Link Count API

Request

- **page** - required string - Name of the page to get the link count for.
- **project** - optional string - Project (domain, name, or database) the page is in, default is en.wikipedia.org.
- **namespace** - optional string - Comma-separated list of namespace numbers which links from are counted. Leave blank to count all namespaces.

Response

- **filelinks** - optional LinkCountObject - Number of pages that show the file.
- **categorylinks** - optional LinkCountObject - Number of category links.
- **wikilinks** - required LinkCountObject - Number of wikilinks.
- **redirects** - required integer - Number of redirects to the page.
- **transclusions** - required LinkCountObject - Number of page that transclude the page.

LinkCountObject

- **all** - required integer - Sum of direct and indirect links.
- **direct** - required integer - Number of links the directly link to the page.
- **indirect** - required integer - Number of links that link to the page through a redirect.

Error Response

- **error** - required string - Message explaining the error that occurred.

Module:Citation/CS1/sandbox

From Wikipedia, the free encyclopedia
< Module:Citation | CS1

> This is the module sandbox page for Module:Citation/CS1 (diff).
> See also the companion subpage for test cases (run).

{{i}} Module documentation [view] [edit] [history] [purge]

This module and associated sub-modules support the Citation Style 1 and Citation Style 2 citation templates. In general, it is not intended to be called directly, but is called by one of the core CS1 and CS2 templates.

These files comprise the module support for CS1|2 citation templates:

CS1 | CS2 modules

live		sandbox		diff	description
Module:Citation/CS1		**Module:Citation/CS1/sandbox**	[edit]	diff	Rendering and support functions
Module:Citation/CS1/Configuration		Module:Citation/CS1/Configuration/sandbox	[edit]	diff	Translation tables; error and

					identifier handlers
	Module:Citation/CS1/Whitelist	Module:Citation/CS1/Whitelist/sandbox	[edit]	diff	List of active and deprecated CS1|2 parameters
	Module:Citation/CS1/Date validation	Module:Citation/CS1/Date validation/sandbox	[edit]	diff	Date format validation functions
	Module:Citation/CS1/Identifiers	Module:Citation/CS1/Identifiers/sandbox	[edit]	diff	Functions that support the named identifiers (ISBN, DOI, PMID, etc.)
	Module:Citation/CS1/Utilities	Module:Citation/CS1/Utilities/sandbox	[edit]	diff	Common functions and tables
	Module:Citation/CS1/COinS	Module:Citation/CS1/COinS/sandbox	[edit]	diff	Functions that render a CS1|2 template's metadata
	Module:Citation/CS1/styles.css	Module:Citation/CS1/sandbox/styles.css	[edit]	diff	CSS styles applied to the CS1|2 templates
	Module:Citation/CS1/Suggestions	Module:Citation/CS1/Suggestions/sandbox	[edit]	diff	List that maps common erroneous parameter names to valid parameter names

Other documentation:

- Module talk:Citation/CS1/Feature requests
- Module talk:Citation/CS1/COinS
- Module:Cs1 documentation support – a set of functions (some experimental) that extract information from the module suite for the purpose of documenting CS1|2
  - Module:Citation/CS1/doc/Category list – lists of category names taken directly from Module:Citation/CS1/Configuration and Module:Citation/CS1/Configuration/sandbox

testcases

Module:Citation/CS1/testcases (run)

