What links here](#)
- [Related changes](#)
- [Special pages](#)
- [Permanent link](#)
- [Page information](#)

Personal tools

- [Log in](#)

personal-extra

☐ Toggle search

Search

Random page

Views

- [View](#)
- [View source](#)
- [History](#)
- [PDF Export](#)

Actions

Module:String

From Blue Gold Program Wiki

The printable version is no longer supported and may have rendering errors. Please update your browser bookmarks and please use the default browser print function instead.

Documentation for this module may be created at [Module:String/doc](#)

```
--[[
```

This module is intended to provide access to basic string functions.

Most of the functions provided here can be invoked with named parameters, unnamed parameters, or a mixture. If named parameters are used, Mediawiki will automatically remove any leading or trailing whitespace from the parameter. Depending on the intended use, it may be advantageous to either preserve or remove such whitespace.

Global options

`ignore_errors`: If set to 'true' or 1, any error condition will result in an empty string being returned rather than an error message.

`error_category`: If an error occurs, specifies the name of a category to include with the error message. The default category is [Category:Errors reported by Module String].

`no_category`: If set to 'true' or 1, no category will be added if an error is generated.

Unit tests for this module are available at `Module:String/tests`.

```
]]
```

```
local str = {}
```

```
--[[
```

```
len
```

This function returns the length of the target string.

Usage:

```
{{#invoke:String|len|target_string|}}
```

OR

```
{{#invoke:String|len|s=target_string|}}
```

Parameters

`s`: The string whose length to report

If invoked using named parameters, Mediawiki will automatically remove any leading or trailing whitespace from the target string.

```
]]
```

```
function str.len( frame )
    local new_args = str._getParameters( frame.args, {'s'} )
    local s = new_args['s'] or ''
    return mw.ustr.len( s )
```

```
end
```

```
--[[
```

sub

This function returns a substring of the target string at specified indices.

Usage:

```
{{#invoke:String|sub|target_string|start_index|end_index}}
```

OR

```
{{#invoke:String|sub|s=target_string|i=start_index|j=end_index}}
```

Parameters

s: The string to return a subset of

i: The first index of the substring to return, defaults to 1.

j: The last index of the string to return, defaults to the last character.

The first character of the string is assigned an index of 1. If either i or j is a negative value, it is interpreted the same as selecting a character by counting from the end of the string. Hence, a value of -1 is the same as selecting the last character of the string.

If the requested indices are out of range for the given string, an error is reported.

```
]]
function str.sub( frame )
    local new_args = str._getParameters( frame.args, { 's', 'i', 'j' } )
    local s = new_args['s'] or ''
    local i = tonumber( new_args['i'] ) or 1
    local j = tonumber( new_args['j'] ) or -1

    local len = mw.ustr.string.len( s )

    -- Convert negatives for range checking
    if i < 0 then
        i = len + i + 1
    end
    if j < 0 then
        j = len + j + 1
    end

    if i > len or j > len or i < 1 or j < 1 then
        return str._error( 'String subset index out of range' )
    end
    if j < i then
        return str._error( 'String subset indices out of order' )
    end

    return mw.ustr.string.sub( s, i, j )
end

--[[
```

This function implements that features of {{str sub old}} and is kept in order to maintain these older templates.

```
]]
function str.sublength( frame )
    local i = tonumber( frame.args.i ) or 0
    local len = tonumber( frame.args.len )
    return mw.usttring.sub( frame.args.s, i + 1, len and ( i + len ) )
end
```

```
--[[
_match
```

This function returns a substring from the source string that matches a specified pattern. It is exported for use in other modules

Usage:

```
strmatch = require("Module:String")._match
sresult = strmatch( s, pattern, start, match, plain, nomatch )
```

Parameters

- s: The string to search
- pattern: The pattern or string to find within the string
- start: The index within the source string to start the search. The first character of the string has index 1. Defaults to 1.
- match: In some cases it may be possible to make multiple matches on a single string. This specifies which match to return, where the first match is match= 1. If a negative number is specified then a match is returned counting from the last match. Hence match = -1 is the same as requesting the last match. Defaults to 1.
- plain: A flag indicating that the pattern should be understood as plain text. Defaults to false.
- nomatch: If no match is found, output the "nomatch" value rather than an error.

