

☐ 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:Citation/CS1/Utilities

From Blue Gold Program Wiki

< [Module:Citation/CS1](#)

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.

{{#lst:Module:Citation/CS1/doc|header}} This page contains various functions and tables that are common to multiple of the various modules that make up [Module:Citation/CS1](#).

```
{{#lst:Module:Citation/CS1/doc|module_components_table}}
```

```
local z = {
    error_categories = {};
-- for categorizing citations that contain errors
    error_ids = {};
    message_tail = {};
    maintenance_cats = {};
-- for categorizing citations that aren't erroneous per se, but could use a
little work
    properties_cats = {};
-- for categorizing citations based on certain properties, language of source
for instance
};
```

```
--[[-----< F O R W A R D   D E C L A R A T I O N S >-----
-----
]]
```

```
local cfg;
-- table of tables imported from selected Module:Citation/CS1/Configuration
```

```
--[[-----< I S _ S E T >-----
-----
```

Returns true if argument is set; false otherwise. Argument is 'set' when it exists (not nil) or when it is not an empty string.

```
]]
```

```
local function is_set( var )
    return not (var == nil or var == '');
end
```

```
--[[-----< I N _ A R R A Y >-----
-----
```

Whether needle is in haystack

```
]]
```

```
local function in_array( needle, haystack )
    if needle == nil then
        return false;
    end
```

```

        for n,v in ipairs( haystack ) do
            if v == needle then
                return n;
            end
        end
        return false;
    end
end

```

```

--[[-----< S U B S T I T U T E >-----
-----

```

Populates numbered arguments in a message string using an argument table.

```

]]

local function substitute( msg, args )
    return args and mw.message.newRawMessage( msg, args ):plain() or msg;
end

```

```

--[[-----< E R R O R _ C O M M E N T >-----
-----

```

Wraps error messages with css markup according to the state of hidden.

```

]]

local function error_comment( content, hidden )
    return substitute( hidden and cfg.presentation['hidden-error'] or
cfg.presentation['visible-error'], content );
end

```

```

--[=[-----< M A K E _ W I K I L I N K >-----
-----

```

Makes a wikilink; when bot link and display text is provided, returns a wikilink in the form [[L|D]]; if only link is provided, returns a wikilink in the form [[L]]; if neither are provided or link is omitted, returns an empty string.

```

]=]

local function make_wikilink (link, display)
    if is_set (link) then
        if is_set (display) then
            return table.concat ({'[[', link, '|', display,
'']]');
        else

```

```

        return table.concat ({'[[', link, ']]'});
    end
else
    return '';
end
end

--[[-----< S E T _ E R R O R >-----
-----

```

Sets an error condition and returns the appropriate error message. The actual placement of the error message in the output is the responsibility of the calling function.

```
]]
```

```

local function set_error( error_id, arguments, raw, prefix, suffix )
    local error_state = cfg.error_conditions[ error_id ];
    prefix = prefix or "";
    suffix = suffix or "";
    if error_state == nil then
        error( cfg.messages['undefined_error'] );
-- because missing error handler in Module:Citation/CS1/Configuration
    elseif is_set( error_state.category ) then
        table.insert( z.error_categories, error_state.category );
    end
    local message = substitute( error_state.message, arguments );

    message = table.concat (
        {
            message,
            ' (',
            make_wikilink (
                table.concat (
                    {
                        cfg.messages['help page link'],
                        '#',
                        error_state.anchor
                    },
                    cfg.messages['help page label'] ),
                ),
            ')'
        },
        {} );

    z.error_ids[ error_id ] = true;
    if in_array( error_id, { 'bare_url_missing_title',
'trans_missing_title' } )
        and z.error_ids['citation_missing_title'] then
        return '', false;
    end
    message = table.concat({ prefix, message, suffix });

```

```

        if raw == true then
            return message, error_state.hidden;
        end
        return error_comment( message, error_state.hidden );
    end
end

```

```

--[[-----< I S _ A L I A S _ U S E D >-----
-----

```

This function is used by `select_one()` to determine if one of a list of alias parameters is in the argument list provided by the template.

Input:

```

    args – pointer to the arguments table from calling template
    alias – one of the list of possible aliases in the aliases lists from
Module:Citation/CS1/Configuration
    index – for enumerated parameters, identifies which one
    enumerated – true/false flag used choose how enumerated aliases are
examined
    value – value associated with an alias that has previously been
selected; nil if not yet selected
    selected – the alias that has previously been selected; nil if not
yet selected
    error_list – list of aliases that are duplicates of the alias already
selected

```

Returns:

```

    value – value associated with alias we selected or that was
previously selected or nil if an alias not yet selected
    selected – the alias we selected or the alias that was previously
selected or nil if an alias not yet selected

```

```

]]

```

```

local function is_alias_used (args, alias, index, enumerated, value,
selected, error_list)
    if enumerated then
-- is this a test for an enumerated parameters?
        alias = alias:gsub ('#', index);
-- replace '#' with the value in index
    else
        alias = alias:gsub ('#', '');
-- remove '#' if it exists
    end

    if is_set(args[alias]) then
-- alias is in the template's argument list
        if value ~= nil and selected ~= alias then
-- if we have already selected one of the aliases