Module:Citation/CS1/testcases/errors (run) – error and maintenance messaging

Module:Citation/CS1/testcases/dates (run) – date validation

Module:Citation/CS1/testcases/identifiers (run) – identifiers

Module:Citation/CS1/testcases/anchor (run) – CITEREF anchors

The above documentation is transcluded from Module:Citation/CS1/sandbox/doc. (edit | history)
Editors can experiment in this module's ***sandbox*** (edit | diff) *and testcases* (edit | run) *pages.*
Subpages of this module.

```
--[[
History of changes since last sync: 2022-01-26

2022-01-31: move {{citation}} specific |volume= & |issue= lists from to
~/Configuration; see Help_talk:Citation_Style_1#%7Cissue%3D_and_%7Cvolume%3D_i18n
2022-02-14: script param error msg supplements for real this time; see
Help_talk:Citation_Style_1#i18n_%7Cscript-
<param>%3D_error_message_supplements_%232
2022-03-14: |format= bug fix in cite episode; see
Help_talk:Citation_Style_1#cite_episode_complains_about_formatting_errors_that_do
nt_exist
2022-03-16: emit error msg when using identifier-created url and has |archive-
url=; see Help_talk:Citation_Style_1#archive-url_and_identifier-created_urls

]]

require ('Module:No globals');

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

each of these counts against the Lua upvalue limit

]]

local validation;
-- functions in Module:Citation/CS1/Date_validation

local utilities;
-- functions in Module:Citation/CS1/Utilities
local z ={};
-- table of tables in Module:Citation/CS1/Utilities

local identifiers;
-- functions and tables in Module:Citation/CS1/Identifiers
local metadata;
-- functions in Module:Citation/CS1/COinS
local cfg = {};
-- table of configuration tables that are defined in
Module:Citation/CS1/Configuration
local whitelist = {};
-- table of tables listing valid template parameter names; defined in
Module:Citation/CS1/Whitelist


--[[-------------------< P A G E   S C O P E   V A R I A B L E S >---------------

declare variables here that have page-wide scope that are not brought in from
```

```
other modules; that are created here and used here

]]

local added_deprecated_cat;
-- Boolean flag so that the category is added only once
local added_vanc_errs;
-- Boolean flag so we only emit one Vancouver error / category
local added_generic_name_errs;
-- Boolean flag so we only emit one generic name error / category and stop
testing names once an error is encountered
local Frame;
-- holds the module's frame table
local is_preview_mode;
-- true when article is in preview mode; false when using 'Preview page with this
template' (previewing the module)
local is_sandbox;
-- true when using sandbox modules to render citation


--[[--------------------------< F I R S T _ S E T >--------------------------------
------------------------------

Locates and returns the first set value in a table of values where the order
established in the table,
left-to-right (or top-to-bottom), is the order in which the values are evaluated.
Returns nil if none are set.

This version replaces the original 'for _, val in pairs do' and a similar version
that used ipairs.  With the pairs
version the order of evaluation could not be guaranteed.  With the ipairs
version, a nil value would terminate
the for-loop before it reached the actual end of the list.

]]

local function first_set (list, count)
        local i = 1;
        while i <= count do
-- loop through all items in list
                if utilities.is_set( list[i] ) then
                        return list[i];
-- return the first set list member
                end
                i = i + 1;
-- point to next
        end
end


--[[--------------------------< A D D _ V A N C _ E R R O R >--------------------
--------------------------------

Adds a single Vancouver system error message to the template's output regardless
of how many error actually exist.
To prevent duplication, added_vanc_errs is nil until an error message is emitted.

added_vanc_errs is a Boolean declared in page scope variables above

]]

local function add_vanc_error (source, position)
```

```
        if added_vanc_errs then return end

        added_vanc_errs = true;
-- note that we've added this category
        utilities.set_message ('err_vancouver', {source, position});
end


--[[--------------------------< I S _ S C H E M E >--------------------------------
------------------------------

does this thing that purports to be a URI scheme seem to be a valid scheme?  The
scheme is checked to see if it
is in agreement with http://tools.ietf.org/html/std66#section-3.1 which says:
        Scheme names consist of a sequence of characters beginning with a
   letter and followed by any combination of letters, digits, plus
   ("+"), period ("."), or hyphen ("-").

returns true if it does, else false

]]

local function is_scheme (scheme)
        return scheme and scheme:match ('^%a[%a%d%+%.%-]*:');
-- true if scheme is set and matches the pattern
end


--[=[--------------------------< I S _ D O M A I N _ N A M E >--------------------
------------------------------

Does this thing that purports to be a domain name seem to be a valid domain name?

Syntax defined here: http://tools.ietf.org/html/rfc1034#section-3.5
BNF defined here: https://tools.ietf.org/html/rfc4234
Single character names are generally reserved; see
https://tools.ietf.org/html/draft-ietf-dnsind-iana-dns-01#page-15;
        see also [[Single-letter second-level domain]]
list of TLDs: https://www.iana.org/domains/root/db

RFC 952 (modified by RFC 1123) requires the first and last character of a
hostname to be a letter or a digit.  Between
the first and last characters the name may use letters, digits, and the hyphen.

Also allowed are IPv4 addresses. IPv6 not supported

domain is expected to be stripped of any path so that the last character in the
last character of the TLD.  tld
is two or more alpha characters.  Any preceding '//' (from splitting a URL with a
scheme) will be stripped
here.  Perhaps not necessary but retained in case it is necessary for IPv4 dot
decimal.

There are several tests:
        the first character of the whole domain name including subdomains must be
a letter or a digit
        internationalized domain name (ASCII characters with .xn-- ASCII
Compatible Encoding (ACE) prefix xn-- in the TLD) see
https://tools.ietf.org/html/rfc3490
        single-letter/digit second-level domains in the .org, .cash, and .today
TLDs
        q, x, and z SL domains in the .com TLD
```

```
        i and q SL domains in the .net TLD
        single-letter SL domains in the ccTLDs (where the ccTLD is two letters)
        two-character SL domains in gTLDs (where the gTLD is two or more letters)
        three-plus-character SL domains in gTLDs (where the gTLD is two or more
letters)
        IPv4 dot-decimal address format; TLD not allowed

returns true if domain appears to be a proper name and TLD or IPv4 address, else
false

]=]

local function is_domain_name (domain)
        if not domain then
                return false;
-- if not set, abandon
        end

        domain = domain:gsub ('^//', '');
-- strip '//' from domain name if present; done here so we only have to do it
once

        if not domain:match ('^[%w]') then
-- first character must be letter or digit
                return false;
        end

        if domain:match ('^%a+:') then
-- hack to detect things that look like s:Page:Title where Page: is namespace at
Wikisource
                return false;
        end

        local patterns = {
-- patterns that look like URLs
                '%f[%w][%w][%w%-]+[%w]%.%a%a+$',
-- three or more character hostname.hostname or hostname.tld
                '%f[%w][%w][%w%-]+[%w]%.xn%-%-[%w]+$',
-- internationalized domain name with ACE prefix
                '%f[%a][qxz]%.com$',
-- assigned one character .com hostname (x.com times out 2015-12-10)
                '%f[%a][iq]%.net$',
-- assigned one character .net hostname (q.net registered but not active 2015-12-
10)
                '%f[%w][%w]%.%a%a$',
-- one character hostname and ccTLD (2 chars)
                '%f[%w][%w][%w]%.%a%a+$',
-- two character hostname and TLD
                '^%d%d?%d?%.%d%d?%d?%.%d%d?%d?%.%d%d?%d?',
-- IPv4 address
                }

        for _, pattern in ipairs (patterns) do
-- loop through the patterns list
                if domain:match (pattern) then
                        return true;
-- if a match then we think that this thing that purports to be a URL is a URL
                end
        end

        for _, d in ipairs ({'cash', 'company', 'today', 'org'}) do
-- look for single letter second level domain names for these top level domains
```

```
                if domain:match ('%f[%w][%w]%.' .. d) then
                        return true
                end
        end
        return false;
-- no matches, we don't know what this thing is
end


--[[--------------------------< I S _ U R L >--------------------------------
------------------------------

returns true if the scheme and domain parts of a URL appear to be a valid URL;
else false.

This function is the last step in the validation process.  This function is
separate because there are cases that
are not covered by split_url(), for example is_parameter_ext_wikilink() which is
looking for bracketted external
wikilinks.

]]

local function is_url (scheme, domain)
        if utilities.is_set (scheme) then
-- if scheme is set check it and domain
                return is_scheme (scheme) and is_domain_name (domain);
        else
                return is_domain_name (domain);
-- scheme not set when URL is protocol-relative
        end
end


--[[--------------------------< S P L I T _ U R L >--------------------------
------------------------------

Split a URL into a scheme, authority indicator, and domain.

First remove Fully Qualified Domain Name terminator (a dot following TLD) (if
any) and any path(/), query(?) or fragment(#).

If protocol-relative URL, return nil scheme and domain else return nil for both
scheme and domain.

When not protocol-relative, get scheme, authority indicator, and domain.  If
there is an authority indicator (one
or more '/' characters immediately following the scheme's colon), make sure that
there are only 2.

Any URL that does not have news: scheme must have authority indicator (//).
TODO: are there other common schemes
like news: that don't use authority indicator?

Strip off any port and path;

]]

local function split_url (url_str)
        local scheme, authority, domain;

        url_str = url_str:gsub ('([%a%d])%.?[/%?#].*$', '%1');
```

```
-- strip FQDN terminator and path(/), query(?), fragment (#) (the capture
prevents false replacement of '//')

        if url_str:match ('^//%S*') then
-- if there is what appears to be a protocol-relative URL
                domain = url_str:match ('^//(%S*)')
        elseif url_str:match ('%S-:/*%S+') then
-- if there is what appears to be a scheme, optional authority indicator, and
domain name
                scheme, authority, domain = url_str:match ('(%S-:)(/*)(%S+)');
-- extract the scheme, authority indicator, and domain portions
                if utilities.is_set (authority) then
                        authority = authority:gsub ('//', '', 1);
-- replace place 1 pair of '/' with nothing;
                        if utilities.is_set(authority) then
-- if anything left (1 or 3+ '/' where authority should be) then
                                return scheme;
-- return scheme only making domain nil which will cause an error message
                        end
                else
                        if not scheme:match ('^news:') then
-- except for news:..., MediaWiki won't link URLs that do not have authority
indicator; TODO: a better way to do this test?
                                return scheme;
-- return scheme only making domain nil which will cause an error message
                        end
                end
                domain = domain:gsub ('(%a):%d+', '%1');
-- strip port number if present
        end

        return scheme, domain;
end


--[[--------------------------< L I N K _ P A R A M _ O K >----------------------
------------------------------

checks the content of |title-link=, |series-link=, |author-link=, etc. for
properly formatted content: no wikilinks, no URLs

Link parameters are to hold the title of a Wikipedia article, so none of the
WP:TITLESPECIALCHARACTERS are allowed:
        # < > [ ] | { } _
except the underscore which is used as a space in wiki URLs and # which is used
for section links

returns false when the value contains any of these characters.

When there are no illegal characters, this function returns TRUE if value DOES
NOT appear to be a valid URL (the
|<param>-link= parameter is ok); else false when value appears to be a valid URL
(the |<param>-link= parameter is NOT ok).

]]

local function link_param_ok (value)
        local scheme, domain;
        if value:find ('[<>%[%]|{}]') then
-- if any prohibited characters
                return false;
        end
```

```
        scheme, domain = split_url (value);
-- get scheme or nil and domain or nil from URL;
        return not is_url (scheme, domain);
-- return true if value DOES NOT appear to be a valid URL
end


--[[--------------------------< L I N K _ T I T L E _ O K >----------------------
------------------------------

Use link_param_ok() to validate |<param>-link= value and its matching |<title>=
value.

|<title>= may be wiki-linked but not when |<param>-link= has a value.  This
function emits an error message when
that condition exists

check <link> for inter-language interwiki-link prefix.  prefix must be a
MediaWiki-recognized language
code and must begin with a colon.

]]

local function link_title_ok (link, lorig, title, torig)
local orig;
        if utilities.is_set (link) then
-- don't bother if <param>-link doesn't have a value
                if not link_param_ok (link) then
-- check |<param>-link= markup
                        orig = lorig;
-- identify the failing link parameter
                elseif title:find ('%[%[') then
-- check |title= for wikilink markup
                        orig = torig;
-- identify the failing |title= parameter
                elseif link:match ('^%a+:') then
-- if the link is what looks like an interwiki
                        local prefix = link:match ('^(%a+):'):lower();
-- get the interwiki prefix

                        if cfg.inter_wiki_map[prefix] then
-- if prefix is in the map, must have preceding colon
                                orig = lorig;
-- flag as error
                        end
                end
        end

        if utilities.is_set (orig) then
                link = '';
-- unset
                utilities.set_message ('err_bad_paramlink', orig);
-- URL or wikilink in |title= with |title-link=;
        end

        return link;
-- link if ok, empty string else
end


--[[--------------------------< C H E C K _ U R L >------------------------------
```

```
------------------------------

Determines whether a URL string appears to be valid.

First we test for space characters.  If any are found, return false.  Then split
the URL into scheme and domain
portions, or for protocol-relative (//example.com) URLs, just the domain.  Use
is_url() to validate the two
portions of the URL.  If both are valid, or for protocol-relative if domain is
valid, return true, else false.

Because it is different from a standard URL, and because this module used
external_link() to make external links
that work for standard and news: links, we validate newsgroup names here.  The
specification for a newsgroup name
is at https://tools.ietf.org/html/rfc5536#section-3.1.4

]]

local function check_url( url_str )
        if nil == url_str:match ("^%S+$") then
-- if there are any spaces in |url=value it can't be a proper URL
                return false;
        end
        local scheme, domain;

        scheme, domain = split_url (url_str);
-- get scheme or nil and domain or nil from URL;

        if 'news:' == scheme then
-- special case for newsgroups
                return domain:match('^[%a%d%+%-_]+%.[%a%d%+%-_%.]*[%a%d%+%-_]$');
        end

        return is_url (scheme, domain);
-- return true if value appears to be a valid URL
end


--[=[-------------------------< I S _ P A R A M E T E R _ E X T _ W I K I L I N K
>----------------------------

Return true if a parameter value has a string that begins and ends with square
brackets [ and ] and the first
non-space characters following the opening bracket appear to be a URL.  The test
will also find external wikilinks
that use protocol-relative URLs. Also finds bare URLs.

The frontier pattern prevents a match on interwiki-links which are similar to
scheme:path URLs.  The tests that
find bracketed URLs are required because the parameters that call this test
(currently |title=, |chapter=, |work=,
and |publisher=) may have wikilinks and there are articles or redirects like
'//Hus' so, while uncommon, |title=[[//Hus]]
is possible as might be [[en://Hus]].

]=]

local function is_parameter_ext_wikilink (value)
local scheme, domain;

        if value:match ('%f[%[]%[%a%S*:%S+.*%]') then
```

```
-- if ext. wikilink with scheme and domain: [xxxx://yyyyy.zzz]
                scheme, domain = split_url (value:match ('%f[%[]%
[(%a%S*:%S+).*%]'));
        elseif value:match ('%f[%[]%[//%S+.*%]') then
-- if protocol-relative ext. wikilink: [//yyyyy.zzz]
                scheme, domain = split_url (value:match ('%f[%[]%[(//%S+).*%]'));
        elseif value:match ('%a%S*:%S+') then
-- if bare URL with scheme; may have leading or trailing plain text
                scheme, domain = split_url (value:match ('(%a%S*:%S+)'));
        elseif value:match ('//%S+') then
-- if protocol-relative bare URL: //yyyyy.zzz; may have leading or trailing plain
text
                scheme, domain = split_url (value:match ('(//%S+)'));
-- what is left should be the domain
        else
                return false;
-- didn't find anything that is obviously a URL
        end

        return is_url (scheme, domain);
-- return true if value appears to be a valid URL
end


--[[--------------------------< C H E C K _ F O R _ U R L >----------------------
------------------------------

loop through a list of parameters and their values.  Look at the value and if it
has an external link, emit an error message.

]]

local function check_for_url (parameter_list, error_list)
        for k, v in pairs (parameter_list) do
-- for each parameter in the list
                if is_parameter_ext_wikilink (v) then
-- look at the value; if there is a URL add an error message
                        table.insert (error_list, utilities.wrap_style
('parameter', k));
                end
        end
end


--[[--------------------------< S A F E _ F O R _ U R L >------------------------
------------------------------

Escape sequences for content that will be used for URL descriptions

]]

local function safe_for_url( str )
        if str:match( "%[%[.-%]%]" ) ~= nil then
                utilities.set_message ('err_wikilink_in_url', {});
        end

        return str:gsub( '[%[%]\n]', {
                ['['] = '&#91;',
                [']'] = '&#93;',
                ['\n'] = ' ' } );
end
```

```
--[[--------------------------< E X T E R N A L _ L I N K >----------------------
------------------------------

Format an external link with error checking

]]

local function external_link (URL, label, source, access)
	local err_msg = '';
	local domain;
	local path;
	local base_url;

	if not utilities.is_set (label) then
		label = URL;
		if utilities.is_set (source) then
			utilities.set_message ('err_bare_url_missing_title',
{utilities.wrap_style ('parameter', source)});
		else
			error (cfg.messages["bare_url_no_origin"]);
		end
	end
	if not check_url (URL) then
		utilities.set_message ('err_bad_url', {utilities.wrap_style
('parameter', source)});
	end

	domain, path = URL:match ('^([/%.%-%+:%a%d]+)([/%?#].*)$');
-- split the URL into scheme plus domain and path
	if path then
-- if there is a path portion
		path = path:gsub ('[%[%]]', {['['] = '%5b', [']'] = '%5d'});
-- replace '[' and ']' with their percent-encoded values
		URL = table.concat ({domain, path});
-- and reassemble
	end

	base_url = table.concat ({ "[", URL, " ", safe_for_url (label), "]" });
-- assemble a wiki-markup URL

	if utilities.is_set (access) then
-- access level (subscription, registration, limited)
		base_url = utilities.substitute (cfg.presentation['ext-link-
access-signal'], {cfg.presentation[access].class, cfg.presentation[access].title,
base_url});	-- add the appropriate icon
	end

	return base_url;
end


--[[--------------------------< D E P R E C A T E D _ P A R A M E T E R >--------
------------------------------

Categorize and emit an error message when the citation contains one or more
deprecated parameters.  The function includes the
offending parameter name to the error message.  Only one error message is emitted
regardless of the number of deprecated
parameters in the citation.

added_deprecated_cat is a Boolean declared in page scope variables above
```

```
]]

local function deprecated_parameter(name)
        if not added_deprecated_cat then
                added_deprecated_cat = true;
-- note that we've added this category
                utilities.set_message ('err_deprecated_params', {name});
-- add error message
        end
end


--[=[-------------------------< K E R N _ Q U O T E S >--------------------------
------------------------------

Apply kerning to open the space between the quote mark provided by the module and
a leading or trailing quote
mark contained in a |title= or |chapter= parameter's value.

This function will positive kern either single or double quotes:
        "'Unkerned title with leading and trailing single quote marks'"
        " 'Kerned title with leading and trailing single quote marks' " (in real
life the kerning isn't as wide as this example)
Double single quotes (italic or bold wiki-markup) are not kerned.

Replaces Unicode quote marks in plain text or in the label portion of a [[L|D]]
style wikilink with typewriter
quote marks regardless of the need for kerning.  Unicode quote marks are not
replaced in simple [[D]] wikilinks.

Call this function for chapter titles, for website titles, etc.; not for book
titles.

]=]

local function kern_quotes (str)
        local cap = '';
        local wl_type, label, link;

        wl_type, label, link = utilities.is_wikilink (str);
-- wl_type is: 0, no wl (text in label variable); 1, [[D]]; 2, [[L|D]]

        if 1 == wl_type then
-- [[D]] simple wikilink with or without quote marks
                if mw.ustring.match (str, '%[%[[\"“”\'‘’].+[\"“”\'‘’]%]%]') then
-- leading and trailing quote marks
                        str = utilities.substitute (cfg.presentation['kern-
left'], str);
                        str = utilities.substitute (cfg.presentation['kern-
right'], str);
                elseif mw.ustring.match (str, '%[%[[\"“”\'‘’].+%]%]')   then
-- leading quote marks
                        str = utilities.substitute (cfg.presentation['kern-
left'], str);
                elseif mw.ustring.match (str, '%[%[.+[\"“”\'‘’]%]%]') then
-- trailing quote marks
                        str = utilities.substitute (cfg.presentation['kern-
right'], str);
                end

        else
```

```
-- plain text or [[L|D]]; text in label variable
                label = mw.ustring.gsub (label, '[“”]', '\"');
-- replace “” (U+201C & U+201D) with " (typewriter double quote mark)
                label = mw.ustring.gsub (label, '[‘’]', '\'');
-- replace ‘’ (U+2018 & U+2019) with ' (typewriter single quote mark)

                cap = mw.ustring.match (label, "^([\"\'][^\'].+)");
-- match leading double or single quote but not doubled single quotes (italic
markup)
                if utilities.is_set (cap) then
                        label = utilities.substitute (cfg.presentation['kern-
left'], cap);
                end

                cap = mw.ustring.match (label, "^(.+[^\'][\"\'])$")
-- match trailing double or single quote but not doubled single quotes (italic
markup)
                if utilities.is_set (cap) then
                        label = utilities.substitute (cfg.presentation['kern-
right'], cap);
                end

                if 2 == wl_type then
                        str = utilities.make_wikilink (link, label);
-- reassemble the wikilink
                else
                        str = label;
                end
        end
        return str;
end


--[[--------------------------< F O R M A T _ S C R I P T _ V A L U E >----------
------------------------------

|script-title= holds title parameters that are not written in Latin-based
scripts: Chinese, Japanese, Arabic, Hebrew, etc. These scripts should
not be italicized and may be written right-to-left.  The value supplied by
|script-title= is concatenated onto Title after Title has been wrapped
in italic markup.

Regardless of language, all values provided by |script-title= are wrapped in
<bdi>...</bdi> tags to isolate RTL languages from the English left to right.

|script-title= provides a unique feature.  The value in |script-title= may be
prefixed with a two-character ISO 639-1 language code and a colon:
        |script-title=ja:*** *** (where * represents a Japanese character)
Spaces between the two-character code and the colon and the colon and the first
script character are allowed:
        |script-title=ja : *** ***
        |script-title=ja: *** ***
        |script-title=ja :*** ***
Spaces preceding the prefix are allowed: |script-title = ja:*** ***

The prefix is checked for validity.  If it is a valid ISO 639-1 language code,
the lang attribute (lang="ja") is added to the <bdi> tag so that browsers can
know the language the tag contains.  This may help the browser render the script
more correctly.  If the prefix is invalid, the lang attribute
is not added.  At this time there is no error message for this condition.

Supports |script-title=, |script-chapter=, |script-<periodical>=
```

```
]]

local function format_script_value (script_value, script_param)
	local lang='';
-- initialize to empty string
	local name;
	if script_value:match('^%l%l%l?%s*:') then
-- if first 3 or 4 non-space characters are script language prefix
		lang = script_value:match('^(%l%l%l?)%s*:%s*%S.*');
-- get the language prefix or nil if there is no script
		if not utilities.is_set (lang) then
			utilities.set_message ('err_script_parameter',
{script_param, cfg.err_msg_supl['missing title part']});		-- prefix
without 'title'; add error message
			return '';
-- script_value was just the prefix so return empty string
		end

-- if we get this far we have prefix and script
		name = cfg.lang_code_remap[lang] or
mw.language.fetchLanguageName( lang, cfg.this_wiki_code );	-- get language
name so that we can use it to categorize
		if utilities.is_set (name) then
-- is prefix a proper ISO 639-1 language code?
			script_value = script_value:gsub ('^%l+%s*:%s*', '');
-- strip prefix from script

-- is prefix one of these language codes?
			if utilities.in_array (lang, cfg.script_lang_codes) then
				utilities.add_prop_cat ('script', {name, lang})
			else
				utilities.set_message ('err_script_parameter',
{script_param, cfg.err_msg_supl['unknown language code']});	-- unknown
script-language; add error message
			end
			lang = ' lang="' .. lang .. '" ';
-- convert prefix into a lang attribute
		else
			utilities.set_message ('err_script_parameter',
{script_param, cfg.err_msg_supl['invalid language code']});		--
invalid language code; add error message
			lang = '';
-- invalid so set lang to empty string
		end
	else
		utilities.set_message ('err_script_parameter', {script_param,
cfg.err_msg_supl['missing prefix']});				-- no language
code prefix; add error message
	end
	script_value = utilities.substitute (cfg.presentation['bdi'], {lang,
script_value}); -- isolate in case script is RTL

	return script_value;
end


--[[--------------------------< S C R I P T _ C O N C A T E N A T E >------------
------------------------------

Initially for |title= and |script-title=, this function concatenates those two
parameter values after the script
```

```
value has been wrapped in <bdi> tags.

]]

local function script_concatenate (title, script, script_param)
	if utilities.is_set (script) then
		script = format_script_value (script, script_param);
-- <bdi> tags, lang attribute, categorization, etc.; returns empty string on
error
		if utilities.is_set (script) then
			title = title .. ' ' .. script;
-- concatenate title and script title
		end
	end
	return title;
end


--[[--------------------------< W R A P _ M S G >--------------------------------
------------------------------

Applies additional message text to various parameter values. Supplied string is
wrapped using a message_list
configuration taking one argument.  Supports lower case text for {{citation}}
templates.  Additional text taken
from citation_config.messages - the reason this function is similar to but
separate from wrap_style().

]]

local function wrap_msg (key, str, lower)
	if not utilities.is_set ( str ) then
		return "";
	end
	if true == lower then
		local msg;