For information on constructing Lua patterns, a form of [regular expression], see:

* <http://www.lua.org/manual/5.1/manual.html#5.4.1>

*

http://www.mediawiki.org/wiki/Extension:Scribunto/Lua_reference_manual#Patterns

*

http://www.mediawiki.org/wiki/Extension:Scribunto/Lua_reference_manual#Ustring_patterns

```
]]
```

```
-- This sub-routine is exported for use in other modules
```

```

function str._match( s, pattern, start, match_index, plain_flag, nomatch )
    if s == '' then
        return str._error( 'Target string is empty' )
    end
    if pattern == '' then
        return str._error( 'Pattern string is empty' )
    end
    start = tonumber(start) or 1
    if math.abs(start) < 1 or math.abs(start) > mw.ustring.len( s ) then
        return str._error( 'Requested start is out of range' )
    end
    if match_index == 0 then
        return str._error( 'Match index is out of range' )
    end
    if plain_flag then
        pattern = str._escapePattern( pattern )
    end

    local result
    if match_index == 1 then
        -- Find first match is simple case
        result = mw.ustring.match( s, pattern, start )
    else
        if start > 1 then
            s = mw.ustring.sub( s, start )
        end

        local iterator = mw.ustring.gmatch(s, pattern)
        if match_index > 0 then
            -- Forward search
            for w in iterator do
                match_index = match_index - 1
                if match_index == 0 then
                    result = w
                    break
                end
            end
        else
            -- Reverse search
            local result_table = {}
            local count = 1
            for w in iterator do
                result_table[count] = w
                count = count + 1
            end

            result = result_table[ count + match_index ]
        end
    end

    if result == nil then

```

```

        if nomatch == nil then
            return str._error( 'Match not found' )
        else
            return nomatch
        end
    else
        return result
    end
end

--[[
match

```

This function returns a substring from the source string that matches a specified pattern.

Usage:

```

{{#invoke:String|match|source_string|pattern_string|start_index|match_number|
plain_flag|nomatch_output}}
OR
{{#invoke:String|match|s=source_string|pattern=pattern_string|start=start_ind
ex
|match=match_number|plain=plain_flag|nomatch=nomatch_output}}

```

Parameters

- s: The string to search
- pattern: The pattern or string to find within the string
- start: The index within the source string to start the search. The first character of the string has index 1. Defaults to 1.
- match: In some cases it may be possible to make multiple matches on a single string. This specifies which match to return, where the first match is match= 1. If a negative number is specified then a match is returned counting from the last match. Hence match = -1 is the same as requesting the last match. Defaults to 1.
- plain: A flag indicating that the pattern should be understood as plain text. Defaults to false.
- nomatch: If no match is found, output the "nomatch" value rather than an error.

If invoked using named parameters, Mediawiki will automatically remove any leading or trailing whitespace from each string. In some circumstances this is desirable, in other cases one may want to preserve the whitespace.

If the match_number or start_index are out of range for the string being queried, then this function generates an error. An error is also generated if no match is

found.

If one adds the parameter `ignore_errors=true`, then the error will be suppressed and an empty string will be returned on any failure.

For information on constructing Lua patterns, a form of [regular expression], see:

* <http://www.lua.org/manual/5.1/manual.html#5.4.1>

*

http://www.mediawiki.org/wiki/Extension:Scribunto/Lua_reference_manual#Patterns

*

http://www.mediawiki.org/wiki/Extension:Scribunto/Lua_reference_manual#Ustring_patterns

```
]]
```

```
-- This is the entry point for #invoke:String|match
```

```
function str.match( frame )
```

```
    local new_args = str._getParameters( frame.args, {'s', 'pattern',  
'start', 'match', 'plain', 'nomatch'} )
```

```
    local s = new_args['s'] or ''
```

```
    local start = tonumber( new_args['start'] ) or 1
```

```
    local plain_flag = str._getBoolean( new_args['plain'] or false )
```

```
    local pattern = new_args['pattern'] or ''
```

```
    local match_index = math.floor( tonumber(new_args['match']) or 1 )
```

```
    local nomatch = new_args['nomatch']
```

```
    return str._match( s, pattern, start, match_index, plain_flag,  
nomatch )
```

```
end
```

```
--[[
```

```
pos
```

This function returns a single character from the target string at position `pos`.

Usage:

```
{{#invoke:String|pos|target_string|index_value}}
```

OR

```
{{#invoke:String|pos|target=target_string|pos=index_value}}
```

Parameters

target: The string to search

pos: The index for the character to return

If invoked using named parameters, Mediawiki will automatically remove any leading or trailing whitespace from the target string. In some circumstances this is desirable, in

other cases one may want to preserve the whitespace.

The first character has an index value of 1.

If one requests a negative value, this function will select a character by counting backwards from the end of the string. In other words `pos = -1` is the same as asking for the last character.