```

```

        local skip;
        for _, v in ipairs(error_list) do
-- spin through the error list to see if we've added this alias
            if v == alias then
                skip = true;
                break;
-- has been added so stop looking
            end
        end
        if not skip then
-- has not been added so
            table.insert( error_list, alias );
-- add error alias to the error list
        end
    else
        value = args[alias];
-- not yet selected an alias, so select this one
        selected = alias;
    end
end
return value, selected;
-- return newly selected alias, or previously selected alias
end

```

```

--[[-----< A D D _ M A I N T _ C A T >-----
-----

```

Adds a category to z.maintenance_cats using names from the configuration file with additional text if any.

To prevent duplication, the added_maint_cats table lists the categories by key that have been added to z.maintenance_cats.

```

]]

```

```

local added_maint_cats = {}
-- list of maintenance categories that have been added to z.maintenance_cats
local function add_maint_cat (key, arguments)
    if not added_maint_cats [key] then
        added_maint_cats [key] = true;
-- note that we've added this category
        table.insert( z.maintenance_cats, substitute (cfg.maint_cats
[key], arguments));    -- make name then add to table
    end
end

```

```

--[[-----< S A F E _ F O R _ I T A L I C S >-----
-----

```

Protects a string that will be wrapped in wiki italic markup '' ... ''

Note: We cannot use `<i>` for italics, as the expected behavior for italics specified by `''...''` in the title is that they will be inverted (i.e. unitalicized) in the resulting references. In addition, `<i>` and `''` tend to interact poorly under Mediawiki's HTML tidy.

```
]]

local function safe_for_italics( str )
    if not is_set(str) then
        return str;
    else
        if str:sub(1,1) == "'" then str = "<span></span>" .. str; end
        if str:sub(-1,-1) == "'" then str = str .. "<span></span>";
    end
    -- Remove newlines as they break italics.
    return str:gsub( '\n', ' ' );
end
end
```

```
--[[-----< W R A P _ S T Y L E >-----
-----
```

Applies styling to various parameters. Supplied string is wrapped using a `message_list` configuration taking one argument; protects italic styled parameters. Additional text taken from `citation_config.presentation` - the reason this function is similar to but separate from `wrap_msg()`.

```
]]

local function wrap_style (key, str)
    if not is_set( str ) then
        return "";
    elseif in_array( key, { 'italic-title', 'trans-italic-title' } ) then
        str = safe_for_italics( str );
    end

    return substitute( cfg.presentation[key], {str} );
end
```

```
--[[-----< S E L E C T _ O N E >-----
-----
```

Chooses one matching parameter from a list of parameters to consider. The list of parameters to consider is just names. For parameters that may be enumerated, the position of the numerator in the parameter name is identified by the `#` so `|author-last1=` and `|author1-last=` are represented as `'author-`

last#' and 'author#-last'.

Because enumerated parameter |<param>1= is an alias of |<param>= we must test for both possibilities.

Generates an error if more than one match is present.

```
]]

local function select_one( args, aliases_list, error_condition, index )
    local value = nil;
-- the value assigned to the selected parameter
    local selected = '';
-- the name of the parameter we have chosen
    local error_list = {};

    if index ~= nil then index = tostring(index); end

    for _, alias in ipairs( aliases_list ) do
-- for each alias in the aliases list
        if alias:match('#') then
-- if this alias can be enumerated
            if '1' == index then
-- when index is 1 test for enumerated and non-enumerated aliases
                value, selected = is_alias_used( args, alias,
index, false, value, selected, error_list);          -- first test for non-
enumerated alias
            end
                value, selected = is_alias_used( args, alias, index,
true, value, selected, error_list);          -- test for enumerated
alias
            else
                value, selected = is_alias_used( args, alias, index,
false, value, selected, error_list);          --test for non-enumerated
alias
            end
        end
    end

    if #error_list > 0 and 'none' ~= error_condition then
-- for cases where this code is used outside of extract_names()
        local error_str = "";
        for _, k in ipairs( error_list ) do
            if error_str ~= "" then error_str = error_str ..
cfg.messages['parameter-separator'] end
            error_str = error_str .. wrap_style( 'parameter', k);
        end
        if #error_list > 1 then
            error_str = error_str .. cfg.messages['parameter-
final-separator'];
        else
```

```

        error_str = error_str .. cfg.messages['parameter-
pair-separator'];
    end
    error_str = error_str .. wrap_style ('parameter', selected);
    table.insert( z.message_tail, { set_error( error_condition,
{error_str}, true ) } );
    end
    return value, selected;
end

```

```

--[=[-----< R E M O V E _ W I K I _ L I N K >-----
-----

```

Gets the display text from a wikilink like [[A|B]] or [[B]] gives B

The str:gsub() returns either A|B froma [[A|B]] or B from [[B]] or B from B (no wikilink markup).