		msg = cfg.messages[key]:lower();
-- set the message to lower case before
		return utilities.substitute ( msg, str );
-- including template text
	else
		return utilities.substitute ( cfg.messages[key], str );
	end
end


--[[----------------< W I K I S O U R C E _ U R L _ M A K E >-------------------

Makes a Wikisource URL from Wikisource interwiki-link.  Returns the URL and
appropriate
label; nil else.

str is the value assigned to |chapter= (or aliases) or |title= or |title-link=

]]

local function wikisource_url_make (str)
	local wl_type, D, L;
	local ws_url, ws_label;
	local wikisource_prefix = table.concat ({'https://', cfg.this_wiki_code,
'.wikisource.org/wiki/'});
```

```
        wl_type, D, L = utilities.is_wikilink (str);
-- wl_type is 0 (not a wikilink), 1 (simple wikilink), 2 (complex wikilink)

        if 0 == wl_type then
-- not a wikilink; might be from |title-link=
                str = D:match ('^[Ww]ikisource:(.+)') or D:match ('^[Ss]:(.+)');
-- article title from interwiki link with long-form or short-form namespace
                if utilities.is_set (str) then
                        ws_url = table.concat ({
-- build a Wikisource URL
                                wikisource_prefix,
-- prefix
                                str,
-- article title
                                });
                        ws_label = str;
-- label for the URL
                end
        elseif 1 == wl_type then
-- simple wikilink: [[Wikisource:ws article]]
                str = D:match ('^[Ww]ikisource:(.+)') or D:match ('^[Ss]:(.+)');
-- article title from interwiki link with long-form or short-form namespace
                if utilities.is_set (str) then
                        ws_url = table.concat ({
-- build a Wikisource URL
                                wikisource_prefix,
-- prefix
                                str,
-- article title
                                });
                        ws_label = str;
-- label for the URL
                end
        elseif 2 == wl_type then
-- non-so-simple wikilink: [[Wikisource:ws article|displayed text]] ([[L|D]])
                str = L:match ('^[Ww]ikisource:(.+)') or L:match ('^[Ss]:(.+)');
-- article title from interwiki link with long-form or short-form namespace
                if utilities.is_set (str) then
                        ws_label = D;
-- get ws article name from display portion of interwiki link
                        ws_url = table.concat ({
-- build a Wikisource URL
                                wikisource_prefix,
-- prefix
                                str,
-- article title without namespace from link portion of wikilink
                                });
                end
        end

        if ws_url then
                ws_url = mw.uri.encode (ws_url, 'WIKI');
-- make a usable URL
                ws_url = ws_url:gsub ('%%23', '#');
-- undo percent-encoding of fragment marker
        end

        return ws_url, ws_label, L or D;
-- return proper URL or nil and a label or nil
end
```

```
--[[----------------< F O R M A T _ P E R I O D I C A L >----------------------

Format the three periodical parameters: |script-<periodical>=, |<periodical>=,
and |trans-<periodical>= into a single Periodical meta-parameter.

]]

local function format_periodical (script_periodical, script_periodical_source,
periodical, trans_periodical)

	if not utilities.is_set (periodical) then
		periodical = '';
-- to be safe for concatenation
	else
		periodical = utilities.wrap_style ('italic-title', periodical);
-- style
	end

	periodical = script_concatenate (periodical, script_periodical,
script_periodical_source);	-- <bdi> tags, lang attribute, categorization,
etc.; must be done after title is wrapped

	if utilities.is_set (trans_periodical) then
		trans_periodical = utilities.wrap_style ('trans-italic-title',
trans_periodical);
		if utilities.is_set (periodical) then
			periodical = periodical .. ' ' .. trans_periodical;
		else
-- here when trans-periodical without periodical or script-periodical
			periodical = trans_periodical;
			utilities.set_message ('err_trans_missing_title',
{'periodical'});
		end
	end

	return periodical;
end


--[[----------------< F O R M A T _ C H A P T E R _ T I T L E >---------------

Format the four chapter parameters: |script-chapter=, |chapter=, |trans-chapter=,
and |chapter-url= into a single chapter meta- parameter (chapter_url_source used
for error messages).

]]

local function format_chapter_title (script_chapter, script_chapter_source,
chapter, chapter_source, trans_chapter, trans_chapter_source, chapter_url,
chapter_url_source, no_quotes, access)
	local ws_url, ws_label, L = wikisource_url_make (chapter);
-- make a wikisource URL and label from a wikisource interwiki link
	if ws_url then
		ws_label = ws_label:gsub ('_', ' ');
-- replace underscore separators with space characters
		chapter = ws_label;
	end

	if not utilities.is_set (chapter) then
		chapter = '';
-- to be safe for concatenation
	else
```

```
                if false == no_quotes then
                        chapter = kern_quotes (chapter);
-- if necessary, separate chapter title's leading and trailing quote marks from
module provided quote marks
                        chapter = utilities.wrap_style ('quoted-title', chapter);
                end
        end

        chapter = script_concatenate (chapter, script_chapter,
script_chapter_source); -- <bdi> tags, lang attribute, categorization, etc.; must
be done after title is wrapped

        if utilities.is_set (chapter_url) then
                chapter = external_link (chapter_url, chapter,
chapter_url_source, access);    -- adds bare_url_missing_title error if
appropriate
        elseif ws_url then
                chapter = external_link (ws_url, chapter .. ' ', 'ws link in
chapter');      -- adds bare_url_missing_title error if appropriate; space char
to move icon away from chap text; TODO: better way to do this?
                chapter = utilities.substitute (cfg.presentation['interwiki-
icon'], {cfg.presentation['class-wikisource'], L, chapter});
        end

        if utilities.is_set (trans_chapter) then
                trans_chapter = utilities.wrap_style ('trans-quoted-title',
trans_chapter);
                if utilities.is_set (chapter) then
                        chapter = chapter .. ' ' .. trans_chapter;
                else
-- here when trans_chapter without chapter or script-chapter
                        chapter = trans_chapter;
                        chapter_source = trans_chapter_source:match ('trans%-?
(.+)');         -- when no chapter, get matching name from trans-<param>
                        utilities.set_message ('err_trans_missing_title',
{chapter_source});
                end
        end

        return chapter;
end


--[[----------------< H A S _ I N V I S I B L E _ C H A R S >-------------------

This function searches a parameter's value for non-printable or invisible
characters.
The search stops at the first match.

This function will detect the visible replacement character when it is part of
the Wikisource.

Detects but ignores nowiki and math stripmarkers.  Also detects other named
stripmarkers
(gallery, math, pre, ref) and identifies them with a slightly different error
message.
See also coins_cleanup().

Output of this function is an error message that identifies the character or the
Unicode group, or the stripmarker that was detected along with its position (or,
for multi-byte characters, the position of its first byte) in the parameter
value.
```

```
]]

local function has_invisible_chars (param, v)
	local position = '';
-- position of invisible char or starting position of stripmarker
	local capture;
-- used by stripmarker detection to hold name of the stripmarker
	local stripmarker;
-- boolean set true when a stripmarker is found

	capture = string.match (v, '[%w%p ]*');
-- test for values that are simple ASCII text and bypass other tests if true
	if capture == v then
-- if same there are no Unicode characters
		return;
	end

	for _, invisible_char in ipairs (cfg.invisible_chars) do
		local char_name = invisible_char[1];
-- the character or group name
		local pattern = invisible_char[2];
-- the pattern used to find it
		position, _, capture = mw.ustring.find (v, pattern);
-- see if the parameter value contains characters that match the pattern

		if position and (cfg.invisible_defs.zwj == capture) then
-- if we found a zero-width joiner character
			if mw.ustring.find (v, cfg.indic_script) then
-- it's ok if one of the Indic scripts
				position = nil;
-- unset position
			elseif cfg.emoji[mw.ustring.codepoint (v, position+1)]
then			-- is zwj followed by a character listed in emoji{}?
				position = nil;
-- unset position
			end
		end

		if position then
			if 'nowiki' == capture or 'math' == capture or
-- nowiki and math stripmarkers (not an error condition)
				('templatestyles' == capture and
utilities.in_array (param, {'id', 'quote'})) then	-- templatestyles
stripmarker allowed in these parameters
					stripmarker = true;
-- set a flag
			elseif true == stripmarker and cfg.invisible_defs.del ==
capture then	-- because stripmakers begin and end with the delete char, assume
that we've found one end of a stripmarker
				position = nil;
-- unset
			else
				local err_msg;
				if capture and not (cfg.invisible_defs.del ==
capture or cfg.invisible_defs.zwj == capture) then
					err_msg = capture .. ' ' .. char_name;
				else
					err_msg = char_name .. ' ' ..
'character';
				end
```

```
                                utilities.set_message ('err_invisible_char',
{err_msg, utilities.wrap_style ('parameter', param), position});        -- add
error message
                                return;
-- and done with this parameter
                        end
                end
        end
end


--[[--------------------< A R G U M E N T _ W R A P P E R >----------------------

Argument wrapper.  This function provides support for argument mapping defined
in the configuration file so that multiple names can be transparently aliased to
single internal variable.

]]

local function argument_wrapper ( args )
        local origin = {};

        return setmetatable({
                ORIGIN = function ( self, k )
                        local dummy = self[k];
-- force the variable to be loaded.
                        return origin[k];
                end
        },
        {
                __index = function ( tbl, k )
                        if origin[k] ~= nil then
                                return nil;
                        end

                        local args, list, v = args, cfg.aliases[k];

                        if type( list ) == 'table' then
                                v, origin[k] = utilities.select_one ( args, list,
'err_redundant_parameters' );
                                if origin[k] == nil then
                                        origin[k] = '';
-- Empty string, not nil
                                end
                        elseif list ~= nil then
                                v, origin[k] = args[list], list;
                        else
                                -- maybe let through instead of raising an error?
                                -- v, origin[k] = args[k], k;
                                error( cfg.messages['unknown_argument_map'] .. ':
' .. k);
                        end

                        -- Empty strings, not nil;
                        if v == nil then
                                v = '';
                                origin[k] = '';
                        end

                        tbl = rawset( tbl, k, v );
                        return v;
                end,
```

```
        });
end


--[[--------------------------< N O W R A P _ D A T E >--------------------------

When date is YYYY-MM-DD format wrap in nowrap span: <span ...>YYYY-MM-DD</span>.
When date is DD MMMM YYYY or is MMMM DD, YYYY then wrap in nowrap span:
<span ...>DD MMMM</span> YYYY or <span ...>MMMM DD,</span> YYYY

DOES NOT yet support MMMM YYYY or any of the date ranges.

]]

local function nowrap_date (date)
        local cap = '';
        local cap2 = '';

        if date:match("^%d%d%d%d%-%d%d%-%d%d$") then
                date = utilities.substitute (cfg.presentation['nowrap1'], date);

        elseif date:match("^%a+%s*%d%d?,%s+%d%d%d%d$") or date:match ("^%d%d?
%s*%a+%s+%d%d%d%d$") then
                cap, cap2 = string.match (date, "^(.*)%s+(%d%d%d%d)$");
                date = utilities.substitute (cfg.presentation['nowrap2'], {cap,
cap2});
        end

        return date;
end


--[[--------------------------< S E T _ T I T L E T Y P E >----------------------

This function sets default title types (equivalent to the citation including
|type=<default value>) for those templates that have defaults. Also handles the
special case where it is desirable to omit the title type from the rendered
citation
(|type=none).

]]

local function set_titletype (cite_class, title_type)
        if utilities.is_set (title_type) then
                if 'none' == cfg.keywords_xlate[title_type] then
                        title_type = '';
-- if |type=none then type parameter not displayed
                end
                return title_type;
-- if |type= has been set to any other value use that value
        end

        return cfg.title_types [cite_class] or '';
-- set template's default title type; else empty string for concatenation
end


--[[--------------------------< S A F E _ J O I N >------------------------------

Joins a sequence of strings together while checking for duplicate separation
characters.
```

```
]]

local function safe_join( tbl, duplicate_char )
        local f = {};
-- create a function table appropriate to type of 'duplicate character'
                if 1 == #duplicate_char then
-- for single byte ASCII characters use the string library functions
                        f.gsub = string.gsub
                        f.match = string.match
                        f.sub = string.sub
                else
-- for multi-byte characters use the ustring library functions
                        f.gsub = mw.ustring.gsub
                        f.match = mw.ustring.match
                        f.sub = mw.ustring.sub
                end

        local str = '';
-- the output string
        local comp = '';
-- what does 'comp' mean?
        local end_chr = '';
        local trim;
        for _, value in ipairs( tbl ) do
                if value == nil then value = ''; end

                if str == '' then
-- if output string is empty
                        str = value;
-- assign value to it (first time through the loop)
                elseif value ~= '' then
                        if value:sub(1, 1) == '<' then
-- special case of values enclosed in spans and other markup.
                                comp = value:gsub( "%b<>", "" );
-- remove HTML markup (<span>string</span> -> string)
                        else
                                comp = value;
                        end

-- typically duplicate_char is sepc
                        if f.sub(comp, 1, 1) == duplicate_char then
-- is first character same as duplicate_char? why test first character?

--   Because individual string segments often (always?) begin with terminal punct
for the

--   preceding segment: 'First element' .. 'sepc next element' .. etc.?
                                trim = false;
                                end_chr = f.sub(str, -1, -1);
-- get the last character of the output string
                                -- str = str .. "<HERE(enchr=" .. end_chr .. ")"
-- debug stuff?
                                if end_chr == duplicate_char then
-- if same as separator
                                        str = f.sub(str, 1, -2);
-- remove it
                                elseif end_chr == "'" then
-- if it might be wiki-markup
                                        if f.sub(str, -3, -1) == duplicate_char
.. "''" then            -- if last three chars of str are sepc''
                                                str = f.sub(str, 1, -4) .. "''";
-- remove them and add back ''
```

```
                                        elseif  f.sub(str, -5, -1) ==
duplicate_char .. "]]''" then   -- if last five chars of str are sepc]]''
                                                trim = true;
-- why? why do this and next differently from previous?
                                        elseif f.sub(str, -4, -1) ==
duplicate_char .. "]'" then    -- if last four chars of str are sepc]''
                                                trim = true;
-- same question
                                        end
                                elseif end_chr == "]" then
-- if it might be wiki-markup
                                        if f.sub(str, -3, -1) == duplicate_char
.. "]]" then            -- if last three chars of str are sepc]] wikilink
                                                trim = true;
                                        elseif f.sub(str, -3, -1) ==
duplicate_char .. '"]' then     -- if last three chars of str are sepc"] quoted
external link
                                                trim = true;
                                        elseif  f.sub(str, -2, -1) ==
duplicate_char .. "]" then      -- if last two chars of str are sepc] external
link
                                                trim = true;
                                        elseif f.sub(str, -4, -1) ==
duplicate_char .. "'']" then    -- normal case when |url=something &
|title=Title.
                                                trim = true;
                                        end
                                elseif end_chr == " " then
-- if last char of output string is a space
                                        if f.sub(str, -2, -1) == duplicate_char
.. " " then                     -- if last two chars of str are <sepc><space>
                                                str = f.sub(str, 1, -3);
-- remove them both
                                        end
                                end

                                if trim then
                                        if value ~= comp then
-- value does not equal comp when value contains HTML markup
                                                local dup2 = duplicate_char;
                                                if f.match(dup2, "%A" ) then dup2
= "%" .. dup2; end      -- if duplicate_char not a letter then escape it

                                                value = f.gsub(value, "(%b<>)" ..
dup2, "%1", 1 )         -- remove duplicate_char if it follows HTML markup
                                        else
                                                value = f.sub(value, 2, -1 );
-- remove duplicate_char when it is first character
                                        end
                                end
                        end
                        str = str .. value;
-- add it to the output string
                end
        end
        return str;
end


--[[--------------------------< I S _ S U F F I X >--------------------------

returns true if suffix is properly formed Jr, Sr, or ordinal in the range 1–9.
```

```
Puncutation not allowed.

]]

local function is_suffix (suffix)
	if utilities.in_array (suffix, {'Jr', 'Sr', 'Jnr', 'Snr', '1st', '2nd',
'3rd'}) or suffix:match ('^%dth$') then
		return true;
	end
	return false;
end


--[[--------------------< I S _ G O O D _ V A N C _ N A M E >-------------------

For Vancouver style, author/editor names are supposed to be rendered in Latin
(read ASCII) characters.  When a name uses characters that contain diacritical
marks, those characters are to be converted to the corresponding Latin
character. When a name is written using a non-Latin alphabet or logogram, that
name is to be transliterated into Latin characters. The module doesn't do this
so editors may/must.

This test allows |first= and |last= names to contain any of the letters defined
in the four Unicode Latin character sets
	[http://www.unicode.org/charts/PDF/U0000.pdf C0 Controls and Basic Latin]
0041–005A, 0061–007A
	[http://www.unicode.org/charts/PDF/U0080.pdf C1 Controls and Latin-1
Supplement] 00C0–00D6, 00D8–00F6, 00F8–00FF
	[http://www.unicode.org/charts/PDF/U0100.pdf Latin Extended-A] 0100–017F
	[http://www.unicode.org/charts/PDF/U0180.pdf Latin Extended-B] 0180–01BF,
01C4–024F

|lastn= also allowed to contain hyphens, spaces, and apostrophes.
	(http://www.ncbi.nlm.nih.gov/books/NBK7271/box/A35029/)
|firstn= also allowed to contain hyphens, spaces, apostrophes, and periods

This original test:
	if nil == mw.ustring.find (last, "^[A-Za-zÀ-ÖØ-öø-pǄ-ƴ%-%s%']*$")
	or nil == mw.ustring.find (first, "^[A-Za-zÀ-ÖØ-öø-pǄ-ƴ%-%s%'%.]+[2-
6%a]*$") then
was written outside of the code editor and pasted here because the code editor
gets confused between character insertion point and cursor position. The test has
been rewritten to use decimal character escape sequence for the individual bytes
of the Unicode characters so that it is not necessary to use an external editor
to maintain this code.

	\195\128-\195\150 – À-Ö (U+00C0–U+00D6 – C0 controls)
	\195\152-\195\182 – Ø-ö (U+00D8-U+00F6 – C0 controls)
	\195\184-\198\191 – ø-ƿ (U+00F8-U+01BF – C0 controls, Latin extended A &
B)
	\199\132-\201\143 – Ǆ-ƴ (U+01C4-U+024F – Latin extended B)

]]

local function is_good_vanc_name (last, first, suffix, position)
	if not suffix then
		if first:find ('[,%s]') then
-- when there is a space or comma, might be first name/initials + generational
suffix
			first = first:match ('(.-)[,%s]+');
-- get name/initials
			suffix = first:match ('[,%s]+(.+)$');
```

```
-- get generational suffix
                end
        end
        if utilities.is_set (suffix) then
                if not is_suffix (suffix) then
                        add_vanc_error (cfg.err_msg_supl.suffix, position);
                        return false;
-- not a name with an appropriate suffix
                end
        end
        if nil == mw.ustring.find (last, "^[A-Za-z\195\128-\195\150\195\152-
\195\182\195\184-\198\191\199\132-\201\143%-%s%']*$") or
                nil == mw.ustring.find (first, "^[A-Za-z\195\128-
\195\150\195\152-\195\182\195\184-\198\191\199\132-\201\143%-%s%'%.]*$") then
                        add_vanc_error (cfg.err_msg_supl['non-Latin char'],
position);
                        return false;
-- not a string of Latin characters; Vancouver requires Romanization
        end;
        return true;
end


--[[--------------------------< R E D U C E _ T O _ I N I T I A L S >------------
------------------------------

Attempts to convert names to initials in support of |name-list-style=vanc.

Names in |firstn= may be separated by spaces or hyphens, or for initials, a
period.
See http://www.ncbi.nlm.nih.gov/books/NBK7271/box/A35062/.

Vancouver style requires family rank designations (Jr, II, III, etc.) to be
rendered
as Jr, 2nd, 3rd, etc.  See http://www.ncbi.nlm.nih.gov/books/NBK7271/box/A35085/.
This code only accepts and understands generational suffix in the Vancouver
format
because Roman numerals look like, and can be mistaken for, initials.

This function uses ustring functions because firstname initials may be any of the
Unicode Latin characters accepted by is_good_vanc_name ().

]]

local function reduce_to_initials(first, position)
        local name, suffix = mw.ustring.match(first, "^(%u+) ([%dJS]
[%drndth]+)$");

        if not name then
-- if not initials and a suffix
                name = mw.ustring.match(first, "^(%u+)$");
-- is it just initials?
        end

        if name then
-- if first is initials with or without suffix
                if 3 > mw.ustring.len (name) then
-- if one or two initials
                        if suffix then
-- if there is a suffix
                                if is_suffix (suffix) then
-- is it legitimate?
```

```
                                        return first;
-- one or two initials and a valid suffix so nothing to do
                                else
                                        add_vanc_error (cfg.err_msg_supl.suffix,
position);                      -- one or two initials with invalid suffix so
error message
                                        return first;
-- and return first unmolested
                                end
                        else
                                return first;
-- one or two initials without suffix; nothing to do
                        end
                end
        end
-- if here then name has 3 or more uppercase letters so treat them as a word

        local initials, names = {}, {};
-- tables to hold name parts and initials
        local i = 1;
-- counter for number of initials

        names = mw.text.split (first, '[%s,]+');
-- split into a table of names and possible suffix

        while names[i] do
-- loop through the table
                if 1 < i and names[i]:match ('[%dJS][%drndth]+%.?$') then
-- if not the first name, and looks like a suffix (may have trailing dot)
                        names[i] = names[i]:gsub ('%.', '');
-- remove terminal dot if present
                        if is_suffix (names[i]) then
-- if a legitimate suffix
                                table.insert (initials, ' ' .. names[i]);
-- add a separator space, insert at end of initials table
                                break;
-- and done because suffix must fall at the end of a name
                        end
-- no error message if not a suffix; possibly because of Romanization
                end
                if 3 > i then
                        table.insert (initials, mw.ustring.sub(names[i], 1, 1));
-- insert the initial at end of initials table
                end
                i = i + 1;
-- bump the counter
        end

        return table.concat(initials)
-- Vancouver format does not include spaces.
end


--[[-------------------------< L I S T _ P E O P L E >-------------------------

Formats a list of people (authors, contributors, editors, interviewers,
translators)

names in the list will be linked when
        |<name>-link= has a value
        |<name>-mask= does NOT have a value; masked names are presumed to have
been
```

```
                rendered previously so should have been linked there

when |<name>-mask=0, the associated name is not rendered

]]

local function list_people (control, people, etal)
        local sep;
        local namesep;
        local format = control.format;
        local maximum = control.maximum;
        local name_list = {};

        if 'vanc' == format then
-- Vancouver-like name styling?
                sep = cfg.presentation['sep_nl_vanc'];
-- name-list separator between names is a comma
                namesep = cfg.presentation['sep_name_vanc'];
-- last/first separator is a space
        else
                sep = cfg.presentation['sep_nl'];
-- name-list separator between names is a semicolon
                namesep = cfg.presentation['sep_name'];
-- last/first separator is <comma><space>
        end

        if sep:sub (-1, -1) ~= " " then sep = sep .. " " end
        if utilities.is_set (maximum) and maximum < 1 then return "", 0; end
-- returned 0 is for EditorCount; not used for other names

        for i, person in ipairs (people) do
                if utilities.is_set (person.last) then
                        local mask = person.mask;
                        local one;
                        local sep_one = sep;

                        if utilities.is_set (maximum) and i > maximum then
                                etal = true;
                                break;
                        end

                        if mask then
                                local n = tonumber (mask);
-- convert to a number if it can be converted; nil else
                                if n then
                                        one = 0 ~= n and string.rep("&mdash;", n)
or nil;                 -- make a string of (n > 0) mdashes, nil else, to replace
name
                                        person.link = nil;
-- don't create link to name if name is replaces with mdash string or has been
set nil
                                else
                                        one = mask;
-- replace name with mask text (must include name-list separator)
                                        sep_one = " ";
-- modify name-list separator
                                end
                        else
                                one = person.last;
-- get surname
                                local first = person.first
-- get given name
```

```
                                if utilities.is_set (first) then
                                        if ("vanc" == format) then
-- if Vancouver format
                                                one = one:gsub ('%.', '');
-- remove periods from surnames
(http://www.ncbi.nlm.nih.gov/books/NBK7271/box/A35029/)
                                                if not person.corporate and
is_good_vanc_name (one, first, nil, i) then             -- and name is all Latin
characters; corporate authors not tested
                                                        first =