A requested value of zero, or a value greater than the length of the string returns an error.

```
]]
function str.pos( frame )
    local new_args = str._getParameters( frame.args, {'target', 'pos'} )
    local target_str = new_args['target'] or ''
    local pos = tonumber( new_args['pos'] ) or 0

    if pos == 0 or math.abs(pos) > mw.usttring.len( target_str ) then
        return str._error( 'String index out of range' )
    end

    return mw.usttring.sub( target_str, pos, pos )
end

--[[
str_find
```

This function duplicates the behavior of `{{str_find}}`, including all of its quirks.

This is provided in order to support existing templates, but is NOT RECOMMENDED for new code and templates. New code is recommended to use the "find" function instead.

Returns the first index in "source" that is a match to "target". Indexing is 1-based, and the function returns -1 if the "target" string is not present in "source".

Important Note: If the "target" string is empty / missing, this function returns a value of "1", which is generally unexpected behavior, and must be accounted for separately.

```
]]
function str.str_find( frame )
    local new_args = str._getParameters( frame.args, {'source', 'target'} )

    local source_str = new_args['source'] or ''
    local target_str = new_args['target'] or ''
```



```

        if target_str == '' then
            return 1
        end

        local start = mw.usttring.find( source_str, target_str, 1, true )
        if start == nil then
            start = -1
        end

        return start
    end

end

--[[
find

```

This function allows one to search for a target string or pattern within another string.

Usage:

```

{{#invoke:String|find|source_str|target_string|start_index|plain_flag}}
OR
{{#invoke:String|find|source=source_str|target=target_str|start=start_index|plain=plain_flag}}

```

Parameters

- source: The string to search
- target: The string or pattern to find within source
- start: The index within the source string to start the search, defaults to 1
- plain: Boolean flag indicating that target should be understood as plain text and not as a Lua style regular expression, defaults to true

If invoked using named parameters, Mediawiki will automatically remove any leading or trailing whitespace from the parameter. In some circumstances this is desirable, in other cases one may want to preserve the whitespace.

This function returns the first index \geq "start" where "target" can be found within "source". Indices are 1-based. If "target" is not found, then this function returns 0. If either "source" or "target" are missing / empty, this function also returns 0.

This function should be safe for UTF-8 strings.

```

]]
function str.find( frame )
    local new_args = str._getParameters( frame.args, { 'source', 'target', 'start', 'plain' } )
    local source_str = new_args['source'] or ''
    local pattern = new_args['target'] or ''

```

```

    local start_pos = tonumber(new_args['start']) or 1
    local plain = new_args['plain'] or true

    if source_str == '' or pattern == '' then
        return 0
    end

    plain = str._getBoolean( plain )

    local start = mw.usttring.find( source_str, pattern, start_pos, plain
)
    if start == nil then
        start = 0
    end

    return start
end

--[[
replace

```

This function allows one to replace a target string or pattern within another string.

Usage:

```

{{#invoke:String|replace|source_str|pattern_string|replace_string|replacement_count|plain_flag}}

```

OR

```

{{#invoke:String|replace|source=source_string|pattern=pattern_string|replace=replace_string|
count=replacement_count|plain=plain_flag}}

```

Parameters

- source: The string to search
- pattern: The string or pattern to find within source
- replace: The replacement text
- count: The number of occurrences to replace, defaults to all.
- plain: Boolean flag indicating that pattern should be understood as plain text and not as a Lua style regular expression, defaults to true

```

]]

```

```

function str.replace( frame )
    local new_args = str._getParameters( frame.args, {'source',
'pattern', 'replace', 'count', 'plain' } )
    local source_str = new_args['source'] or ''
    local pattern = new_args['pattern'] or ''
    local replace = new_args['replace'] or ''
    local count = tonumber( new_args['count'] )
    local plain = new_args['plain'] or true

    if source_str == '' or pattern == '' then
        return source_str
    end

```

```

    end
    plain = str._getBoolean( plain )

    if plain then
        pattern = str._escapePattern( pattern )
        replace = mw.usttring.gsub( replace, "%%", "%%%" ) --Only
need to escape replacement sequences.
    end

    local result

    if count ~= nil then
        result = mw.usttring.gsub( source_str, pattern, replace, count
)
    else
        result = mw.usttring.gsub( source_str, pattern, replace )
    end

    return result
end

--[[
    simple function to pipe string.rep to templates.
]]
function str.rep( frame )
    local repetitions = tonumber( frame.args[2] )
    if not repetitions then
        return str._error( 'function rep expects a number as second
parameter, received "' .. ( frame.args[2] or '' ) .. '"' )
    end
    return string.rep( frame.args[1] or '', repetitions )
end

--[[
escapePattern

```

This function escapes special characters from a Lua string pattern. See [1] for details on how patterns work.