In l(), l:gsub() removes the link and pipe (if they exist); the second :gsub() trims white space from the label
if str was wrapped in wikilink markup. Presumably, this is because without wikimarkup in str, there is no match
in the initial gsub, the replacement function l() doesn't get called.

```

]=]

```

```

local function remove_wiki_link (str)
    return (str:gsub( "%[%[[^%[%]]*)%]", function(l)
        return l:gsub( "^[^|]*|(.*)$", "%1" ):gsub("^%s*(.)%s*$",
"%1");
    end));
end

```

```

--[=[-----< I S _ W I K I L I N K >-----
-----

```

Determines if str is a wikilink, extracts, and returns the the wikilink type, link text, and display text parts.

If str is a complex wikilink ([[L|D]]):

returns wl_type 2 and D and L from [[L|D]];

if str is a simple wikilink ([[D]])

returns wl_type 1 and D from [[D]] and L as empty string;

if not a wikilink:

returns wl_type 0, str as D, and L as empty string.

trims leading and trailing white space and pipes from L and D ([[L|]] and [[|D]]) are accepted by MediaWiki and
treated like [[D]]; while [[|D|]] is not accepted by MediaWiki, here, we accept it and return D without the pipes).

```

]=]

local function is_wikilink (str)
    local D, L
    local wl_type = 2;
-- assume that str is a complex wikilink [[L|D]]

    if not str:match ('^%[%[^\%]]+[%]$\') then
-- is str some sort of a wikilink (must have some sort of content)
        return 0, str, '';
-- not a wililink; return wl_type as 0, str as D, and empty string as L
    end
    L, D = str:match ('^%[%[([^\|]+)|([^\%]]+)%[%]$\');
-- get L and D from [[L|D]]

    if not is_set (D) then
-- if no separate display
        D = str:match ('^%[%[([^\%]]*)|*[%]$\');
-- get D from [[D]] or [[D|]]
        wl_type = 1;
    end
    D = mw.text.trim (D, '%s|');
-- trim white space and pipe characters
-- L = L and mw.text.trim (L, '%s|');
    return wl_type, D, L or '';
end

--[[-----< S T R I P _ A P O S T R O P H E _ M A R K U P
>-----

Strip wiki italic and bold markup from argument so that it doesn't
contaminate COinS metadata.
This function strips common patterns of apostrophe markup. We presume that
editors who have taken the time to
markup a title have, as a result, provided valid markup. When they don't,
some single apostrophes are left behind.

Returns the argument without wiki markup and a number; the number is more-or-
less meaningless except as a flag
to indicate that markup was replaced; do not rely on it as an indicator of
how many of any kind of markup was
removed; returns the argument and nil when no markup removed

]]

local function strip_apostrophe_markup (argument)
    if not is_set (argument) then
        return argument, nil;
-- no argument, nothing to do
    end

```

```

        if nil == argument:find ( "'", 1, true ) then
-- Is there at least one double apostrophe?  If not, exit.
            return argument, nil;
        end

        local flag;
        while true do
            if argument:find ( "''''", 1, true ) then
-- bold italic (5)
                argument, flag=argument:gsub("%'%'%'%''", "");
-- remove all instances of it
            elseif argument:find ( "''''", 1, true ) then
-- italic start and end without content (4)
                argument, flag=argument:gsub("%'%'%''", "");
            elseif argument:find ( "'''", 1, true ) then
-- bold (3)
                argument, flag=argument:gsub("%'%''", "");
            elseif argument:find ( "'", 1, true ) then
-- italic (2)
                argument, flag=argument:gsub("%'%"', "");
            else
                break;
            end
        end

        return argument, flag;
-- done
end

--[[-----< S E T _ S E L E C T E D _ M O D U L E S >-----
-----

Sets local cfg table to same (live or sandbox) as that used by the other
modules.

]]

local function set_selected_modules (cfg_table_ptr)
    cfg = cfg_table_ptr;
end

--[[-----< E X P O R T S >-----
-----

]]

return {
    is_set = is_set,
-- exported functions
    in_array = in_array,

```

```

        substitute = substitute,
        error_comment = error_comment,
        set_error = set_error,
        select_one = select_one,
        add_maint_cat = add_maint_cat,
        wrap_style = wrap_style,
        safe_for_italics = safe_for_italics,
        remove_wiki_link = remove_wiki_link,
        is_wikilink = is_wikilink,
        make_wikilink = make_wikilink,
        set_selected_modules = set_selected_modules,
        strip_apostrophe_markup = strip_apostrophe_markup,
        z = z,
-- exported table
    }

```

actions taken to prevent or repair the deterioration of water management infrastructure and to keep the physical components of a water management system in such a state that they can serve their intended function.

Retrieved from

"<https://www.bluegoldwiki.com/index.php?title=Module:Citation/CS1/Utilities&oldid=1605>"

Namespaces

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

Variants

This page was last edited on 19 February 2020, at 07:06.

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](#)