reduce_to_initials (first, i);                          -- attempt to convert
first name(s) to initials
                                                end
                                        end
                                        one = one .. namesep .. first;
                                end
                        end
                        if utilities.is_set (person.link) then
                                one = utilities.make_wikilink (person.link, one);
-- link author/editor
                        end
                        if one then
-- if <one> has a value (name, mdash replacement, or mask text replacement)
                                table.insert (name_list, one);
-- add it to the list of names
                                table.insert (name_list, sep_one);
-- add the proper name-list separator
                        end
                end
        end

        local count = #name_list / 2;
-- (number of names + number of separators) divided by 2
        if 0 < count then
                if 1 < count and not etal then
                        if 'amp' == format then
                                name_list[#name_list-2] = " & ";
-- replace last separator with ampersand text
                        elseif 'and' == format then
                                if 2 == count then
                                        name_list[#name_list-2] =
cfg.presentation.sep_nl_and;            -- replace last separator with 'and' text
                                else
                                        name_list[#name_list-2] =
cfg.presentation.sep_nl_end;            -- replace last separator with '(sep)
and' text
                                end
                        end
                end
                name_list[#name_list] = nil;
-- erase the last separator
        end

        local result = table.concat (name_list);
-- construct list
        if etal and utilities.is_set (result) then
-- etal may be set by |display-authors=etal but we might not have a last-first
list
                result = result .. sep .. ' ' .. cfg.messages['et al'];
-- we've got a last-first list and etal so add et al.
        end
```

```
        return result, count;
-- return name-list string and count of number of names (count used for editor
names only)
end


--[[--------------------< M A K E _ C I T E R E F _ I D >----------------------

Generates a CITEREF anchor ID if we have at least one name or a date.  Otherwise
returns an empty string.

namelist is one of the contributor-, author-, or editor-name lists chosen in that
order.  year is Year or anchor_year.

]]

local function make_citeref_id (namelist, year)
        local names={};                                                 -- a
table for the one to four names and year
        for i,v in ipairs (namelist) do                 -- loop through the list
and take up to the first four last names
                names[i] = v.last
                if i == 4 then break end                        -- if four then
done
        end
        table.insert (names, year);                             -- add the year
at the end
        local id = table.concat(names);                 -- concatenate names and
year for CITEREF id
        if utilities.is_set (id) then                   -- if concatenation is
not an empty string
                return "CITEREF" .. id;                         -- add the
CITEREF portion
        else
                return '';                                                  -
- return an empty string; no reason to include CITEREF id in this citation
        end
end


--[[--------------------------< C I T E _ C L A S S _A T T R I B U T E _M A K E
>------------------------------

construct <cite> tag class attribute for this citation.

<cite_class> – config.CitationClass from calling template
<mode> – value from |mode= parameter

]]

local function cite_class_attribute_make (cite_class, mode)
        local class_t = {};
        table.insert (class_t, 'citation');
-- required for blue highlight
        if 'citation' ~= cite_class then
                table.insert (class_t, cite_class);
-- identify this template for user css
                table.insert (class_t, utilities.is_set (mode) and mode or
'cs1');         -- identify the citation style for user css or javascript
        else
                table.insert (class_t, utilities.is_set (mode) and mode or
'cs2');         -- identify the citation style for user css or javascript
```

```
        end
        for _, prop_key in ipairs (z.prop_keys_t) do
                table.insert (class_t, prop_key);
-- identify various properties for user css or javascript
        end

        return table.concat (class_t, ' ');
-- make a big string and done
end


--[[---------------------< N A M E _ H A S _ E T A L >--------------------------

Evaluates the content of name parameters (author, editor, etc.) for variations on
the theme of et al.  If found, the et al. is removed, a flag is set to true and
the function returns the modified name and the flag.

This function never sets the flag to false but returns its previous state because
it may have been set by previous passes through this function or by the
associated
|display-<names>=etal parameter

]]

local function name_has_etal (name, etal, nocat, param)

        if utilities.is_set (name) then
-- name can be nil in which case just return
                local patterns = cfg.et_al_patterns;
-- get patterns from configuration

                for _, pattern in ipairs (patterns) do
-- loop through all of the patterns
                        if name:match (pattern) then
-- if this 'et al' pattern is found in name
                                name = name:gsub (pattern, '');
-- remove the offending text
                                etal = true;
-- set flag (may have been set previously here or by |display-<names>=etal)
                                if not nocat then
-- no categorization for |vauthors=
                                        utilities.set_message ('err_etal',
{param});                               -- and set an error if not added
                                end
                        end
                end
        end

        return name, etal;
end


--[[---------------------< N A M E _ I S _ N U M E R I C >----------------------

Add maint cat when name parameter value does not contain letters.  Does not catch
mixed alphanumeric names so |last=A. Green (1922-1987) does not get caught in the
current version of this test but |first=(1888) is caught.

returns nothing

]]
```

```
local function name_is_numeric (name, list_name)
	if utilities.is_set (name) then
		if mw.ustring.match (name, '^[%A]+$') then
-- when name does not contain any letters
			utilities.set_message ('maint_numeric_names',
cfg.special_case_translation [list_name]);	-- add a maint cat for this
template
		end
	end
end


--[[-----------------< N A M E _ H A S _ M U L T _ N A M E S >------------------

Evaluates the content of last/surname (authors etc.) parameters for multiple
names.
Multiple names are indicated if there is more than one comma or any "unescaped"
semicolons. Escaped semicolons are ones used as part of selected HTML entities.
If the condition is met, the function adds the multiple name maintenance
category.

returns nothing

]]

local function name_has_mult_names (name, list_name)
	local _, commas, semicolons, nbsps;
	if utilities.is_set (name) then
		_, commas = name:gsub (',', '');
-- count the number of commas
		_, semicolons = name:gsub (';', '');
-- count the number of semicolons
		-- nbsps probably should be its own separate count rather than
merged in
		-- some way with semicolons because Lua patterns do not support
the
		-- grouping operator that regex does, which means there is no way
to add
		-- more entities to escape except by adding more counts with the
new
		-- entities
		_, nbsps = name:gsub (' ','');
-- count nbsps

		-- There is exactly 1 semicolon per   entity, so subtract
nbsps
		-- from semicolons to 'escape' them. If additional entities are
added,
		-- they also can be subtracted.
		if 1 < commas or 0 < (semicolons - nbsps) then
			utilities.set_message ('maint_mult_names',
cfg.special_case_translation [list_name]);	-- add a maint message
		end
	end
end


--[=[-------------------------< I S _ G E N E R I C >--------------------------
------------------------------

Compares values assigned to various parameters according to the string provided
as <item> in the function call.
```

```
<item> can have on of two values:
        'generic_names' - for name-holding parameters: |last=, |first=, |editor-
last=, etc
        'generic_titles' - for |title=

There are two types of generic tests.  The 'accept' tests look for a pattern that
should not be rejected by the
'reject' test.  For example,
        |author=[[John Smith (author)|Smith, John]]
would be rejected by the 'author' reject test.  But piped wikilinks with 'author'
disambiguation should not be
rejected so the 'accept' test prevents that from happening.  Accept tests are
always performed before reject
tests.

Each of the 'accept' and 'reject' sequence tables hold tables for en.wiki
(['en']) and local.wiki (['local'])
that each can hold a test sequence table  The sequence table holds, at index [1],
a test pattern, and, at index
[2], a boolean control value.  The control value tells string.find() or
mw.ustring.find() to do plain-text search (true)
or a pattern search (false).  The intent of all this complexity is to make these
searches as fast as possible so
that we don't run out of processing time on very large articles.

Returns
        true when a reject test finds the pattern or string
        false when an accept test finds the pattern or string
        nil else

]=]

local function is_generic (item, value, wiki)
        local test_val;
        local str_lower = {
-- use string.lower() for en.wiki (['en']) and use mw.ustring.lower() or
local.wiki (['local'])
                ['en'] = string.lower,
                ['local'] = mw.ustring.lower,
                }
        local str_find = {
-- use string.find() for en.wiki (['en']) and use mw.ustring.find() or local.wiki
(['local'])
                ['en'] = string.find,
                ['local'] = mw.ustring.find,
                }

        local function test (val, test_t, wiki)
-- local function to do the testing; <wiki> selects lower() and find() functions
                val = test_t[2] and str_lower[wiki](value) or val;
-- when <test_t[2]> set to 'true', plaintext search using lowercase value
                return str_find[wiki] (val, test_t[1], 1, test_t[2]);
-- return nil when not found or matched
        end

        local test_types_t = {'accept', 'reject'};
-- test accept patterns first, then reject patterns
        local wikis_t = {'en', 'local'};
-- do tests for each of these keys; en.wiki first, local.wiki second

        for _, test_type in ipairs (test_types_t) do
-- for each test type
```

```
                for _, generic_value in pairs (cfg.special_case_translation[item]
[test_type]) do -- spin through the list of generic value fragments to accept or
reject
                        for _, wiki in ipairs (wikis_t) do
                                if generic_value[wiki] then
                                        if test (value, generic_value[wiki],
wiki) then                              -- go do the test
                                                return ('reject' == test_type);
-- param value rejected, return true; false else
                                        end
                                end
                        end
                end
        end
end


--[[--------------------------< N A M E _ I S _ G E N E R I C >------------------
------------------------------

calls is_generic() to determine if <name> is a 'generic name' listed in
cfg.generic_names; <name_alias> is the
parameter name used in error messaging

]]

local function name_is_generic (name, name_alias)
        if not added_generic_name_errs  and is_generic ('generic_names', name)
then
                utilities.set_message ('err_generic_name', name_alias);
-- set an error message
                added_generic_name_errs = true;
        end
end


--[[--------------------------< N A M E _ C H E C K S >--------------------------
------------------------------

This function calls various name checking functions used to validate the content
of the various name-holding parameters.

]]

local function name_checks (last, first, list_name, last_alias, first_alias)
        local accept_name;

        if utilities.is_set (last) then
                last, accept_name = utilities.has_accept_as_written (last);
-- remove accept-this-as-written markup when it wraps all of <last>

                if not accept_name then
-- <last> not wrapped in accept-as-written markup
                        name_has_mult_names (last, list_name);
-- check for multiple names in the parameter (last only)
                        name_is_numeric (last, list_name);
-- check for names that are composed of digits and punctuation
                        name_is_generic (last, last_alias);
-- check for names found in the generic names list
                end
        end
```

```
        if utilities.is_set (first) then
                first, accept_name = utilities.has_accept_as_written (first);
-- remove accept-this-as-written markup when it wraps all of <first>

                if not accept_name then
-- <first> not wrapped in accept-as-written markup
                        name_is_numeric (first, list_name);
-- check for names that are composed of digits and punctuation
                        name_is_generic (first, first_alias);
-- check for names found in the generic names list
                end
                local wl_type, D = utilities.is_wikilink (first);
                if 0 ~= wl_type then
                        first = D;
                        utilities.set_message ('err_bad_paramlink', first_alias);
                end
        end

        return last, first;
-- done
end


--[[----------------------< E X T R A C T _ N A M E S >-----------------------

Gets name list from the input arguments

Searches through args in sequential order to find |lastn= and |firstn= parameters
(or their aliases), and their matching link and mask parameters. Stops searching
when both |lastn= and |firstn= are not found in args after two sequential
attempts:
found |last1=, |last2=, and |last3= but doesn't find |last4= and |last5= then the
search is done.

This function emits an error message when there is a |firstn= without a matching
|lastn=.  When there are 'holes' in the list of last names, |last1= and |last3=
are present but |last2= is missing, an error message is emitted. |lastn= is not
required to have a matching |firstn=.

When an author or editor parameter contains some form of 'et al.', the 'et al.'
is stripped from the parameter and a flag (etal) returned that will cause
list_people()
to add the static 'et al.' text from Module:Citation/CS1/Configuration.  This
keeps
'et al.' out of the template's metadata.  When this occurs, an error is emitted.

]]

local function extract_names(args, list_name)
        local names = {};
-- table of names
        local last;
-- individual name components
        local first;
        local link;
        local mask;
        local i = 1;
-- loop counter/indexer
        local n = 1;
-- output table indexer
        local count = 0;
-- used to count the number of times we haven't found a |last= (or alias for
```

```
authors, |editor-last or alias for editors)
        local etal = false;
-- return value set to true when we find some form of et al. in an author
parameter

        local last_alias, first_alias, link_alias;
-- selected parameter aliases used in error messaging
        while true do
                last, last_alias = utilities.select_one ( args,
cfg.aliases[list_name .. '-Last'], 'err_redundant_parameters', i );             -
- search through args for name components beginning at 1
                first, first_alias = utilities.select_one ( args,
cfg.aliases[list_name .. '-First'], 'err_redundant_parameters', i );
                link, link_alias = utilities.select_one ( args,
cfg.aliases[list_name .. '-Link'], 'err_redundant_parameters', i );
                mask = utilities.select_one ( args, cfg.aliases[list_name .. '-
Mask'], 'err_redundant_parameters', i );

                last, etal = name_has_etal (last, etal, false, last_alias);
-- find and remove variations on et al.
                first, etal = name_has_etal (first, etal, false, first_alias);
-- find and remove variations on et al.
                last, first = name_checks (last, first, list_name, last_alias,
first_alias);                                           -- multiple names,
extraneous annotation, etc. checks

                if first and not last then
-- if there is a firstn without a matching lastn
                        local alias = first_alias:find ('given', 1, true) and
'given' or 'first';     -- get first or given form of the alias
                        utilities.set_message ('err_first_missing_last', {
                                first_alias,
-- param name of alias missing its mate
                                first_alias:gsub (alias, {['first'] = 'last',
['given'] = 'surname'}),        -- make param name appropriate to the alias form
                                });
-- add this error message
                elseif not first and not last then
-- if both firstn and lastn aren't found, are we done?
                        count = count + 1;
-- number of times we haven't found last and first
                        if 2 <= count then
-- two missing names and we give up
                                break;
-- normal exit or there is a two-name hole in the list; can't tell which
                        end
                else
-- we have last with or without a first
                        local result;
                        link = link_title_ok (link, link_alias, last,
last_alias);                    -- check for improper wiki-markup

                        if first then
                                link = link_title_ok (link, link_alias, first,
first_alias);   -- check for improper wiki-markup
                        end

                        names[n] = {last = last, first = first, link = link, mask
= mask, corporate = false};     -- add this name to our names list (corporate for
|vauthors= only)
                        n = n + 1;
-- point to next location in the names table
```

```
                        if 1 == count then
-- if the previous name was missing
                                utilities.set_message ('err_missing_name',
{list_name:match ("(%w+)List"):lower(), i - 1});        -- add this error message
                        end
                        count = 0;
-- reset the counter, we're looking for two consecutive missing names
                end
                i = i + 1;
-- point to next args location
        end

        return names, etal;
-- all done, return our list of names and the etal flag
end


--[[--------------------------< N A M E _ T A G _ G E T >------------------------
------------------------------

attempt to decode |language=<lang_param> and return language name and matching
tag; nil else.

This function looks for:
        <lang_param> as a tag in cfg.lang_code_remap{}
        <lang_param> as a name in cfg.lang_name_remap{}

        <lang_param> as a name in cfg.mw_languages_by_name_t
        <lang_param> as a tag in cfg.mw_languages_by_tag_t
when those fail, presume that <lang_param> is an IETF-like tag that MediaWiki
does not recognize.  Strip all
script, region, variant, whatever subtags from <lang_param> to leave just a two
or three character language tag
and look for the new <lang_param> in cfg.mw_languages_by_tag_t{}

on success, returns name (in properly capitalized form) and matching tag (in
lowercase); on failure returns nil

]]

local function name_tag_get (lang_param)
        local lang_param_lc = mw.ustring.lower (lang_param);
-- use lowercase as an index into the various tables
        local name;
        local tag;

        name = cfg.lang_code_remap[lang_param_lc];
-- assume <lang_param_lc> is a tag; attempt to get remapped language name
        if name then
-- when <name>, <lang_param> is a tag for a remapped language name
                return name, lang_param_lc;
-- so return <name> from remap and <lang_param_lc>
        end

        tag = lang_param_lc:match ('^(%a%a%a?)%-.*');
-- still assuming that <lang_param_lc> is a tag; strip script, region, variant
subtags
        name = cfg.lang_code_remap[tag];
-- attempt to get remapped language name with language subtag only
        if name then
-- when <name>, <tag> is a tag for a remapped language name
                return name, tag;
```

```
-- so return <name> from remap and <tag>
        end

        if cfg.lang_name_remap[lang_param_lc] then
-- not a tag, assume <lang_param_lc> is a name; attempt to get remapped language
tag
                return cfg.lang_name_remap[lang_param_lc][1],
cfg.lang_name_remap[lang_param_lc][2];  -- for this <lang_param_lc>, return a
(possibly) new name and appropriate tag
        end

        tag = cfg.mw_languages_by_name_t[lang_param_lc];
-- assume that <lang_param_lc> is a language name; attempt to get its matching
tag

        if tag then
                return cfg.mw_languages_by_tag_t[tag], tag;
-- <lang_param_lc> is a name so return the name from the table and <tag>
        end

        name = cfg.mw_languages_by_tag_t[lang_param_lc];
-- assume that <lang_param_lc> is a tag; attempt to get its matching language
name

        if name then
                return name, lang_param_lc;
-- <lang_param_lc> is a tag so return it and <name>
        end

        tag = lang_param_lc:match ('^(%a%a%a?)%-.*');
-- is <lang_param_lc> an IETF-like tag that MediaWiki doesn't recognize? <tag>
gets the language subtag; nil else

        if tag then
                name = cfg.mw_languages_by_tag_t[tag];
-- attempt to get a language name using the shortened <tag>
                if name then
                        return name, tag;
-- <lang_param_lc> is an unrecognized IETF-like tag so return <name> and language
subtag
                end
        end
end


--[[--------------------< L A N G U A G E _ P A R A M E T E R >-------------------

Gets language name from a provided two- or three-character ISO 639 code.  If a
code
is recognized by MediaWiki, use the returned name; if not, then use the value
that
was provided with the language parameter.

When |language= contains a recognized language (either code or name), the page is
assigned to the category for that code: Category:Norwegian-language sources (no).
For valid three-character code languages, the page is assigned to the single
category
for '639-2' codes: Category:CS1 ISO 639-2 language sources.

Languages that are the same as the local wiki are not categorized.  MediaWiki
does
not recognize three-character equivalents of two-character codes: code 'ar' is
```

```
recognized but code 'ara' is not.

This function supports multiple languages in the form |language=nb, French, th
where the language names or codes are separated from each other by commas with
optional space characters.

]]

local function language_parameter (lang)
	local tag;
-- some form of IETF-like language tag; language subtag with optional region,
sript, vatiant, etc subtags
	local lang_subtag;
-- ve populates |language= with mostly unecessary region subtags the MediaWiki
does not recognize; this is the base language subtag
	local name;
-- the language name
	local language_list = {};
-- table of language names to be rendered
	local names_t = {};
-- table made from the value assigned to |language=

	local this_wiki_name = mw.language.fetchLanguageName (cfg.this_wiki_code,
cfg.this_wiki_code);	-- get this wiki's language name

	names_t = mw.text.split (lang, '%s*,%s*');
-- names should be a comma separated list

	for _, lang in ipairs (names_t) do
-- reuse lang here because we don't yet know if lang is a language name or a
language tag
		name, tag = name_tag_get (lang);
-- attempt to get name/tag pair for <lang>; <name> has proper capitalization;
<tag> is lowercase

		if utilities.is_set (tag) then
			lang_subtag = tag:gsub ('^(%a%a%a?)%-.*', '%1');
-- for categorization, strip any IETF-like tags from language tag

			if cfg.this_wiki_code ~= lang_subtag then
-- when the language is not the same as this wiki's language
				if 2 == lang_subtag:len() then
-- and is a two-character tag
--					utilities.add_prop_cat ('foreign-lang-
source', {name, lang_subtag}, lang_subtag);		-- categorize it; tag
appended to allow for multiple language categorization
					utilities.add_prop_cat ('foreign-lang-
source', {name, tag}, lang_subtag);		-- categorize it; tag appended to
allow for multiple language categorization
				else
-- or is a recognized language (but has a three-character tag)
					utilities.add_prop_cat ('foreign-lang-
source-2', {lang_subtag}, lang_subtag);		-- categorize it
differently TODO: support multiple three-character tag categories per cs1|2
template?
				end
			elseif cfg.local_lang_cat_enable then
-- when the language and this wiki's language are the same and categorization is
enabled
				utilities.add_prop_cat ('local-lang-source',
{name, lang_subtag});		-- categorize it
			end
```

```
                else
                        name = lang;
-- return whatever <lang> has so that we show something
                        utilities.set_message ('maint_unknown_lang');
-- add maint category if not already added
                end

                table.insert (language_list, name);
                name = '';
-- so we can reuse it
        end

        name = utilities.make_sep_list (#language_list, language_list);
        if (1 == #language_list) and (lang_subtag == cfg.this_wiki_code) then
-- when only one language, find lang name in this wiki lang name; for
|language=en-us, 'English' in 'American English'
                return '';
-- if one language and that language is this wiki's return an empty string (no
annotation)
        end
        return (" " .. wrap_msg ('language', name));
-- otherwise wrap with '(in ...)'
        --[[ TODO: should only return blank or name rather than full list
        so we can clean up the bunched parenthetical elements Language, Type,
Format
        ]]
end


--[[------------------------< S E T _ C S _ S T Y L E >--------------------------

Gets the default CS style configuration for the given mode.
Returns default separator and either postscript as passed in or the default.
In CS1, the default postscript and separator are '.'.
In CS2, the default postscript is the empty string and the default separator is
','.

]]

local function set_cs_style (postscript, mode)
        if utilities.is_set(postscript) then
                -- emit a maintenance message if user postscript is the default
cs1 postscript
                -- we catch the opposite case for cs2 in set_style
                if mode == 'cs1' and postscript == cfg.presentation['ps_' ..
mode] then
                        utilities.set_message ('maint_postscript');
                end
        else
                postscript = cfg.presentation['ps_' .. mode];
        end
        return cfg.presentation['sep_' .. mode], postscript;
end

--[[--------------------------< S E T _ S T Y L E >------------------------------

Sets the separator and postscript styles. Checks the |mode= first and the
#invoke CitationClass second. Removes the postscript if postscript == none.

]]

local function set_style (mode, postscript, cite_class)
```

```
        local sep;
        if 'cs2' == mode then
                sep, postscript = set_cs_style (postscript, 'cs2');
        elseif 'cs1' == mode then
                sep, postscript = set_cs_style (postscript, 'cs1');
        elseif 'citation' == cite_class then
                sep, postscript = set_cs_style (postscript, 'cs2');
        else
                sep, postscript = set_cs_style (postscript, 'cs1');
        end

        if cfg.keywords_xlate[postscript:lower()] == 'none' then
                -- emit a maintenance message if user postscript is the default
cs2 postscript
                -- we catch the opposite case for cs1 in set_cs_style
                if 'cs2' == mode or 'citation' == cite_class then
                        utilities.set_message ('maint_postscript');
                end
                postscript = '';
        end

        return sep, postscript
end


--[=[-------------------------< I S _ P D F >--------------------------------

Determines if a URL has the file extension that is one of the PDF file extensions
used by [[MediaWiki:Common.css]] when applying the PDF icon to external links.

returns true if file extension is one of the recognized extensions, else false

]=]

local function is_pdf (url)
        return url:match ('%.pdf$') or url:match ('%.PDF$') or
                url:match ('%.pdf[%?#]') or url:match ('%.PDF[%?#]') or
                url:match ('%.PDF&#035') or url:match ('%.pdf&#035');
end