[1]
https://www.mediawiki.org/wiki/Extension:Scribunto/Lua_reference_manual#Patterns

Usage:

```
{{#invoke:String|escapePattern|pattern_string}}
```

Parameters

pattern_string: The pattern string to escape.

```

]]
function str.escapePattern( frame )
    local pattern_str = frame.args[1]

```

```

        if not pattern_str then
            return str._error( 'No pattern string specified' )
        end
        local result = str._escapePattern( pattern_str )
        return result
    end
end

```

```

--[[
count
This function counts the number of occurrences of one string in another.
]]

```

```

function str.count(frame)
    local args = str._getParameters(frame.args, {'source', 'pattern',
'plain'})
    local source = args.source or ''
    local pattern = args.pattern or ''
    local plain = str._getBoolean(args.plain or true)
    if plain then
        pattern = str._escapePattern(pattern)
    end
    local _, count = mw.ustring.gsub(source, pattern, '')
    return count
end

```

```

--[[
endswith
This function determines whether a string ends with another string.
]]
function str.endswith(frame)
    local args = str._getParameters(frame.args, {'source', 'pattern'})
    local source = args.source or ''
    local pattern = args.pattern or ''
    if pattern == '' then
        -- All strings end with the empty string.
        return "yes"
    end
    if mw.ustring.sub(source, -mw.ustring.len(pattern), -1) == pattern
then
        return "yes"
    else
        return ""
    end
end

```

```

--[[
join

```

```

Join all non empty arguments together; the first argument is the separator.
Usage:
{{#invoke:String|join|sep|one|two|three}}
]]

```

```

function str.join(frame)
    local args = {}
    local sep
    for _, v in ipairs( frame.args ) do
        if sep then
            if v ~= '' then
                table.insert(args, v)
            end
        else
            sep = v
        end
    end
    return table.concat( args, sep or '' )
end

```

```

--[[
Helper function that populates the argument list given that user may need to
use a mix of
named and unnamed parameters. This is relevant because named parameters are
not
identical to unnamed parameters due to string trimming, and when dealing with
strings
we sometimes want to either preserve or remove that whitespace depending on
the application.
]]

```

```

function str._getParameters( frame_args, arg_list )
    local new_args = {}
    local index = 1
    local value

    for _, arg in ipairs( arg_list ) do
        value = frame_args[arg]
        if value == nil then
            value = frame_args[index]
            index = index + 1
        end
        new_args[arg] = value
    end

    return new_args
end

```

```

--[[
Helper function to handle error messages.
]]
function str._error( error_str )
    local frame = mw.getCurrentFrame()
    local error_category = frame.args.error_category or 'Errors reported
by Module String'
    local ignore_errors = frame.args.ignore_errors or false
    local no_category = frame.args.no_category or false

```

```

        if str._getBoolean(ignore_errors) then
            return ''
        end

        local error_str = '<strong class="error">String Module Error: ' ..
error_str .. '</strong>'
        if error_category ~= '' and not str._getBoolean( no_category ) then
            error_str = '[[Category:' .. error_category .. ']]' ..
error_str
        end

        return error_str
end

--[[
Helper Function to interpret boolean strings
]]
function str._getBoolean( boolean_str )
    local boolean_value

    if type( boolean_str ) == 'string' then
        boolean_str = boolean_str:lower()
        if boolean_str == 'false' or boolean_str == 'no' or
boolean_str == '0'
            or boolean_str == '' then
            boolean_value = false
        else
            boolean_value = true
        end
    elseif type( boolean_str ) == 'boolean' then
        boolean_value = boolean_str
    else
        error( 'No boolean value found' )
    end
    return boolean_value
end

--[[
Helper function that escapes all pattern characters so that they will be
treated
as plain text.
]]
function str._escapePattern( pattern_str )
    return mw.ustr.gsub( pattern_str, "([%(%)%.%%%+%-%*%?%[%^$%]])",
"%%%1" )
end

return str

```

Retrieved from "<https://www.bluegoldwiki.com/index.php?title=Module:String&oldid=3532>"

Namespaces

- [Module](#)
- [Discussion](#)

Variants

This page was last edited on 23 August 2020, at 06:04.

Blue Gold Program Wiki

The wiki version of the Lessons Learnt Report of the Blue Gold program, documents the experiences of a technical assistance (TA) team working in a development project implemented by the Bangladesh Water Development Board (BWDB) and the Department of Agricultural Extension (DAE) over an eight+ year period from March 2013 to December 2021. The wiki lessons learnt report (LLR) is intended to complement the BWDB and DAE project completion reports (PCRs), with the aim of recording lessons learnt for use in the design and implementation of future interventions in the coastal zone.

- [Privacy policy](#)
- [About Blue Gold Program Wiki](#)
- [Disclaimers](#)

Developed and maintained by Big Blue Communications for Blue Gold Program



[Blue Gold Program Wiki](#)