--[[-------------------------< S T Y L E _ F O R M A T >----------------------

Applies CSS style to |format=, |chapter-format=, etc.  Also emits an error
message
if the format parameter does not have a matching URL parameter.  If the format
parameter
is not set and the URL contains a file extension that is recognized as a PDF
document
by MediaWiki's commons.css, this code will set the format parameter to (PDF) with
the appropriate styling.

]]

local function style_format (format, url, fmt_param, url_param)
        if utilities.is_set (format) then
                format = utilities.wrap_style ('format', format);
-- add leading space, parentheses, resize
                if not utilities.is_set (url) then
                        utilities.set_message ('err_format_missing_url',
{fmt_param, url_param});        -- add an error message
                end
```

```
        elseif is_pdf (url) then
-- format is not set so if URL is a PDF file then
                format = utilities.wrap_style ('format', 'PDF');
-- set format to PDF
        else
                format = '';
-- empty string for concatenation
        end
        return format;
end


--[[---------------------< G E T _ D I S P L A Y _ N A M E S >------------------

Returns a number that defines the number of names displayed for author and editor
name lists and a Boolean flag to indicate when et al. should be appended to the
name list.

When the value assigned to |display-xxxxors= is a number greater than or equal to
zero,
return the number and the previous state of the 'etal' flag (false by default
but may have been set to true if the name list contains some variant of the text
'et al.').

When the value assigned to |display-xxxxors= is the keyword 'etal', return a
number
that is one greater than the number of authors in the list and set the 'etal'
flag true.
This will cause the list_people() to display all of the names in the name list
followed by 'et al.'

In all other cases, returns nil and the previous state of the 'etal' flag.

inputs:
        max: A['DisplayAuthors'] or A['DisplayEditors']; a number or some flavor
of etal
        count: #a or #e
        list_name: 'authors' or 'editors'
        etal: author_etal or editor_etal

]]

local function get_display_names (max, count, list_name, etal, param)
        if utilities.is_set (max) then
                if 'etal' == max:lower():gsub("[ '%.]", '') then
-- the :gsub() portion makes 'etal' from a variety of 'et al.' spellings and
stylings
                        max = count + 1;
-- number of authors + 1 so display all author name plus et al.
                        etal = true;
-- overrides value set by extract_names()
                elseif max:match ('^%d+$') then
-- if is a string of numbers
                        max = tonumber (max);
-- make it a number
                        if max >= count then
-- if |display-xxxxors= value greater than or equal to number of authors/editors
                                utilities.set_message ('err_disp_name', {param,
max});                  -- add error message
                                max = nil;
                        end
                else
```

```
-- not a valid keyword or number
                        utilities.set_message ('err_disp_name', {param, max});
-- add error message
                        max = nil;
-- unset; as if |display-xxxxors= had not been set
                end
        end

        return max, etal;
end


--[[----------< E X T R A _ T E X T _ I N _ P A G E _ C H E C K >---------------

Adds error if |page=, |pages=, |quote-page=, |quote-pages= has what appears to be
some form of p. or pp. abbreviation in the first characters of the parameter
content.

check page for extraneous p, p., pp, pp., pg, pg. at start of parameter value:
        good pattern: '^P[^%.P%l]' matches when page begins PX or P# but not Px
                       where x and X are letters and # is a digit
        bad pattern:  '^[Pp][PpGg]' matches when page begins pp, pP, Pp, PP, pg,
pG, Pg, PG

]]

local function extra_text_in_page_check (val, name)
        if not val:match (cfg.vol_iss_pg_patterns.good_ppattern) then
                for _, pattern in ipairs (cfg.vol_iss_pg_patterns.bad_ppatterns)
do              -- spin through the selected sequence table of patterns
                        if val:match (pattern) then
-- when a match, error so
                                utilities.set_message ('err_extra_text_pages',
name);                  -- add error message
                                return;
-- and done
                        end
                end
        end
end


--[[--------------------------< E X T R A _ T E X T _ I N _ V O L _ I S S _ C H E
C K >------------------------

Adds error if |volume= or |issue= has what appears to be some form of redundant
'type' indicator.

For |volume=:
        'V.', or 'Vol.' (with or without the dot) abbreviations or 'Volume' in
the first characters of the parameter
        content (all case insensitive). 'V' and 'v' (without the dot) are
presumed to be roman numerals so
        are allowed.

For |issue=:
        'No.', 'I.', 'Iss.' (with or without the dot) abbreviations, or 'Issue'
in the first characters of the
        parameter content (all case insensitive).

Single character values ('v', 'i', 'n') allowed when not followed by separator
character ('.', ':', '=', or
```

```
whitespace character) – param values are trimmed of whitespace by MediaWiki
before delivered to the module.

<val> is |volume= or |issue= parameter value
<name> is |volume= or |issue= parameter name for error message
<selector> is 'v' for |volume=, 'i' for |issue=

sets error message on failure; returns nothing

]]

local function extra_text_in_vol_iss_check (val, name, selector)
	if not utilities.is_set (val) then
		return;
	end

	local patterns = 'v' == selector and cfg.vol_iss_pg_patterns.vpatterns or
cfg.vol_iss_pg_patterns.ipatterns;

	local handler = 'v' == selector and 'err_extra_text_volume' or
'err_extra_text_issue';
	val = val:lower();
-- force parameter value to lower case
	for _, pattern in ipairs (patterns) do
-- spin through the selected sequence table of patterns
		if val:match (pattern) then
-- when a match, error so
			utilities.set_message (handler, name);
-- add error message
			return;
-- and done
		end
	end
end


--[=[-------------------------< G E T _ V _ N A M E _ T A B L E >----------------
------------------------------

split apart a |vauthors= or |veditors= parameter.  This function allows for
corporate names, wrapped in doubled
parentheses to also have commas; in the old version of the code, the doubled
parentheses were included in the
rendered citation and in the metadata.  Individual author names may be wikilinked

	|vauthors=Jones AB, [[E. B. White|White EB]], ((Black, Brown, and Co.))

]=]

local function get_v_name_table (vparam, output_table, output_link_table)
	local name_table = mw.text.split(vparam, "%s*,%s*");
-- names are separated by commas
	local wl_type, label, link;
-- wl_type not used here; just a placeholder

	local i = 1;

	while name_table[i] do
		if name_table[i]:match ('^%(%(.*[^%)][^%)]$') then
-- first segment of corporate with one or more commas; this segment has the
opening doubled parentheses
			local name = name_table[i];
```

```
                        i = i + 1;
-- bump indexer to next segment
                        while name_table[i] do
                                name = name .. ', ' .. name_table[i];
-- concatenate with previous segments
                                if name_table[i]:match ('^.*%)%)$') then
-- if this table member has the closing doubled parentheses
                                        break;
-- and done reassembling so
                                end
                                i = i + 1;
-- bump indexer
                        end
                        table.insert (output_table, name);
-- and add corporate name to the output table
                        table.insert (output_link_table, '');
-- no wikilink
                else
                        wl_type, label, link = utilities.is_wikilink
(name_table[i]);                -- wl_type is: 0, no wl (text in label variable);
1, [[D]]; 2, [[L|D]]
                        table.insert (output_table, label);
-- add this name
                        if 1 == wl_type then
                                table.insert (output_link_table, label);
-- simple wikilink [[D]]
                        else
                                table.insert (output_link_table, link);
-- no wikilink or [[L|D]]; add this link if there is one, else empty string
                        end
                end
                i = i + 1;
        end
        return output_table;
end


--[[--------------------------< P A R S E _ V A U T H O R S _ V E D I T O R S >--
------------------------------

This function extracts author / editor names from |vauthors= or |veditors= and
finds matching |xxxxor-maskn= and
|xxxxor-linkn= in args.  It then returns a table of assembled names just as
extract_names() does.

Author / editor names in |vauthors= or |veditors= must be in Vancouver system
style. Corporate or institutional names
may sometimes be required and because such names will often fail the
is_good_vanc_name() and other format compliance
tests, are wrapped in doubled parentheses ((corporate name)) to suppress the
format tests.

Supports generational suffixes Jr, 2nd, 3rd, 4th–6th.

This function sets the Vancouver error when a required comma is missing and when
there is a space between an author's initials.

]]

local function parse_vauthors_veditors (args, vparam, list_name)
        local names = {};
-- table of names assembled from |vauthors=, |author-maskn=, |author-linkn=
```

```
        local v_name_table = {};
        local v_link_table = {};
-- when name is wikilinked, targets go in this table
        local etal = false;
-- return value set to true when we find some form of et al. vauthors parameter
        local last, first, link, mask, suffix;
        local corporate = false;

        vparam, etal = name_has_etal (vparam, etal, true);
-- find and remove variations on et al. do not categorize (do it here because et
al. might have a period)
        v_name_table = get_v_name_table (vparam, v_name_table, v_link_table);
-- names are separated by commas

        for i, v_name in ipairs(v_name_table) do
                first = '';
-- set to empty string for concatenation and because it may have been set for
previous author/editor
                local accept_name;
                v_name, accept_name = utilities.has_accept_as_written (v_name);
-- remove accept-this-as-written markup when it wraps all of <v_name>

                if accept_name then
                        last = v_name;
                        corporate = true;
-- flag used in list_people()
                elseif string.find(v_name, "%s") then
                        if v_name:find('[;%.]') then
-- look for commonly occurring punctuation characters;
                                add_vanc_error (cfg.err_msg_supl.punctuation, i);
                        end
                        local lastfirstTable = {}
                        lastfirstTable = mw.text.split(v_name, "%s+")
                        first = table.remove(lastfirstTable);
-- removes and returns value of last element in table which should be initials or
generational suffix

                        if not mw.ustring.match (first, '^%u+$') then
-- mw.ustring here so that later we will catch non-Latin characters
                                suffix = first;
-- not initials so assume that whatever we got is a generational suffix
                                first = table.remove(lastfirstTable);
-- get what should be the initials from the table
                        end
                        last = table.concat(lastfirstTable, ' ')
-- returns a string that is the concatenation of all other names that are not
initials and generational suffix
                        if not utilities.is_set (last) then
                                first = '';
-- unset
                                last = v_name;
-- last empty because something wrong with first
                                add_vanc_error (cfg.err_msg_supl.name, i);
                        end
                        if mw.ustring.match (last, '%a+%s+%u+%s+%a+') then
                                add_vanc_error (cfg.err_msg_supl['missing
comma'], i);                    -- matches last II last; the case when a comma is
missing
                        end
                        if mw.ustring.match (v_name, ' %u %u$') then
-- this test is in the wrong place TODO: move or replace with a more appropriate
test
```

```
                                add_vanc_error (cfg.err_msg_supl.initials, i);
-- matches a space between two initials
                        end
                else
                        last = v_name;
-- last name or single corporate name?  Doesn't support multiword corporate
names? do we need this?
                end

                if utilities.is_set (first) then
                        if not mw.ustring.match (first, "^%u?%u$") then
-- first shall contain one or two upper-case letters, nothing else
                                add_vanc_error (cfg.err_msg_supl.initials, i);
-- too many initials; mixed case initials (which may be ok Romanization);
hyphenated initials
                        end
                        is_good_vanc_name (last, first, suffix, i);
-- check first and last before restoring the suffix which may have a non-Latin
digit
                        if utilities.is_set (suffix) then
                                first = first .. ' ' .. suffix;
-- if there was a suffix concatenate with the initials
                                suffix = '';
-- unset so we don't add this suffix to all subsequent names
                        end
                else
                        if not corporate then
                                is_good_vanc_name (last, '', nil, i);
                        end
                end

                link = utilities.select_one ( args, cfg.aliases[list_name .. '-
Link'], 'err_redundant_parameters', i ) or v_link_table[i];
                mask = utilities.select_one ( args, cfg.aliases[list_name .. '-
Mask'], 'err_redundant_parameters', i );
                names[i] = {last = last, first = first, link = link, mask = mask,
corporate = corporate};         -- add this assembled name to our names list
        end
        return names, etal;
-- all done, return our list of names
end


--[[--------------------------< S E L E C T _ A U T H O R _ E D I T O R _ S O U R
C E >------------------------

Select one of |authors=, |authorn= / |lastn / firstn=, or |vauthors= as the
source of the author name list or
select one of |editorn= / editor-lastn= / |editor-firstn= or |veditors= as the
source of the editor name list.

Only one of these appropriate three will be used.  The hierarchy is: |authorn=
(and aliases) highest and |authors= lowest;
|editorn= (and aliases) highest and |veditors= lowest (support for |editors=
withdrawn)

When looking for |authorn= / |editorn= parameters, test |xxxxor1= and |xxxxor2=
(and all of their aliases); stops after the second
test which mimicks the test used in extract_names() when looking for a hole in
the author name list.  There may be a better
way to do this, I just haven't discovered what that way is.
```

```
Emits an error message when more than one xxxxor name source is provided.

In this function, vxxxxors = vauthors or veditors; xxxxors = authors as
appropriate.

]]

local function select_author_editor_source (vxxxxors, xxxxors, args, list_name)
        local lastfirst = false;
        if utilities.select_one ( args, cfg.aliases[list_name .. '-Last'],
'none', 1 ) or          -- do this twice in case we have a |first1= without a
|last1=; this ...
                utilities.select_one ( args, cfg.aliases[list_name .. '-First'],
'none', 1 ) or          -- ... also catches the case where |first= is used with
|vauthors=
                utilities.select_one ( args, cfg.aliases[list_name .. '-Last'],
'none', 2 ) or
                utilities.select_one ( args, cfg.aliases[list_name .. '-First'],
'none', 2 ) then
                        lastfirst = true;
        end

        if (utilities.is_set (vxxxxors) and true == lastfirst) or
-- these are the three error conditions
                (utilities.is_set (vxxxxors) and utilities.is_set (xxxxors)) or
                (true == lastfirst and utilities.is_set (xxxxors)) then
                        local err_name;
                        if 'AuthorList' == list_name then
-- figure out which name should be used in error message
                                err_name = 'author';
                        else
                                err_name = 'editor';
                        end
                        utilities.set_message ('err_redundant_parameters',
err_name .. '-name-list parameters');   -- add error message
        end

        if true == lastfirst then return 1 end;
-- return a number indicating which author name source to use
        if utilities.is_set (vxxxxors) then return 2 end;
        if utilities.is_set (xxxxors) then return 3 end;
        return 1;
-- no authors so return 1; this allows missing author name test to run in case
there is a first without last
end


--[[--------------------------< I S _ V A L I D _ P A R A M E T E R _ V A L U E
>----------------------------

This function is used to validate a parameter's assigned value for those
parameters that have only a limited number
of allowable values (yes, y, true, live, dead, etc.).  When the parameter value
has not been assigned a value (missing
or empty in the source template) the function returns the value specified by
ret_val.  If the parameter value is one
of the list of allowed values returns the translated value; else, emits an error
message and returns the value
specified by ret_val.

TODO: explain <invert>
```

```
]]

local function is_valid_parameter_value (value, name, possible, ret_val, invert)
	if not utilities.is_set (value) then
		return ret_val;
-- an empty parameter is ok
	end

	if (not invert and utilities.in_array (value, possible)) then
-- normal; <value> is in <possible> table
		return cfg.keywords_xlate[value];
-- return translation of parameter keyword
	elseif invert and not utilities.in_array (value, possible) then
-- invert; <value> is not in <possible> table
		return value;
-- return <value> as it is
	else
		utilities.set_message ('err_invalid_param_val', {name, value});
-- not an allowed value so add error message
		return ret_val;
	end
end


--[[--------------------------< T E R M I N A T E _ N A M E _ L I S T >----------
------------------------------

This function terminates a name list (author, contributor, editor) with a
separator character (sepc) and a space
when the last character is not a sepc character or when the last three characters
are not sepc followed by two
closing square brackets (close of a wikilink).  When either of these is true, the
name_list is terminated with a
single space character.

]]

local function terminate_name_list (name_list, sepc)
	if (string.sub (name_list, -3, -1) == sepc .. '. ') then
-- if already properly terminated
		return name_list;
-- just return the name list
	elseif (string.sub (name_list, -1, -1) == sepc) or (string.sub
(name_list, -3, -1) == sepc .. ']]') then	-- if last name in list ends with
sepc char
		return name_list .. " ";
-- don't add another
	else
		return name_list .. sepc .. ' ';
-- otherwise terminate the name list
	end
end


--[[-------------------------< F O R M A T _ V O L U M E _ I S S U E >----------
------------------------------

returns the concatenation of the formatted volume and issue parameters as a
single string; or formatted volume
or formatted issue, or an empty string if neither are set.

]]
```

```
local function format_volume_issue (volume, issue, cite_class, origin, sepc,
lower)
	if not utilities.is_set (volume) and not utilities.is_set (issue) then
		return '';
	end

	-- same condition as in format_pages_sheets()
	local is_journal = 'journal' == cite_class or (utilities.in_array
(cite_class, {'citation', 'map', 'interview'}) and 'journal' == origin);

	local is_numeric_vol = volume and (volume:match ('^[MDCLXVI]+$') or
volume:match ('^%d+$'));	-- is only uppercase roman numerals or only
digits?
	local is_long_vol = volume and (4 < mw.ustring.len(volume));
-- is |volume= value longer than 4 characters?

	if volume and (not is_numeric_vol and is_long_vol) then
-- when not all digits or Roman numerals, is |volume= longer than 4 characters?
		utilities.add_prop_cat ('long-vol');
-- yes, add properties cat
	end

	if is_journal then
-- journal-style formatting
		local vol = '';

		if utilities.is_set (volume) then
			if is_numeric_vol then
-- |volume= value all digits or all uppercase Roman numerals?
				vol = utilities.substitute
(cfg.presentation['vol-bold'], {sepc, volume}); -- render in bold face
			elseif is_long_vol then
-- not all digits or Roman numerals; longer than 4 characters?
				vol = utilities.substitute (cfg.messages['j-
vol'], {sepc, utilities.hyphen_to_dash (volume)});	-- not bold
			else
-- four or fewer characters
				vol = utilities.substitute
(cfg.presentation['vol-bold'], {sepc, utilities.hyphen_to_dash (volume)});	-
- bold
			end
		end
		if utilities.is_set (issue) then
			return vol .. utilities.substitute (cfg.messages['j-
issue'], issue);
		end
		return vol;
	end

	if 'podcast' == cite_class and utilities.is_set (issue) then
		return wrap_msg ('issue', {sepc, issue}, lower);
	end

	-- all other types of citation
	if utilities.is_set (volume) and utilities.is_set (issue) then
		return wrap_msg ('vol-no', {sepc, utilities.hyphen_to_dash
(volume), issue}, lower);
	elseif utilities.is_set (volume) then
		return wrap_msg ('vol', {sepc, utilities.hyphen_to_dash
(volume)}, lower);
	else
```

```
                return wrap_msg ('issue', {sepc, issue}, lower);
        end
end


--[[-------------------------< F O R M A T _ P A G E S _ S H E E T S >----------
------------------------------

adds static text to one of |page(s)= or |sheet(s)= values and returns it with all
of the others set to empty strings.
The return order is:
        page, pages, sheet, sheets

Singular has priority over plural when both are provided.

]]

local function format_pages_sheets (page, pages, sheet, sheets, cite_class,
origin, sepc, nopp, lower)
        if 'map' == cite_class then
-- only cite map supports sheet(s) as in-source locators
                if utilities.is_set (sheet) then
                        if 'journal' == origin then
                                return '', '', wrap_msg ('j-sheet', sheet,
lower), '';
                        else
                                return '', '', wrap_msg ('sheet', {sepc, sheet},
lower), '';
                        end
                elseif utilities.is_set (sheets) then
                        if 'journal' == origin then
                                return '', '', '', wrap_msg ('j-sheets', sheets,
lower);
                        else
                                return '', '', '', wrap_msg ('sheets', {sepc,
sheets}, lower);
                        end
                end
        end

        local is_journal = 'journal' == cite_class or (utilities.in_array
(cite_class, {'citation', 'map', 'interview'}) and 'journal' == origin);

        if utilities.is_set (page) then
                if is_journal then
                        return utilities.substitute (cfg.messages['j-page(s)'],
page), '', '', '';
                elseif not nopp then
                        return utilities.substitute (cfg.messages['p-prefix'],
{sepc, page}), '', '', '';
                else
                        return utilities.substitute (cfg.messages['nopp'], {sepc,
page}), '', '', '';
                end
        elseif utilities.is_set (pages) then
                if is_journal then
                        return utilities.substitute (cfg.messages['j-page(s)'],
pages), '', '', '';
                elseif tonumber(pages) ~= nil and not nopp then
-- if pages is only digits, assume a single page number
                        return '', utilities.substitute (cfg.messages['p-
prefix'], {sepc, pages}), '', '';
```

```
                elseif not nopp then
                        return '', utilities.substitute (cfg.messages['pp-
prefix'], {sepc, pages}), '', '';
                else
                        return '', utilities.substitute (cfg.messages['nopp'],
{sepc, pages}), '', '';
                end
        end

        return '', '', '', '';
-- return empty strings
end


--[[--------------------------< I N S O U R C E _ L O C _ G E T >----------------
------------------------------

returns one of the in-source locators: page, pages, or at.

If any of these are interwiki links to Wikisource, returns the label portion of
the interwiki-link as plain text
for use in COinS.  This COinS thing is done because here we convert an interwiki-
link to an external link and
add an icon span around that; get_coins_pages() doesn't know about the span.
TODO: should it?

TODO: add support for sheet and sheets?; streamline;

TODO: make it so that this function returns only one of the three as the single
in-source (the return value assigned
to a new name)?

]]

local function insource_loc_get (page, page_orig, pages, pages_orig, at)
        local ws_url, ws_label, coins_pages, L;
-- for Wikisource interwiki-links; TODO: this corrupts page metadata (span
remains in place after cleanup; fix there?)

        if utilities.is_set (page) then
                if utilities.is_set (pages) or utilities.is_set (at) then
                        pages = '';
-- unset the others
                        at = '';
                end
                extra_text_in_page_check (page, page_orig);
-- emit error message when |page= value begins with what looks like p., pp., etc.

                ws_url, ws_label, L = wikisource_url_make (page);
-- make ws URL from |page= interwiki link; link portion L becomes tooltip label
                if ws_url then
                        page = external_link (ws_url, ws_label .. ' ', 'ws
link in page'); -- space char after label to move icon away from in-source text;
TODO: a better way to do this?
                        page = utilities.substitute (cfg.presentation['interwiki-
icon'], {cfg.presentation['class-wikisource'], L, page});
                        coins_pages = ws_label;
                end
        elseif utilities.is_set (pages) then
                if utilities.is_set (at) then
                        at = '';
-- unset
```

```
                end
                extra_text_in_page_check (pages, pages_orig);
-- emit error message when |page= value begins with what looks like p., pp., etc.

                ws_url, ws_label, L = wikisource_url_make (pages);
-- make ws URL from |pages= interwiki link; link portion L becomes tooltip label
                if ws_url then
                        pages = external_link (ws_url, ws_label .. ' ', 'ws
link in pages');        -- space char after label to move icon away from in-
source text; TODO: a better way to do this?
                        pages = utilities.substitute
(cfg.presentation['interwiki-icon'], {cfg.presentation['class-wikisource'], L,
pages});
                        coins_pages = ws_label;
                end
        elseif utilities.is_set (at) then
                ws_url, ws_label, L = wikisource_url_make (at);
-- make ws URL from |at= interwiki link; link portion L becomes tooltip label
                if ws_url then
                        at = external_link (ws_url, ws_label .. ' ', 'ws
link in at');   -- space char after label to move icon away from in-source text;
TODO: a better way to do this?
                        at = utilities.substitute (cfg.presentation['interwiki-
icon'], {cfg.presentation['class-wikisource'], L, at});
                        coins_pages = ws_label;
                end
        end

        return page, pages, at, coins_pages;
end

--[[--------------------------< I S _ U N I Q U E _ A R C H I V E _ U R L >------
------------------------------

add error message when |archive-url= value is same as |url= or chapter-url= (or
alias...) value

]]

local function is_unique_archive_url (archive, url, c_url, source, date)
        if utilities.is_set (archive) then
                if archive == url or archive == c_url then
                        utilities.set_message ('err_bad_url',
{utilities.wrap_style ('parameter', source)});  -- add error message
                        return '', '';
-- unset |archive-url= and |archive-date= because same as |url= or |chapter-url=
                end
        end

        return archive, date;
end


--[=[-------------------------< A R C H I V E _ U R L _ C H E C K >--------------
------------------------------

Check archive.org URLs to make sure they at least look like they are pointing at
valid archives and not to the
save snapshot URL or to calendar pages.  When the archive URL is
'https://web.archive.org/save/' (or http://...)
archive.org saves a snapshot of the target page in the URL.  That is something
that Wikipedia should not allow
```

```
unwitting readers to do.

When the archive.org URL does not have a complete timestamp, archive.org chooses
a snapshot according to its own
algorithm or provides a calendar 'search' result.  [[WP:ELNO]] discourages links
to search results.

This function looks at the value assigned to |archive-url= and returns empty
strings for |archive-url= and
|archive-date= and an error message when:
        |archive-url= holds an archive.org save command URL
        |archive-url= is an archive.org URL that does not have a complete
timestamp (YYYYMMDDhhmmss 14 digits) in the
                correct place
otherwise returns |archive-url= and |archive-date=

There are two mostly compatible archive.org URLs:
        //web.archive.org/<timestamp>...                -- the old form
        //web.archive.org/web/<timestamp>...    -- the new form

The old form does not support or map to the new form when it contains a display
flag.  There are four identified flags
('id_', 'js_', 'cs_', 'im_') but since archive.org ignores others following the
same form (two letters and an underscore)
we don't check for these specific flags but we do check the form.

This function supports a preview mode.  When the article is rendered in preview
mode, this function may return a modified
archive URL:
        for save command errors, return undated wildcard (/*/)
        for timestamp errors when the timestamp has a wildcard, return the URL
unmodified
        for timestamp errors when the timestamp does not have a wildcard, return
with timestamp limited to six digits plus wildcard (/yyyymm*/)

]=]

local function archive_url_check (url, date)
        local err_msg = '';
-- start with the error message empty
        local path, timestamp, flag;
-- portions of the archive.org URL

        if (not url:match('//web%.archive%.org/')) and (not
url:match('//liveweb%.archive%.org/')) then             -- also deprecated
liveweb Wayback machine URL
                return url, date;
-- not an archive.org archive, return ArchiveURL and ArchiveDate
        end

        if url:match('//web%.archive%.org/save/') then
-- if a save command URL, we don't want to allow saving of the target page
                err_msg = cfg.err_msg_supl.save;
                url = url:gsub ('(//web%.archive%.org)/save/', '%1/*/', 1);
-- for preview mode: modify ArchiveURL
        elseif url:match('//liveweb%.archive%.org/') then
                err_msg = cfg.err_msg_supl.liveweb;
        else
                path, timestamp, flag = url:match('//web%.archive%.org/([^%d]*)
(%d+)([^/]*)/');        -- split out some of the URL parts for evaluation
                if not path then
-- malformed in some way; pattern did not match
```

```
                        err_msg = cfg.err_msg_supl.timestamp;
                elseif 14 ~= timestamp:len() then
-- path and flag optional, must have 14-digit timestamp here
                        err_msg = cfg.err_msg_supl.timestamp;
                        if '*' ~= flag then
                                local replacement = timestamp:match
('^%d%d%d%d%d%d') or timestamp:match ('^%d%d%d%d');     -- get the first 6
(YYYYMM) or first 4 digits (YYYY)
                                if replacement then
-- nil if there aren't at least 4 digits (year)
                                        replacement = replacement .. string.rep
('0', 14 - replacement:len());  -- year or yearmo (4 or 6 digits) zero-fill to
make 14-digit timestamp
                                        url=url:gsub
('(//web%.archive%.org/[^%d]*)%d[^/]*', '%1' .. replacement .. '*', 1)  -- for
preview, modify ts to 14 digits plus splat for calendar display
                                end
                        end
                elseif utilities.is_set (path) and 'web/' ~= path then
-- older archive URLs do not have the extra 'web/' path element
                        err_msg = cfg.err_msg_supl.path;
                elseif utilities.is_set (flag) and not utilities.is_set (path)
then            -- flag not allowed with the old form URL (without the 'web/'
path element)
                        err_msg = cfg.err_msg_supl.flag;
                elseif utilities.is_set (flag) and not flag:match ('%a%a_') then
-- flag if present must be two alpha characters and underscore (requires 'web/'
path element)
                        err_msg = cfg.err_msg_supl.flag;
                else
                        return url, date;
-- return ArchiveURL and ArchiveDate
                end
        end

-- if here, something not right so
        utilities.set_message ('err_archive_url', {err_msg});
-- add error message and

        if is_preview_mode then
                return url, date;
-- preview mode so return ArchiveURL and ArchiveDate
        else
                return '', '';
-- return empty strings for ArchiveURL and ArchiveDate
        end
end


--[[--------------------------< P L A C E _ C H E C K >--------------------------
------------------------------

check |place=, |publication-place=, |location= to see if these params include
digits.  This function added because
many editors misuse location to specify the in-source location (|page(s)= and
|at= are supposed to do that)

returns the original parameter value without modification; added maint cat when
parameter value contains digits

]]
```

```
local function place_check (param_val)
        if not utilities.is_set (param_val) then
-- parameter empty or omitted
                return param_val;
-- return that empty state
        end

        if mw.ustring.find (param_val, '%d') then
-- not empty, are there digits in the parameter value
                utilities.set_message ('maint_location');
-- yep, add maint cat
        end

        return param_val;
-- and done
end


--[[--------------------------< I S _ A R C H I V E D _ C O P Y >----------------
------------------------------

compares |title= to 'Archived copy' (placeholder added by bots that can't find
proper title); if matches, return true; nil else

]]

local function is_archived_copy (title)
        title = mw.ustring.lower(title);
-- switch title to lower case
        if title:find (cfg.special_case_translation.archived_copy.en) then
-- if title is 'Archived copy'
                return true;
        elseif cfg.special_case_translation.archived_copy['local'] then
                if mw.ustring.find (title,
cfg.special_case_translation.archived_copy['local']) then       -- mw.ustring()
because might not be Latin script
                        return true;
                end
        end
end


--[[--------------------------< C I T A T I O N 0 >------------------------------
------------------------------

This is the main function doing the majority of the citation formatting.

]]

local function citation0( config, args )
        --[[
        Load Input Parameters
        The argument_wrapper facilitates the mapping of multiple aliases to
single internal variable.
        ]]
        local A = argument_wrapper ( args );
        local i

        -- Pick out the relevant fields from the arguments.  Different citation
templates
        -- define different field names for the same underlying things.
```

```
        local author_etal;
        local a = {};
-- authors list from |lastn= / |firstn= pairs or |vauthors=
        local Authors;
        local NameListStyle = is_valid_parameter_value (A['NameListStyle'],
A:ORIGIN('NameListStyle'), cfg.keywords_lists['name-list-style'], '');
        local Collaboration = A['Collaboration'];

        do
-- to limit scope of selected
                local selected = select_author_editor_source (A['Vauthors'],
A['Authors'], args, 'AuthorList');
                if 1 == selected then
                        a, author_etal = extract_names (args, 'AuthorList');
-- fetch author list from |authorn= / |lastn= / |firstn=, |author-linkn=, and
|author-maskn=
                elseif 2 == selected then
                        NameListStyle = 'vanc';
-- override whatever |name-list-style= might be
                        a, author_etal = parse_vauthors_veditors (args,
args.vauthors, 'AuthorList');   -- fetch author list from |vauthors=, |author-
linkn=, and |author-maskn=
                elseif 3 == selected then
                        Authors = A['Authors'];
-- use content of |authors=
                        if 'authors' == A:ORIGIN('Authors') then
-- but add a maint cat if the parameter is |authors=
                                utilities.set_message ('maint_authors');
-- because use of this parameter is discouraged; what to do about the aliases is
a TODO:
                        end
                end
                if utilities.is_set (Collaboration) then
                        author_etal = true;
-- so that |display-authors=etal not required
                end
        end

        local editor_etal;
        local e = {};
-- editors list from |editor-lastn= / |editor-firstn= pairs or |veditors=

        do
-- to limit scope of selected
                local selected = select_author_editor_source (A['Veditors'], nil,
args, 'EditorList');    -- support for |editors= withdrawn
                if 1 == selected then
                        e, editor_etal = extract_names (args, 'EditorList');
-- fetch editor list from |editorn= / |editor-lastn= / |editor-firstn=, |editor-
linkn=, and |editor-maskn=
                elseif 2 == selected then
                        NameListStyle = 'vanc';
-- override whatever |name-list-style= might be
                        e, editor_etal = parse_vauthors_veditors (args,
args.veditors, 'EditorList');   -- fetch editor list from |veditors=, |editor-
linkn=, and |editor-maskn=
                end
        end

        local Chapter = A['Chapter'];
-- done here so that we have access to |contribution= from |chapter= aliases
        local Chapter_origin = A:ORIGIN ('Chapter');
```

```
        local Contribution;
-- because contribution is required for contributor(s)
                if 'contribution' == Chapter_origin then
                        Contribution = Chapter;
-- get the name of the contribution
                end
        local c = {};
-- contributors list from |contributor-lastn= / contributor-firstn= pairs

        if utilities.in_array (config.CitationClass, {"book", "citation"}) and
not utilities.is_set (A['Periodical']) then     -- |contributor= and
|contribution= only supported in book cites
                c = extract_names (args, 'ContributorList');
-- fetch contributor list from |contributorn= / |contributor-lastn=, -firstn=, -
linkn=, -maskn=

                if 0 < #c then
                        if not utilities.is_set (Contribution) then
-- |contributor= requires |contribution=
                                utilities.set_message
('err_contributor_missing_required_param', 'contribution');     -- add missing
contribution error message
                                c = {};
-- blank the contributors' table; it is used as a flag later
                        end
                        if 0 == #a then
-- |contributor= requires |author=
                                utilities.set_message
('err_contributor_missing_required_param', 'author');   -- add missing author
error message
                                c = {};
-- blank the contributors' table; it is used as a flag later
                        end
                end
        else
-- if not a book cite
                if utilities.select_one (args, cfg.aliases['ContributorList-
Last'], 'err_redundant_parameters', 1 ) then    -- are there contributor name
list parameters?
                        utilities.set_message ('err_contributor_ignored');
-- add contributor ignored error message
                end
                Contribution = nil;
-- unset
        end

        local Title = A['Title'];
        local TitleLink = A['TitleLink'];

        local auto_select = ''; -- default is auto
        local accept_link;
        TitleLink, accept_link = utilities.has_accept_as_written (TitleLink,
true);  -- test for accept-this-as-written markup
        if (not accept_link) and utilities.in_array (TitleLink, {'none', 'pmc',
'doi'}) then -- check for special keywords
                auto_select = TitleLink;
-- remember selection for later
                TitleLink = '';
-- treat as if |title-link= would have been empty
        end

        TitleLink = link_title_ok (TitleLink, A:ORIGIN ('TitleLink'), Title,
```

```
'title');       -- check for wiki-markup in |title-link= or wiki-markup in
|title= when |title-link= is set

        local Section = '';
-- {{cite map}} only; preset to empty string for concatenation if not used
        if 'map' == config.CitationClass and 'section' == Chapter_origin then
                Section = A['Chapter'];
-- get |section= from |chapter= alias list; |chapter= and the other aliases not
supported in {{cite map}}
                Chapter = '';
-- unset for now; will be reset later from |map= if present
        end

        local Periodical = A['Periodical'];
        local Periodical_origin = '';
        if utilities.is_set (Periodical) then
                Periodical_origin = A:ORIGIN('Periodical');
-- get the name of the periodical parameter
                local i;
                Periodical, i = utilities.strip_apostrophe_markup (Periodical);
-- strip apostrophe markup so that metadata isn't contaminated
                if i then
-- non-zero when markup was stripped so emit an error message
                        utilities.set_message ('err_apostrophe_markup',
{Periodical_origin});
                end
        end

        if 'mailinglist' == config.CitationClass then
-- special case for {{cite mailing list}}
                if utilities.is_set (Periodical) and utilities.is_set (A
['MailingList']) then   -- both set emit an error TODO: make a function for this
and similar?
                        utilities.set_message ('err_redundant_parameters',
{utilities.wrap_style ('parameter', Periodical_origin) .. ' and ' ..
utilities.wrap_style ('parameter', 'mailinglist')});
                end

                Periodical = A ['MailingList'];
-- error or no, set Periodical to |mailinglist= value because this template is
{{cite mailing list}}
                Periodical_origin = A:ORIGIN('MailingList');
        end

        local ScriptPeriodical = A['ScriptPeriodical'];

        -- web and news not tested for now because of
        -- Wikipedia:Administrators%27_noticeboard#Is_there_a_semi-
automated_tool_that_could_fix_these_annoying_"Cite_Web"_errors?
        if not (utilities.is_set (Periodical) or utilities.is_set
(ScriptPeriodical)) then        -- 'periodical' templates require periodical
parameter
        --      local p = {['journal'] = 'journal', ['magazine'] = 'magazine',
['news'] = 'newspaper', ['web'] = 'website'};   -- for error message
                local p = {['journal'] = 'journal', ['magazine'] = 'magazine'};
-- for error message
                if p[config.CitationClass]  then
                        utilities.set_message ('err_missing_periodical',
{config.CitationClass, p[config.CitationClass]});
                end
        end
```

```
        local Volume;
        local ScriptPeriodical_origin = A:ORIGIN('ScriptPeriodical');
        if 'citation' == config.CitationClass then
                if utilities.is_set (Periodical) then
                        if not utilities.in_array (Periodical_origin,
cfg.citation_no_volume_t) then  -- {{citation}} does not render |volume= when
these parameters are used
                                Volume = A['Volume'];
-- but does for all other 'periodicals'
                        end
                elseif utilities.is_set (ScriptPeriodical) then
                        if 'script-website' ~= ScriptPeriodical_origin then
-- {{citation}} does not render volume for |script-website=
                                Volume = A['Volume'];
-- but does for all other 'periodicals'
                        end
                else
                        Volume = A['Volume'];
-- and does for non-'periodical' cites
                end
        elseif utilities.in_array (config.CitationClass,
cfg.templates_using_volume) then        -- render |volume= for cs1 according to
the configuration settings
                Volume = A['Volume'];
        end
        extra_text_in_vol_iss_check (Volume, A:ORIGIN ('Volume'), 'v');

        local Issue;
        if 'citation' == config.CitationClass then
                if utilities.is_set (Periodical) and utilities.in_array
(Periodical_origin, cfg.citation_issue_t) then  -- {{citation}} may render
|issue= when these parameters are used
                        Issue = utilities.hyphen_to_dash (A['Issue']);
                end
        elseif utilities.in_array (config.CitationClass,
cfg.templates_using_issue) then -- conference & map books do not support issue;
{{citation}} listed here because included in settings table
                if not (utilities.in_array (config.CitationClass, {'conference',
'map', 'citation'}) and not (utilities.is_set (Periodical) or utilities.is_set
(ScriptPeriodical))) then
                        Issue = utilities.hyphen_to_dash (A['Issue']);
                end
        end
        extra_text_in_vol_iss_check (Issue, A:ORIGIN ('Issue'), 'i');

        local Page;
        local Pages;
        local At;
        if not utilities.in_array (config.CitationClass,
cfg.templates_not_using_page) then
                Page = A['Page'];
                Pages = utilities.hyphen_to_dash (A['Pages']);
                At = A['At'];
        end

        local Edition = A['Edition'];
        local PublicationPlace = place_check (A['PublicationPlace'],
A:ORIGIN('PublicationPlace'));
        local Place = place_check (A['Place'], A:ORIGIN('Place'));

        local PublisherName = A['PublisherName'];
        local PublisherName_origin = A:ORIGIN('PublisherName');
```

```
        if utilities.is_set (PublisherName) then
                local i = 0;
                PublisherName, i = utilities.strip_apostrophe_markup
(PublisherName);        -- strip apostrophe markup so that metadata isn't
contaminated; publisher is never italicized
                if i then
-- non-zero when markup was stripped so emit an error message
                        utilities.set_message ('err_apostrophe_markup',
{PublisherName_origin});
                end
        end

        local Newsgroup = A['Newsgroup'];
-- TODO: strip apostrophe markup?
        local Newsgroup_origin = A:ORIGIN('Newsgroup');

        if 'newsgroup' == config.CitationClass then
                if utilities.is_set (PublisherName) then
-- general use parameter |publisher= not allowed in cite newsgroup
                        utilities.set_message ('err_parameter_ignored',
{PublisherName_origin});
                end

                PublisherName = nil;
-- ensure that this parameter is unset for the time being; will be used again
after COinS
        end

        local URL = A['URL'];
-- TODO: better way to do this for URL, ChapterURL, and MapURL?
        local UrlAccess = is_valid_parameter_value (A['UrlAccess'],
A:ORIGIN('UrlAccess'), cfg.keywords_lists['url-access'], nil);

        if not utilities.is_set (URL) and utilities.is_set (UrlAccess) then
                UrlAccess = nil;
                utilities.set_message ('err_param_access_requires_param', 'url');
        end

        local ChapterURL = A['ChapterURL'];
        local ChapterUrlAccess = is_valid_parameter_value (A['ChapterUrlAccess'],
A:ORIGIN('ChapterUrlAccess'), cfg.keywords_lists['url-access'], nil);
        if not utilities.is_set (ChapterURL) and utilities.is_set
(ChapterUrlAccess) then
                ChapterUrlAccess = nil;
                utilities.set_message ('err_param_access_requires_param',
{A:ORIGIN('ChapterUrlAccess'):gsub ('%-access', '')});
        end

        local MapUrlAccess = is_valid_parameter_value (A['MapUrlAccess'],
A:ORIGIN('MapUrlAccess'), cfg.keywords_lists['url-access'], nil);
        if not utilities.is_set (A['MapURL']) and utilities.is_set (MapUrlAccess)
then
                MapUrlAccess = nil;
                utilities.set_message ('err_param_access_requires_param', {'map-
url'});
        end

        local this_page = mw.title.getCurrentTitle();
-- also used for COinS and for language
        local no_tracking_cats = is_valid_parameter_value (A['NoTracking'],
A:ORIGIN('NoTracking'), cfg.keywords_lists['yes_true_y'], nil);
```

```
        -- check this page to see if it is in one of the namespaces that cs1 is
not supposed to add to the error categories
        if not utilities.is_set (no_tracking_cats) then
-- ignore if we are already not going to categorize this page
                if utilities.in_array (this_page.nsText,
cfg.uncategorized_namespaces) then
                        no_tracking_cats = "true";
-- set no_tracking_cats
                end
                for _, v in ipairs (cfg.uncategorized_subpages) do
-- cycle through page name patterns
                        if this_page.text:match (v) then
-- test page name against each pattern
                                no_tracking_cats = "true";
-- set no_tracking_cats
                                break;
-- bail out if one is found
                        end
                end
        end

-- check for extra |page=, |pages= or |at= parameters. (also sheet and sheets
while we're at it)
        utilities.select_one (args, {'page', 'p', 'pp', 'pages', 'at', 'sheet',
'sheets'}, 'err_redundant_parameters'); -- this is a dummy call simply to get the
error message and category

        local coins_pages;

        Page, Pages, At, coins_pages = insource_loc_get (Page, A:ORIGIN('Page'),
Pages, A:ORIGIN('Pages'), At);

        local NoPP = is_valid_parameter_value (A['NoPP'], A:ORIGIN('NoPP'),
cfg.keywords_lists['yes_true_y'], nil);

        if utilities.is_set (PublicationPlace) and utilities.is_set (Place) then
-- both |publication-place= and |place= (|location=) allowed if different
                utilities.add_prop_cat ('location-test');
-- add property cat to evaluate how often PublicationPlace and Place are used
together
                if PublicationPlace == Place then
                        Place = '';
-- unset; don't need both if they are the same
                end
        elseif not utilities.is_set (PublicationPlace) and utilities.is_set
(Place) then    -- when only |place= (|location=) is set ...
                PublicationPlace = Place;
-- promote |place= (|location=) to |publication-place
        end

        if PublicationPlace == Place then Place = ''; end
-- don't need both if they are the same

        local URL_origin = A:ORIGIN('URL');
-- get name of parameter that holds URL
        local ChapterURL_origin = A:ORIGIN('ChapterURL');
-- get name of parameter that holds ChapterURL
        local ScriptChapter = A['ScriptChapter'];
        local ScriptChapter_origin = A:ORIGIN ('ScriptChapter');
        local Format = A['Format'];
        local ChapterFormat = A['ChapterFormat'];
        local TransChapter = A['TransChapter'];
```

```
        local TransChapter_origin = A:ORIGIN ('TransChapter');
        local TransTitle = A['TransTitle'];
        local ScriptTitle = A['ScriptTitle'];

        --[[
        Parameter remapping for cite encyclopedia:
        When the citation has these parameters:
                |encyclopedia= and |title= then map |title= to |article= and
|encyclopedia= to |title=
                |encyclopedia= and |article= then map |encyclopedia= to |title=

                |trans-title= maps to |trans-chapter= when |title= is re-mapped
                |url= maps to |chapter-url= when |title= is remapped

        All other combinations of |encyclopedia=, |title=, and |article= are not
modified

        ]]

        local Encyclopedia = A['Encyclopedia'];
-- used as a flag by this module and by ~/COinS

        if utilities.is_set (Encyclopedia) then
-- emit error message when Encyclopedia set but template is other than {{cite
encyclopedia}} or {{citation}}
                if 'encyclopaedia' ~= config.CitationClass and 'citation' ~=
config.CitationClass then
                        utilities.set_message ('err_parameter_ignored', {A:ORIGIN
('Encyclopedia')});
                        Encyclopedia = nil;
-- unset because not supported by this template
                end
        end

        if ('encyclopaedia' == config.CitationClass) or ('citation' ==
config.CitationClass and utilities.is_set (Encyclopedia)) then
                if utilities.is_set (Periodical) and utilities.is_set
(Encyclopedia) then     -- when both set emit an error TODO: make a function for
this and similar?
                        utilities.set_message ('err_redundant_parameters',
{utilities.wrap_style ('parameter', A:ORIGIN ('Encyclopedia')) .. ' and ' ..
utilities.wrap_style ('parameter', Periodical_origin)});
                end

                if utilities.is_set (Encyclopedia) then
                        Periodical = Encyclopedia;
-- error or no, set Periodical to Encyclopedia; allow periodical without
encyclopedia
                        Periodical_origin = A:ORIGIN ('Encyclopedia');
                end

                if utilities.is_set (Periodical) then
-- Periodical is set when |encyclopedia= is set
                        if utilities.is_set (Title) or utilities.is_set
(ScriptTitle) then
                                if not utilities.is_set (Chapter) then
                                        Chapter = Title;
-- |encyclopedia= and |title= are set so map |title= to |article= and
|encyclopedia= to |title=
                                        ScriptChapter = ScriptTitle;
                                        ScriptChapter_origin =
A:ORIGIN('ScriptTitle')
```

```
                                        TransChapter = TransTitle;
                                        ChapterURL = URL;
                                        ChapterURL_origin = URL_origin;

                                        ChapterUrlAccess = UrlAccess;

                                        if not utilities.is_set (ChapterURL) and
utilities.is_set (TitleLink) then
                                                Chapter = utilities.make_wikilink
(TitleLink, Chapter);
                                        end
                                        Title = Periodical;
                                        ChapterFormat = Format;
                                        Periodical = '';
-- redundant so unset
                                        TransTitle = '';
                                        URL = '';
                                        Format = '';
                                        TitleLink = '';
                                        ScriptTitle = '';
                                end
                        elseif utilities.is_set (Chapter) or utilities.is_set
(ScriptChapter) then                                                    -
- |title= not set
                                Title = Periodical;
-- |encyclopedia= set and |article= set so map |encyclopedia= to |title=
                                Periodical = '';
-- redundant so unset
                        end
                end
        end

        -- special case for cite techreport.
        local ID = A['ID'];
        if (config.CitationClass == "techreport") then
-- special case for cite techreport
                if utilities.is_set (A['Number']) then
-- cite techreport uses 'number', which other citations alias to 'issue'
                        if not utilities.is_set (ID) then
-- can we use ID for the "number"?
                                ID = A['Number'];
-- yes, use it
                        else
-- ID has a value so emit error message
                                utilities.set_message
('err_redundant_parameters', {utilities.wrap_style ('parameter', 'id') .. ' and '
.. utilities.wrap_style ('parameter', 'number')});
                        end
                end
        end

        -- Account for the oddity that is {{cite conference}}, before generation
of COinS data.
        local ChapterLink -- = A['ChapterLink'];
-- deprecated as a parameter but still used internally by cite episode
        local Conference = A['Conference'];
        local BookTitle = A['BookTitle'];
        local TransTitle_origin = A:ORIGIN ('TransTitle');
        if 'conference' == config.CitationClass then
                if utilities.is_set (BookTitle) then
                        Chapter = Title;
                        Chapter_origin = 'title';
```

```
        --              ChapterLink = TitleLink;
-- |chapter-link= is deprecated
                        ChapterURL = URL;
                        ChapterUrlAccess = UrlAccess;
                        ChapterURL_origin = URL_origin;
                        URL_origin = '';
                        ChapterFormat = Format;
                        TransChapter = TransTitle;
                        TransChapter_origin = TransTitle_origin;
                        Title = BookTitle;
                        Format = '';
        --              TitleLink = '';
                        TransTitle = '';
                        URL = '';
                end
        elseif 'speech' ~= config.CitationClass then
                Conference = '';
-- not cite conference or cite speech so make sure this is empty string
        end

        -- CS1/2 mode
        local Mode = is_valid_parameter_value (A['Mode'], A:ORIGIN('Mode'),
cfg.keywords_lists['mode'], '');
        -- separator character and postscript
        local sepc, PostScript = set_style (Mode:lower(), A['PostScript'],
config.CitationClass);
        -- controls capitalization of certain static text
        local use_lowercase = ( sepc == ',' );

        -- cite map oddities
        local Cartography = "";
        local Scale = "";
        local Sheet = A['Sheet'] or '';
        local Sheets = A['Sheets'] or '';
        if config.CitationClass == "map" then
                if utilities.is_set (Chapter) then
--TODO: make a function for this and similar?
                        utilities.set_message ('err_redundant_parameters',
{utilities.wrap_style ('parameter', 'map') .. ' and ' .. utilities.wrap_style
('parameter', Chapter_origin)});        -- add error message
                end
                Chapter = A['Map'];
                Chapter_origin = A:ORIGIN('Map');
                ChapterURL = A['MapURL'];
                ChapterURL_origin = A:ORIGIN('MapURL');
                TransChapter = A['TransMap'];
                ScriptChapter = A['ScriptMap']
                ScriptChapter_origin = A:ORIGIN('ScriptMap')

                ChapterUrlAccess = MapUrlAccess;
                ChapterFormat = A['MapFormat'];

                Cartography = A['Cartography'];
                if utilities.is_set ( Cartography ) then
                        Cartography = sepc .. " " .. wrap_msg ('cartography',
Cartography, use_lowercase);
                end
                Scale = A['Scale'];
                if utilities.is_set ( Scale ) then
                        Scale = sepc .. " " .. Scale;
                end
        end
```

```
        -- Account for the oddities that are {{cite episode}} and {{cite
serial}}, before generation of COinS data.
        local Series = A['Series'];
        if 'episode' == config.CitationClass or 'serial' == config.CitationClass
then
                local SeriesLink = A['SeriesLink'];

                SeriesLink = link_title_ok (SeriesLink, A:ORIGIN ('SeriesLink'),
Series, 'series');      -- check for wiki-markup in |series-link= or wiki-markup
in |series= when |series-link= is set

                local Network = A['Network'];
                local Station = A['Station'];
                local s, n = {}, {};

-- do common parameters first
                if utilities.is_set (Network) then table.insert(n, Network); end
                if utilities.is_set (Station) then table.insert(n, Station); end
                ID = table.concat(n, sepc .. ' ');

                if 'episode' == config.CitationClass then
-- handle the oddities that are strictly {{cite episode}}
                        local Season = A['Season'];
                        local SeriesNumber = A['SeriesNumber'];

                        if utilities.is_set (Season) and utilities.is_set
(SeriesNumber) then     -- these are mutually exclusive so if both are set TODO:
make a function for this and similar?
                                utilities.set_message
('err_redundant_parameters', {utilities.wrap_style ('parameter', 'season') .. '
and ' .. utilities.wrap_style ('parameter', 'seriesno')});              -- add
error message
                                SeriesNumber = '';
-- unset; prefer |season= over |seriesno=
                        end

-- assemble a table of parts concatenated later into Series
                        if utilities.is_set (Season) then table.insert(s,
wrap_msg ('season', Season, use_lowercase)); end
                        if utilities.is_set (SeriesNumber) then table.insert(s,
wrap_msg ('seriesnum', SeriesNumber, use_lowercase)); end
                        if utilities.is_set (Issue) then table.insert(s, wrap_msg
('episode', Issue, use_lowercase)); end
                        Issue = '';
-- unset because this is not a unique parameter

                        Chapter = Title;
-- promote title parameters to chapter
                        ScriptChapter = ScriptTitle;
                        ScriptChapter_origin = A:ORIGIN('ScriptTitle');
                        ChapterLink = TitleLink;
-- alias |episode-link=
                        TransChapter = TransTitle;
                        ChapterURL = URL;
                        ChapterUrlAccess = UrlAccess;
                        ChapterURL_origin = URL_origin;
                        ChapterFormat = Format;

                        Title = Series;
-- promote series to title
                        TitleLink = SeriesLink;
```

```
                        Series = table.concat(s, sepc .. ' ');
-- this is concatenation of season, seriesno, episode number

                        if utilities.is_set (ChapterLink) and not
utilities.is_set (ChapterURL) then      -- link but not URL
                                Chapter = utilities.make_wikilink (ChapterLink,
Chapter);
                        elseif utilities.is_set (ChapterLink) and
utilities.is_set (ChapterURL) then      -- if both are set, URL links episode;
                                Series = utilities.make_wikilink (ChapterLink,
Series);
                        end
                        URL = '';
-- unset
                        TransTitle = '';
                        ScriptTitle = '';
                        Format = '';

                else
-- now oddities that are cite serial
                        Issue = '';
-- unset because this parameter no longer supported by the citation/core version
of cite serial
                        Chapter = A['Episode'];
-- TODO: make |episode= available to cite episode someday?
                        if utilities.is_set (Series) and utilities.is_set
(SeriesLink) then
                                Series = utilities.make_wikilink (SeriesLink,
Series);
                        end
                        Series = utilities.wrap_style ('italic-title', Series);
-- series is italicized
                end
        end
        -- end of {{cite episode}} stuff

        -- handle type parameter for those CS1 citations that have default values
        local TitleType = A['TitleType'];
        local Degree = A['Degree'];
        if utilities.in_array (config.CitationClass, {'AV-media-notes',
'interview', 'mailinglist', 'map', 'podcast', 'pressrelease', 'report', 'speech',
'techreport', 'thesis'}) then
                TitleType = set_titletype (config.CitationClass, TitleType);
                if utilities.is_set (Degree) and "Thesis" == TitleType then
-- special case for cite thesis
                        TitleType = Degree .. ' ' .. cfg.title_types
['thesis']:lower();
                end
        end

        if utilities.is_set (TitleType) then
-- if type parameter is specified
                TitleType = utilities.substitute ( cfg.messages['type'],
TitleType);     -- display it in parentheses
        -- TODO: Hack on TitleType to fix bunched parentheses problem
        end

        -- legacy: promote PublicationDate to Date if neither Date nor Year are
set.
        local Date = A['Date'];
        local Date_origin;
-- to hold the name of parameter promoted to Date; required for date error
```

```
messaging
        local PublicationDate = A['PublicationDate'];
        local Year = A['Year'];

        if not utilities.is_set (Date) then
                Date = Year;
-- promote Year to Date
                Year = nil;
-- make nil so Year as empty string isn't used for CITEREF
                if not utilities.is_set (Date) and utilities.is_set
(PublicationDate) then  -- use PublicationDate when |date= and |year= are not set
                        Date = PublicationDate;
-- promote PublicationDate to Date
                        PublicationDate = '';
-- unset, no longer needed
                        Date_origin = A:ORIGIN('PublicationDate');
-- save the name of the promoted parameter
                else
                        Date_origin = A:ORIGIN('Year');
-- save the name of the promoted parameter
                end
        else
                Date_origin = A:ORIGIN('Date');
-- not a promotion; name required for error messaging
        end

        if PublicationDate == Date then PublicationDate = ''; end
-- if PublicationDate is same as Date, don't display in rendered citation

        --[[
        Go test all of the date-holding parameters for valid MOS:DATE format and
make sure that dates are real dates. This must be done before we do COinS because
here is where
        we get the date used in the metadata.

        Date validation supporting code is in Module:Citation/CS1/Date_validation
        ]]

        local DF = is_valid_parameter_value (A['DF'], A:ORIGIN('DF'),
cfg.keywords_lists['df'], '');
        if not utilities.is_set (DF) then
                DF = cfg.global_df;
-- local |df= if present overrides global df set by {{use xxx date}} template
        end

        local ArchiveURL;
        local ArchiveDate;
        local ArchiveFormat = A['ArchiveFormat'];

        ArchiveURL, ArchiveDate = archive_url_check (A['ArchiveURL'],
A['ArchiveDate'])
        ArchiveFormat = style_format (ArchiveFormat, ArchiveURL, 'archive-
format', 'archive-url');

        ArchiveURL, ArchiveDate = is_unique_archive_url (ArchiveURL, URL,
ChapterURL, A:ORIGIN('ArchiveURL'), ArchiveDate);               -- add error
message when URL or ChapterURL == ArchiveURL


        local AccessDate = A['AccessDate'];
        local LayDate = A['LayDate'];
        local COinS_date = {};
```

```
-- holds date info extracted from |date= for the COinS metadata by Module:Date
verification
        local DoiBroken = A['DoiBroken'];
        local Embargo = A['Embargo'];
        local anchor_year;
-- used in the CITEREF identifier
        do      -- create defined block to contain local variables error_message,
date_parameters_list, mismatch
                local error_message = '';

-- AirDate has been promoted to Date so not necessary to check it
                local date_parameters_list = {
                        ['access-date'] = {val = AccessDate, name = A:ORIGIN
('AccessDate')},
                        ['archive-date'] = {val = ArchiveDate, name = A:ORIGIN
('ArchiveDate')},
                        ['date'] = {val = Date, name = Date_origin},
                        ['doi-broken-date'] = {val = DoiBroken, name = A:ORIGIN
('DoiBroken')},
                        ['pmc-embargo-date'] = {val = Embargo, name = A:ORIGIN
('Embargo')},
                        ['lay-date'] = {val = LayDate, name = A:ORIGIN
('LayDate')},
                        ['publication-date'] = {val = PublicationDate, name =
A:ORIGIN ('PublicationDate')},
                        ['year'] = {val = Year, name = A:ORIGIN ('Year')},
                        };

                local error_list = {};
                anchor_year, Embargo = validation.dates(date_parameters_list,
COinS_date, error_list);

-- start temporary Julian / Gregorian calendar uncertainty categorization
                if COinS_date.inter_cal_cat then
                        utilities.add_prop_cat ('jul-greg-uncertainty');
                end
-- end temporary Julian / Gregorian calendar uncertainty categorization

                if utilities.is_set (Year) and utilities.is_set (Date) then
-- both |date= and |year= not normally needed;
                        validation.year_date_check (Year, A:ORIGIN ('Year'),
Date, A:ORIGIN ('Date'), error_list);
                end

                if 0 == #error_list then
-- error free dates only; 0 when error_list is empty
                        local modified = false;
-- flag

                        if utilities.is_set (DF) then
-- if we need to reformat dates
                                modified = validation.reformat_dates
(date_parameters_list, DF);     -- reformat to DF format, use long month names if
appropriate
                        end

                        if true == validation.date_hyphen_to_dash
(date_parameters_list) then     -- convert hyphens to dashes where appropriate
                                modified = true;
                                utilities.set_message ('maint_date_format');
-- hyphens were converted so add maint category
                        end
```

```
        -- for those wikis that can and want to have English date names
translated to the local language; not supported at en.wiki
                        if cfg.date_name_auto_xlate_enable and
validation.date_name_xlate (date_parameters_list,
cfg.date_digit_auto_xlate_enable ) then
                                utilities.set_message ('maint_date_auto_xlated');
-- add maint cat
                                modified = true;
                        end

                        if modified then
-- if the date_parameters_list values were modified
                                AccessDate = date_parameters_list['access-
date'].val;                     -- overwrite date holding parameters with
modified values
                                ArchiveDate = date_parameters_list['archive-
date'].val;
                                Date = date_parameters_list['date'].val;
                                DoiBroken = date_parameters_list['doi-broken-
date'].val;
                                LayDate = date_parameters_list['lay-date'].val;
                                PublicationDate =
date_parameters_list['publication-date'].val;
                        end
                else
                        utilities.set_message ('err_bad_date',
{utilities.make_sep_list (#error_list, error_list)});   -- add this error message
                end
        end     -- end of do

        local ID_list = {};
-- sequence table of rendered identifiers
        local ID_list_coins = {};
-- table of identifiers and their values from args; key is same as
cfg.id_handlers's key
        local Class = A['Class'];
-- arxiv class identifier

        local ID_support = {
                {A['ASINTLD'], 'ASIN', 'err_asintld_missing_asin', A:ORIGIN
('ASINTLD')},
                {DoiBroken, 'DOI', 'err_doibroken_missing_doi', A:ORIGIN
('DoiBroken')},
                {Embargo, 'PMC', 'err_embargo_missing_pmc', A:ORIGIN
('Embargo')},
                }

        ID_list, ID_list_coins = identifiers.identifier_lists_get (args,
{DoiBroken = DoiBroken, ASINTLD = A['ASINTLD'], Embargo = Embargo, Class =
Class}, ID_support);

        -- Account for the oddities that are {{cite arxiv}}, {{cite biorxiv}},
{{cite citeseerx}}, {{cite ssrn}}, before generation of COinS data.
        if utilities.in_array (config.CitationClass,
whitelist.preprint_template_list) then
                if not utilities.is_set
(ID_list_coins[config.CitationClass:upper()]) then      -- |arxiv= or |eprint=
required for cite arxiv; |biorxiv= & |citeseerx= required for their templates
                        utilities.set_message ('err_' .. config.CitationClass ..
'_missing');    -- add error message
                end
```

```
                Periodical = ({['arxiv'] = 'arXiv', ['biorxiv'] = 'bioRxiv',
['citeseerx'] = 'CiteSeerX', ['ssrn'] = 'Social Science Research Network'})
[config.CitationClass];
        end

        -- Link the title of the work if no |url= was provided, but we have a
|pmc= or a |doi= with |doi-access=free

        if config.CitationClass == "journal" and not utilities.is_set (URL) and
not utilities.is_set (TitleLink) and not utilities.in_array
(cfg.keywords_xlate[Title], {'off', 'none'}) then -- TODO: remove 'none' once
existing citations have been switched to 'off', so 'none' can be used as token
for "no title" instead
                if 'none' ~= cfg.keywords_xlate[auto_select] then
-- if auto-linking not disabled
                        if identifiers.auto_link_urls[auto_select] then
-- manual selection
                                URL = identifiers.auto_link_urls[auto_select];
-- set URL to be the same as identifier's external link
                                URL_origin =
cfg.id_handlers[auto_select:upper()].parameters[1];     -- set URL_origin to
parameter name for use in error message if citation is missing a |title=
                        elseif identifiers.auto_link_urls['pmc'] then
-- auto-select PMC
                                URL = identifiers.auto_link_urls['pmc'];
-- set URL to be the same as the PMC external link if not embargoed
                                URL_origin =
cfg.id_handlers['PMC'].parameters[1];                           -- set URL_origin
to parameter name for use in error message if citation is missing a |title=
                        elseif identifiers.auto_link_urls['doi'] then
-- auto-select DOI
                                URL = identifiers.auto_link_urls['doi'];
                                URL_origin =
cfg.id_handlers['DOI'].parameters[1];
                        end
                end

                if utilities.is_set (URL) then
-- set when using an identifier-created URL
                        if utilities.is_set (AccessDate) then
-- |access-date= requires |url=; identifier-created URL is not |url=
                                utilities.set_message
('err_accessdate_missing_url');                 -- add an error message
                                AccessDate = '';
-- unset
                        end

                        if utilities.is_set (ArchiveURL) then
-- |archive-url= requires |url=; identifier-created URL is not |url=
                                utilities.set_message
('err_archive_missing_url');                            -- add an error message
                                ArchiveURL = '';
-- unset
                        end
                end
        end

        -- At this point fields may be nil if they weren't specified in the
template use.  We can use that fact.
        -- Test if citation has no title
        if      not utilities.is_set (Title) and not utilities.is_set
```

```
(TransTitle) and not utilities.is_set (ScriptTitle) then       -- has special
case for cite episode
                utilities.set_message ('err_citation_missing_title', {'episode'
== config.CitationClass and 'series' or 'title'});
        end

        if utilities.in_array (cfg.keywords_xlate[Title], {'off', 'none'}) and
                        utilities.in_array (config.CitationClass, {'journal',
'citation'}) and
                        (utilities.is_set (Periodical) or utilities.is_set
(ScriptPeriodical)) and
                        ('journal' == Periodical_origin or 'script-journal' ==
ScriptPeriodical_origin) then   -- special case for journal cites
                                Title = '';
-- set title to empty string
                                utilities.set_message ('maint_untitled');
-- add maint cat
        end

        -- COinS metadata (see <http://ocoins.info/>) for automated parsing of
citation information.
        -- handle the oddity that is cite encyclopedia and {{citation
|encyclopedia=something}}. Here we presume that
        -- when Periodical, Title, and Chapter are all set, then Periodical is
the book (encyclopedia) title, Title
        -- is the article title, and Chapter is a section within the article.
So, we remap

        local coins_chapter = Chapter;
-- default assuming that remapping not required
        local coins_title = Title;
-- et tu
        if 'encyclopaedia' == config.CitationClass or ('citation' ==
config.CitationClass and utilities.is_set (Encyclopedia)) then
                if utilities.is_set (Chapter) and utilities.is_set (Title) and
utilities.is_set (Periodical) then              -- if all are used then
                        coins_chapter = Title;
-- remap
                        coins_title = Periodical;
                end
        end
        local coins_author = a;
-- default for coins rft.au
        if 0 < #c then
-- but if contributor list
                coins_author = c;
-- use that instead
        end

        local QuotePage = A['QuotePage'];
        local QuotePages = utilities.hyphen_to_dash (A['QuotePages']);

        -- this is the function call to COinS()
        local OCinSoutput = metadata.COinS({
                ['Periodical'] = utilities.strip_apostrophe_markup (Periodical),
-- no markup in the metadata
                ['Encyclopedia'] = Encyclopedia,
-- just a flag; content ignored by ~/COinS
                ['Chapter'] = metadata.make_coins_title (coins_chapter,
ScriptChapter), -- Chapter and ScriptChapter stripped of bold / italic / accept-
as-written markup
                ['Degree'] = Degree;
```

```
-- cite thesis only
                ['Title'] = metadata.make_coins_title (coins_title, ScriptTitle),
-- Title and ScriptTitle stripped of bold / italic / accept-as-written markup
                ['PublicationPlace'] = PublicationPlace,
                ['Date'] = COinS_date.rftdate,
-- COinS_date has correctly formatted date if Date is valid;
                ['Season'] = COinS_date.rftssn,
                ['Quarter'] = COinS_date.rftquarter,
                ['Chron'] =  COinS_date.rftchron or (not COinS_date.rftdate and
Date) or '',    -- chron but if not set and invalid date format use Date; keep
this last bit?
                ['Series'] = Series,
                ['Volume'] = Volume,
                ['Issue'] = Issue,
                ['Pages'] = coins_pages or metadata.get_coins_pages (first_set
({Sheet, Sheets, Page, Pages, At, QuotePage, QuotePages}, 7)),  -- pages stripped
of external links
                ['Edition'] = Edition,
                ['PublisherName'] = PublisherName or Newsgroup,
-- any apostrophe markup already removed from PublisherName
                ['URL'] = first_set ({ChapterURL, URL}, 2),
                ['Authors'] = coins_author,
                ['ID_list'] = ID_list_coins,
                ['RawPage'] = this_page.prefixedText,
        }, config.CitationClass);

        -- Account for the oddities that are {{cite arxiv}}, {{cite biorxiv}},
{{cite citeseerx}}, and {{cite ssrn}} AFTER generation of COinS data.
        if utilities.in_array (config.CitationClass,
whitelist.preprint_template_list) then  -- we have set rft.jtitle in COinS to
arXiv, bioRxiv, CiteSeerX, or ssrn now unset so it isn't displayed
                Periodical = '';
-- periodical not allowed in these templates; if article has been published, use
cite journal
        end

        -- special case for cite newsgroup.  Do this after COinS because we are
modifying Publishername to include some static text
        if 'newsgroup' == config.CitationClass and utilities.is_set (Newsgroup)
then
                PublisherName = utilities.substitute (cfg.messages['newsgroup'],
external_link( 'news:' .. Newsgroup, Newsgroup, Newsgroup_origin, nil ));
        end

        local Editors;
        local EditorCount;
-- used only for choosing {ed.) or (eds.) annotation at end of editor name-list
        local Contributors;
-- assembled contributors name list
        local contributor_etal;
        local Translators;
-- assembled translators name list
        local translator_etal;
        local t = {};
-- translators list from |translator-lastn= / translator-firstn= pairs
        t = extract_names (args, 'TranslatorList');
-- fetch translator list from |translatorn= / |translator-lastn=, -firstn=, -
linkn=, -maskn=
        local Interviewers;
        local interviewers_list = {};
        interviewers_list = extract_names (args, 'InterviewerList');
-- process preferred interviewers parameters
```

```
        local interviewer_etal;

        -- Now perform various field substitutions.
        -- We also add leading spaces and surrounding markup and punctuation to
the
        -- various parts of the citation, but only when they are non-nil.
        do
                local last_first_list;
                local control = {
                        format = NameListStyle,
-- empty string or 'vanc'
                        maximum = nil,
-- as if display-authors or display-editors not set
                        mode = Mode
                };

                do
-- do editor name list first because the now unsupported coauthors used to modify
control table
                        control.maximum , editor_etal = get_display_names
(A['DisplayEditors'], #e, 'editors', editor_etal, A:ORIGIN ('DisplayEditors'));
                        Editors, EditorCount = list_people (control, e,
editor_etal);

                        if 1 == EditorCount and (true == editor_etal or 1 < #e)
then            -- only one editor displayed but includes etal then
                                EditorCount = 2;
-- spoof to display (eds.) annotation
                        end
                end
                do
-- now do interviewers
                        control.maximum, interviewer_etal = get_display_names
(A['DisplayInterviewers'], #interviewers_list, 'interviewers', interviewer_etal,
A:ORIGIN ('DisplayInterviewers'));
                        Interviewers = list_people (control, interviewers_list,
interviewer_etal);
                end
                do
-- now do translators
                        control.maximum, translator_etal = get_display_names
(A['DisplayTranslators'], #t, 'translators', translator_etal, A:ORIGIN
('DisplayTranslators'));
                        Translators = list_people (control, t, translator_etal);
                end
                do
-- now do contributors
                        control.maximum, contributor_etal = get_display_names
(A['DisplayContributors'], #c, 'contributors', contributor_etal, A:ORIGIN
('DisplayContributors'));
                        Contributors = list_people (control, c,
contributor_etal);
                end
                do
-- now do authors
                        control.maximum, author_etal = get_display_names
(A['DisplayAuthors'], #a, 'authors', author_etal, A:ORIGIN ('DisplayAuthors'));

                        last_first_list = list_people (control, a, author_etal);

                        if utilities.is_set (Authors) then
                                Authors, author_etal = name_has_etal (Authors,
```

```
author_etal, false, 'authors'); -- find and remove variations on et al.
                                if author_etal then
                                        Authors = Authors .. ' ' ..
cfg.messages['et al'];                  -- add et al. to authors parameter
                                end
                        else
                                Authors = last_first_list;
-- either an author name list or an empty string
                        end
                end
-- end of do

                if utilities.is_set (Authors) and utilities.is_set
(Collaboration) then
                        Authors = Authors .. ' (' .. Collaboration .. ')';
-- add collaboration after et al.
                end

        end

        local ConferenceFormat = A['ConferenceFormat'];
        local ConferenceURL = A['ConferenceURL'];
        ConferenceFormat = style_format (ConferenceFormat, ConferenceURL,
'conference-format', 'conference-url');
        Format = style_format (Format, URL, 'format', 'url');

        -- special case for chapter format so no error message or cat when
chapter not supported
        if not (utilities.in_array (config.CitationClass, {'web', 'news',
'journal', 'magazine', 'pressrelease', 'podcast', 'newsgroup', 'arxiv',
'biorxiv', 'citeseerx', 'ssrn'}) or
                ('citation' == config.CitationClass and (utilities.is_set
(Periodical) or utilities.is_set (ScriptPeriodical)) and not utilities.is_set
(Encyclopedia))) then
                        ChapterFormat = style_format (ChapterFormat, ChapterURL,
'chapter-format', 'chapter-url');
        end

        if not utilities.is_set (URL) then
                if utilities.in_array (config.CitationClass, {"web", "podcast",
"mailinglist"}) or              -- |url= required for cite web, cite podcast, and
cite mailinglist
                        ('citation' == config.CitationClass and ('website' ==
Periodical_origin or 'script-website' == ScriptPeriodical_origin)) then -- and
required for {{citation}} with |website= or |script-website=
                                utilities.set_message ('err_cite_web_url');
                end

                -- do we have |accessdate= without either |url= or |chapter-url=?
                if utilities.is_set (AccessDate) and not utilities.is_set
(ChapterURL) then               -- ChapterURL may be set when URL is not set;
                        utilities.set_message ('err_accessdate_missing_url');
                        AccessDate = '';
                end
        end

        local UrlStatus = is_valid_parameter_value (A['UrlStatus'],
A:ORIGIN('UrlStatus'), cfg.keywords_lists['url-status'], '');
        local OriginalURL
        local OriginalURL_origin
        local OriginalFormat
        local OriginalAccess;
```

```
        UrlStatus = UrlStatus:lower();
-- used later when assembling archived text
        if utilities.is_set ( ArchiveURL ) then
                if utilities.is_set (ChapterURL) then
-- if chapter-url= is set apply archive url to it
                        OriginalURL = ChapterURL;
-- save copy of source chapter's url for archive text
                        OriginalURL_origin = ChapterURL_origin;
-- name of |chapter-url= parameter for error messages
                        OriginalFormat = ChapterFormat;
-- and original |chapter-format=

                        if 'live' ~= UrlStatus then
                                ChapterURL = ArchiveURL
-- swap-in the archive's URL
                                ChapterURL_origin = A:ORIGIN('ArchiveURL')
-- name of |archive-url= parameter for error messages
                                ChapterFormat = ArchiveFormat or '';
-- swap in archive's format
                                ChapterUrlAccess = nil;
-- restricted access levels do not make sense for archived URLs
                        end
                elseif utilities.is_set (URL) then
                        OriginalURL = URL;
-- save copy of original source URL
                        OriginalURL_origin = URL_origin;
-- name of URL parameter for error messages
                        OriginalFormat = Format;
-- and original |format=
                        OriginalAccess = UrlAccess;

                        if 'live' ~= UrlStatus then
-- if URL set then |archive-url= applies to it
                                URL = ArchiveURL
-- swap-in the archive's URL
                                URL_origin = A:ORIGIN('ArchiveURL')
-- name of archive URL parameter for error messages
                                Format = ArchiveFormat or '';
-- swap in archive's format
                                UrlAccess = nil;
-- restricted access levels do not make sense for archived URLs
                        end
                end
        elseif utilities.is_set (UrlStatus) then
-- if |url-status= is set when |archive-url= is not set
                utilities.set_message ('maint_url_status');
-- add maint cat
        end

        if utilities.in_array (config.CitationClass, {'web', 'news', 'journal',
'magazine', 'pressrelease', 'podcast', 'newsgroup', 'arxiv', 'biorxiv',
'citeseerx', 'ssrn'}) or        -- if any of the 'periodical' cites except
encyclopedia
                ('citation' == config.CitationClass and (utilities.is_set
(Periodical) or utilities.is_set (ScriptPeriodical)) and not utilities.is_set
(Encyclopedia)) then
                        local chap_param;
                        if utilities.is_set (Chapter) then
-- get a parameter name from one of these chapter related meta-parameters
                                chap_param = A:ORIGIN ('Chapter')
                        elseif utilities.is_set (TransChapter) then
                                chap_param = A:ORIGIN ('TransChapter')
```

```
                        elseif utilities.is_set (ChapterURL) then
                                chap_param = A:ORIGIN ('ChapterURL')
                        elseif utilities.is_set (ScriptChapter) then
                                chap_param = ScriptChapter_origin;
                        else utilities.is_set (ChapterFormat)
                                chap_param = A:ORIGIN ('ChapterFormat')
                        end

                        if utilities.is_set (chap_param) then
-- if we found one
                                utilities.set_message ('err_chapter_ignored',
{chap_param});  -- add error message
                                Chapter = '';
-- and set them to empty string to be safe with concatenation
                                TransChapter = '';
                                ChapterURL = '';
                                ScriptChapter = '';
                                ChapterFormat = '';
                        end
        else
-- otherwise, format chapter / article title
                local no_quotes = false;
-- default assume that we will be quoting the chapter parameter value
                if utilities.is_set (Contribution) and 0 < #c then
-- if this is a contribution with contributor(s)
                        if utilities.in_array (Contribution:lower(),
cfg.keywords_lists.contribution) then   -- and a generic contribution title
                                no_quotes = true;
-- then render it unquoted
                        end
                end

                Chapter = format_chapter_title (ScriptChapter,
ScriptChapter_origin, Chapter, Chapter_origin, TransChapter, TransChapter_origin,
ChapterURL, ChapterURL_origin, no_quotes, ChapterUrlAccess);            --
Contribution is also in Chapter
                if utilities.is_set (Chapter) then
                        Chapter = Chapter .. ChapterFormat ;
                        if 'map' == config.CitationClass and utilities.is_set
(TitleType) then
                                Chapter = Chapter .. ' ' .. TitleType;
-- map annotation here; not after title
                        end
                        Chapter = Chapter .. sepc .. ' ';
                elseif utilities.is_set (ChapterFormat) then
-- |chapter= not set but |chapter-format= is so ...
                        Chapter = ChapterFormat .. sepc .. ' ';
-- ... ChapterFormat has error message, we want to see it
                end
        end

        -- Format main title
        local plain_title = false;
        local accept_title;
        Title, accept_title = utilities.has_accept_as_written (Title, true);
-- remove accept-this-as-written markup when it wraps all of <Title>
        if accept_title and ('' == Title) then
-- only support forced empty for now "(())"
                Title = cfg.messages['notitle'];
-- replace by predefined "No title" message
                        -- TODO: utilities.set_message (
'err_redundant_parameters', ...);       -- issue proper error message instead of
```

```
muting
                        ScriptTitle = '';
-- just mute for now
                        TransTitle = '';
-- just mute for now
                plain_title = true;
-- suppress text decoration for descriptive title
                utilities.set_message ('maint_untitled');
-- add maint cat
        end

        if not accept_title then
-- <Title> not wrapped in accept-as-written markup
                if '...' == Title:sub (-3) then
-- if ellipsis is the last three characters of |title=
                        Title = Title:gsub ('(%.%.%.)%.+$', '%1');
-- limit the number of dots to three
                elseif not mw.ustring.find (Title, '%.%s*%a%.$') and
-- end of title is not a 'dot-(optional space-)letter-dot' initialism ...
                        not mw.ustring.find (Title, '%s+%a%.$') then
-- ...and not a 'space-letter-dot' initial (''Allium canadense'' L.)
                                Title = mw.ustring.gsub(Title, '%' .. sepc ..
'$', '');                       -- remove any trailing separator character; sepc
and ms.ustring() here for languages that use multibyte separator characters
                end

                if utilities.is_set (ArchiveURL) and is_archived_copy (Title)
then
                        utilities.set_message ('maint_archived_copy');
-- add maintenance category before we modify the content of Title
                end

                if is_generic ('generic_titles', Title) then
                        utilities.set_message ('err_generic_title');
-- set an error message
                end
        end

        if (not plain_title) and (utilities.in_array (config.CitationClass,
{'web', 'news', 'journal', 'magazine', 'pressrelease', 'podcast', 'newsgroup',
'mailinglist', 'interview', 'arxiv', 'biorxiv', 'citeseerx', 'ssrn'}) or
                ('citation' == config.CitationClass and (utilities.is_set
(Periodical) or utilities.is_set (ScriptPeriodical)) and not utilities.is_set
(Encyclopedia)) or
                ('map' == config.CitationClass and (utilities.is_set (Periodical)
or utilities.is_set (ScriptPeriodical)))) then          -- special case for cite
map when the map is in a periodical treat as an article
                        Title = kern_quotes (Title);
-- if necessary, separate title's leading and trailing quote marks from module
provided quote marks
                        Title = utilities.wrap_style ('quoted-title', Title);
                        Title = script_concatenate (Title, ScriptTitle, 'script-
title');        -- <bdi> tags, lang attribute, categorization, etc.; must be done
after title is wrapped
                        TransTitle = utilities.wrap_style ('trans-quoted-title',
TransTitle );
        elseif plain_title or ('report' == config.CitationClass) then
-- no styling for cite report and descriptive titles (otherwise same as above)
                Title = script_concatenate (Title, ScriptTitle, 'script-title');
-- <bdi> tags, lang attribute, categorization, etc.; must be done after title is
wrapped
                TransTitle = utilities.wrap_style ('trans-quoted-title',
```

```
TransTitle );   -- for cite report, use this form for trans-title
        else
                Title = utilities.wrap_style ('italic-title', Title);
                Title = script_concatenate (Title, ScriptTitle, 'script-title');
-- <bdi> tags, lang attribute, categorization, etc.; must be done after title is
wrapped
                TransTitle = utilities.wrap_style ('trans-italic-title',
TransTitle);
        end

        if utilities.is_set (TransTitle) then
                if utilities.is_set (Title) then
                        TransTitle = " " .. TransTitle;
                else
                        utilities.set_message ('err_trans_missing_title',
{'title'});
                end
        end

        if utilities.is_set (Title) then
-- TODO: is this the right place to be making Wikisource URLs?
                if utilities.is_set (TitleLink) and utilities.is_set (URL) then
                        utilities.set_message ('err_wikilink_in_url');
-- set an error message because we can't have both
                        TitleLink = '';
-- unset
                end

                if not utilities.is_set (TitleLink) and utilities.is_set (URL)
then
                        Title = external_link (URL, Title, URL_origin, UrlAccess)
.. TransTitle .. Format;
                        URL = '';
-- unset these because no longer needed
                        Format = "";
                elseif utilities.is_set (TitleLink) and not utilities.is_set
(URL) then
                        local ws_url;
                        ws_url = wikisource_url_make (TitleLink);
-- ignore ws_label return; not used here
                        if ws_url then
                                Title = external_link (ws_url, Title .. ' ',
'ws link in title-link');       -- space char after Title to move icon away from
italic text; TODO: a better way to do this?
                                Title = utilities.substitute
(cfg.presentation['interwiki-icon'], {cfg.presentation['class-wikisource'],
TitleLink, Title});
                                Title = Title .. TransTitle;
                        else
                                Title = utilities.make_wikilink (TitleLink,
Title) .. TransTitle;
                        end
                else
                        local ws_url, ws_label, L;
-- Title has italic or quote markup by the time we get here which causes
is_wikilink() to return 0 (not a wikilink)
                        ws_url, ws_label, L = wikisource_url_make
(Title:gsub('^[\'"]*(.-)[\'"]*$', '%1'));       -- make ws URL from |title=
interwiki link (strip italic or quote markup); link portion L becomes tooltip
label
                        if ws_url then
                                Title = Title:gsub ('%b[]', ws_label);
```

```
-- replace interwiki link with ws_label to retain markup
                                Title = external_link (ws_url, Title .. ' ',
'ws link in title');    -- space char after Title to move icon away from italic
text; TODO: a better way to do this?
                                Title = utilities.substitute
(cfg.presentation['interwiki-icon'], {cfg.presentation['class-wikisource'], L,
Title});
                                Title = Title .. TransTitle;
                        else
                                Title = Title .. TransTitle;
                        end
                end
        else
                Title = TransTitle;
        end

        if utilities.is_set (Place) then
                Place = " " .. wrap_msg ('written', Place, use_lowercase) .. sepc
.. " ";
        end

        local ConferenceURL_origin = A:ORIGIN('ConferenceURL');
-- get name of parameter that holds ConferenceURL
        if utilities.is_set (Conference) then
                if utilities.is_set (ConferenceURL) then
                        Conference = external_link( ConferenceURL, Conference,
ConferenceURL_origin, nil );
                end
                Conference = sepc .. " " .. Conference .. ConferenceFormat;
        elseif utilities.is_set (ConferenceURL) then
                Conference = sepc .. " " .. external_link( ConferenceURL, nil,
ConferenceURL_origin, nil );
        end

        local Position = '';
        if not utilities.is_set (Position) then
                local Minutes = A['Minutes'];
                local Time = A['Time'];

                if utilities.is_set (Minutes) then
                        if utilities.is_set (Time) then         --TODO: make a
function for this and similar?
                                utilities.set_message
('err_redundant_parameters', {utilities.wrap_style ('parameter', 'minutes') .. '
and ' .. utilities.wrap_style ('parameter', 'time')});
                        end
                        Position = " " .. Minutes .. " " ..
cfg.messages['minutes'];
                else
                        if utilities.is_set (Time) then
                                local TimeCaption = A['TimeCaption']
                                if not utilities.is_set (TimeCaption) then
                                        TimeCaption = cfg.messages['event'];
                                        if sepc ~= '.' then
                                                TimeCaption =
TimeCaption:lower();
                                        end
                                end
                                Position = " " .. TimeCaption .. " " .. Time;
                        end
                end
        else
```

```
                Position = " " .. Position;
                At = '';
        end

        Page, Pages, Sheet, Sheets = format_pages_sheets (Page, Pages, Sheet,
Sheets, config.CitationClass, Periodical_origin, sepc, NoPP, use_lowercase);

        At = utilities.is_set (At) and (sepc .. " " .. At) or "";
        Position = utilities.is_set (Position) and (sepc .. " " .. Position) or
"";
        if config.CitationClass == 'map' then
                local Sections = A['Sections'];
-- Section (singular) is an alias of Chapter so set earlier
                local Inset = A['Inset'];

                if utilities.is_set ( Inset ) then
                        Inset = sepc .. " " .. wrap_msg ('inset', Inset,
use_lowercase);
                end

                if utilities.is_set ( Sections ) then
                        Section = sepc .. " " .. wrap_msg ('sections', Sections,
use_lowercase);
                elseif utilities.is_set ( Section ) then
                        Section = sepc .. " " .. wrap_msg ('section', Section,
use_lowercase);
                end
                At = At .. Inset .. Section;
        end

        local Others = A['Others'];
        if utilities.is_set (Others) and 0 == #a and 0 == #e then
-- add maint cat when |others= has value and used without |author=, |editor=
                if config.CitationClass == "AV-media-notes"
                or config.CitationClass == "audio-visual" then
-- special maint for AV/M which has a lot of 'false' positives right now
                        utilities.set_message ('maint_others_avm')
                else
                        utilities.set_message ('maint_others');
                end
        end
        Others = utilities.is_set (Others) and (sepc .. " " .. Others) or "";

        if utilities.is_set (Translators) then
                Others = safe_join ({sepc .. ' ', wrap_msg ('translated',
Translators, use_lowercase), Others}, sepc);
        end
        if utilities.is_set (Interviewers) then
                Others = safe_join ({sepc .. ' ', wrap_msg ('interview',
Interviewers, use_lowercase), Others}, sepc);
        end

        local TitleNote = A['TitleNote'];
        TitleNote = utilities.is_set (TitleNote) and (sepc .. " " .. TitleNote)
or "";
        if utilities.is_set (Edition) then
                if Edition:match ('%f[%a][Ee]d%n?%.?$') or Edition:match ('%f[%a]
[Ee]dition$') then -- Ed, ed, Ed., ed., Edn, edn, Edn., edn.
                        utilities.set_message ('err_extra_text_edition');
-- add error message
                end
                Edition = " " .. wrap_msg ('edition', Edition);
```

```
        else
                Edition = '';
        end

        Series = utilities.is_set (Series) and wrap_msg ('series', {sepc,
Series}) or ""; -- not the same as SeriesNum
        local Agency = A['Agency'];
        Agency = utilities.is_set (Agency) and wrap_msg ('agency', {sepc,
Agency}) or "";
        Volume = format_volume_issue (Volume, Issue, config.CitationClass,
Periodical_origin, sepc, use_lowercase);

        if utilities.is_set (AccessDate) then
                local retrv_text = " " .. cfg.messages['retrieved']

                AccessDate = nowrap_date (AccessDate);
-- wrap in nowrap span if date in appropriate format
                if (sepc ~= ".") then retrv_text = retrv_text:lower() end
-- if mode is cs2, lower case
                AccessDate = utilities.substitute (retrv_text, AccessDate);
-- add retrieved text

                AccessDate = utilities.substitute
(cfg.presentation['accessdate'], {sepc, AccessDate});   -- allow editors to hide
accessdates
        end

        if utilities.is_set (ID) then ID = sepc .. " " .. ID; end

        local Docket = A['Docket'];
        if "thesis" == config.CitationClass and utilities.is_set (Docket) then
                ID = sepc .. " Docket " .. Docket .. ID;
        end
        if "report" == config.CitationClass and utilities.is_set (Docket) then
-- for cite report when |docket= is set
                ID = sepc .. ' ' .. Docket;
-- overwrite ID even if |id= is set
        end

        if utilities.is_set (URL) then
                URL = " " .. external_link( URL, nil, URL_origin, UrlAccess );
        end

        local Quote = A['Quote'];
        local TransQuote = A['TransQuote'];
        local ScriptQuote = A['ScriptQuote'];
        if utilities.is_set (Quote) or utilities.is_set (TransQuote) or
utilities.is_set (ScriptQuote) then

                if utilities.is_set (Quote) then
                        if Quote:sub(1, 1) == '"' and Quote:sub(-1, -1) == '"'
then                    -- if first and last characters of quote are quote marks
                                Quote = Quote:sub(2, -2);
-- strip them off
                        end
                end

                Quote = utilities.wrap_style ('quoted-text', Quote );
-- wrap in <q>...</q> tags

                if utilities.is_set (ScriptQuote) then
                        Quote = script_concatenate (Quote, ScriptQuote, 'script-
```

```
quote');        -- <bdi> tags, lang attribute, categorization, etc.; must be done
after quote is wrapped
                end

                if utilities.is_set (TransQuote) then
                        if TransQuote:sub(1, 1) == '"' and TransQuote:sub(-1, -1)
== '"' then -- if first and last characters of |trans-quote are quote marks
                                TransQuote = TransQuote:sub(2, -2); -- strip them
off
                        end
                        Quote = Quote .. " " .. utilities.wrap_style ('trans-
quoted-title', TransQuote );
                end

                if utilities.is_set (QuotePage) or utilities.is_set (QuotePages)
then    -- add page prefix
                        local quote_prefix = '';
                        if utilities.is_set (QuotePage) then
                                extra_text_in_page_check (QuotePage, 'quote-
page');                         -- add to maint cat if |quote-page= value begins
with what looks like p., pp., etc.
                                if not NoPP then
                                        quote_prefix = utilities.substitute
(cfg.messages['p-prefix'], {sepc, QuotePage}), '', '', '';
                                else
                                        quote_prefix = utilities.substitute
(cfg.messages['nopp'], {sepc, QuotePage}), '', '', '';
                                end
                        elseif utilities.is_set (QuotePages) then
                                extra_text_in_page_check (QuotePages, 'quote-
pages');                        -- add to maint cat if |quote-pages= value begins
with what looks like p., pp., etc.
                                if tonumber(QuotePages) ~= nil and not NoPP then
-- if only digits, assume single page
                                        quote_prefix = utilities.substitute
(cfg.messages['p-prefix'], {sepc, QuotePages}), '', '';
                                elseif not NoPP then
                                        quote_prefix = utilities.substitute
(cfg.messages['pp-prefix'], {sepc, QuotePages}), '', '';
                                else
                                        quote_prefix = utilities.substitute
(cfg.messages['nopp'], {sepc, QuotePages}), '', '';
                                end
                        end

                        Quote = quote_prefix .. ": " .. Quote;
                else
                        Quote = sepc .. " " .. Quote;
                end

                PostScript = "";
-- cs1|2 does not supply terminal punctuation when |quote= is set
        end

        -- We check length of PostScript here because it will have been nuked by
        -- the quote parameters. We'd otherwise emit a message even if there
wasn't
        -- a displayed postscript.
        -- TODO: Should the max size (1) be configurable?
        -- TODO: Should we check a specific pattern?
        if utilities.is_set(PostScript) and mw.ustring.len(PostScript) > 1 then
                utilities.set_message ('maint_postscript')
```

```
        end

        local Archived;
        if utilities.is_set (ArchiveURL) then
                local arch_text;
                if not utilities.is_set (ArchiveDate) then
                        utilities.set_message ('err_archive_missing_date');
                        ArchiveDate = '';
-- empty string for concatenation
                end
                if "live" == UrlStatus then
                        arch_text = cfg.messages['archived'];
                        if sepc ~= "." then arch_text = arch_text:lower() end
                        if utilities.is_set (ArchiveDate) then
                                Archived = sepc .. ' ' .. utilities.substitute (
cfg.messages['archived-live'],
                                        {external_link( ArchiveURL, arch_text,
A:ORIGIN('ArchiveURL'), nil) .. ArchiveFormat, ArchiveDate } );
                        else
                                Archived = '';
                        end
                        if not utilities.is_set (OriginalURL) then
                                utilities.set_message
('err_archive_missing_url');
                                Archived = '';
-- empty string for concatenation
                        end
                elseif utilities.is_set (OriginalURL) then
-- UrlStatus is empty, 'dead', 'unfit', 'usurped', 'bot: unknown'
                        if utilities.in_array (UrlStatus, {'unfit', 'usurped',
'bot: unknown'}) then
                                arch_text = cfg.messages['archived-unfit'];
                                if sepc ~= "." then arch_text = arch_text:lower()
end
                                Archived = sepc .. ' ' .. arch_text ..
ArchiveDate;                            -- format already styled
                                if 'bot: unknown' == UrlStatus then
                                        utilities.set_message
('maint_bot_unknown');                          -- and add a category if not
already added
                                else
                                        utilities.set_message ('maint_unfit');
-- and add a category if not already added
                                end
                        else
-- UrlStatus is empty, 'dead'
                                arch_text = cfg.messages['archived-dead'];
                                if sepc ~= "." then arch_text = arch_text:lower()
end
                                if utilities.is_set (ArchiveDate) then
                                        Archived = sepc .. " " ..
utilities.substitute ( arch_text,
                                                { external_link( OriginalURL,
cfg.messages['original'], OriginalURL_origin, OriginalAccess ) .. OriginalFormat,
ArchiveDate } );        -- format already styled
                                else
                                        Archived = '';
-- unset for concatenation
                                end
                        end
                else
-- OriginalUrl not set
```

```
                        arch_text = cfg.messages['archived-missing'];
                        if sepc ~= "." then arch_text = arch_text:lower() end
                        utilities.set_message ('err_archive_missing_url');
                        Archived = '';
-- empty string for concatenation
                end
        elseif utilities.is_set (ArchiveFormat) then
                Archived = ArchiveFormat;
-- if set and ArchiveURL not set ArchiveFormat has error message
        else
                Archived = '';
        end

        local Lay = '';
        local LaySource = A['LaySource'];
        local LayURL = A['LayURL'];
        local LayFormat = A['LayFormat'];
        LayFormat = style_format (LayFormat, LayURL, 'lay-format', 'lay-url');
        if utilities.is_set (LayURL) then
                if utilities.is_set (LayDate) then LayDate = " (" .. LayDate ..
")" end
                if utilities.is_set (LaySource) then
                        LaySource = " &ndash; ''" .. utilities.safe_for_italics
(LaySource) .. "''";
                else
                        LaySource = "";
                end
                if sepc == '.' then
                        Lay = sepc .. " " .. external_link( LayURL,
cfg.messages['lay summary'], A:ORIGIN('LayURL'), nil ) .. LayFormat .. LaySource
.. LayDate
                else
                        Lay = sepc .. " " .. external_link( LayURL,
cfg.messages['lay summary']:lower(), A:ORIGIN('LayURL'), nil ) .. LayFormat ..
LaySource .. LayDate
                end
        elseif utilities.is_set (LayFormat) then
-- Test if |lay-format= is given without giving a |lay-url=
                Lay = sepc .. LayFormat;
-- if set and LayURL not set, then LayFormat has error message
        end

        local TranscriptURL = A['TranscriptURL']
        local TranscriptFormat = A['TranscriptFormat'];
        TranscriptFormat = style_format (TranscriptFormat, TranscriptURL,
'transcript-format', 'transcripturl');
        local Transcript = A['Transcript'];
        local TranscriptURL_origin = A:ORIGIN('TranscriptURL');
-- get name of parameter that holds TranscriptURL
        if utilities.is_set (Transcript) then
                if utilities.is_set (TranscriptURL) then
                        Transcript = external_link( TranscriptURL, Transcript,
TranscriptURL_origin, nil );
                end
                Transcript = sepc .. ' ' .. Transcript .. TranscriptFormat;
        elseif utilities.is_set (TranscriptURL) then
                Transcript = external_link( TranscriptURL, nil,
TranscriptURL_origin, nil );
        end

        local Publisher;
        if utilities.is_set (PublicationDate) then
```

```
                PublicationDate = wrap_msg ('published', PublicationDate);
        end
        if utilities.is_set (PublisherName) then
                if utilities.is_set (PublicationPlace) then
                        Publisher = sepc .. " " .. PublicationPlace .. ": " ..
PublisherName .. PublicationDate;
                else
                        Publisher = sepc .. " " .. PublisherName ..
PublicationDate;
                end
        elseif utilities.is_set (PublicationPlace) then
                Publisher= sepc .. " " .. PublicationPlace .. PublicationDate;
        else
                Publisher = PublicationDate;
        end

        local TransPeriodical =  A['TransPeriodical'];
        local TransPeriodical_origin =  A:ORIGIN ('TransPeriodical');
        -- Several of the above rely upon detecting this as nil, so do it last.
        if (utilities.is_set (Periodical) or utilities.is_set (ScriptPeriodical)
or utilities.is_set (TransPeriodical)) then
                if utilities.is_set (Title) or utilities.is_set (TitleNote) then
                        Periodical = sepc .. " " .. format_periodical
(ScriptPeriodical, ScriptPeriodical_origin, Periodical, TransPeriodical,
TransPeriodical_origin);
                else
                        Periodical = format_periodical (ScriptPeriodical,
ScriptPeriodical_origin, Periodical, TransPeriodical, TransPeriodical_origin);
                end
        end

        local Language = A['Language'];
        if utilities.is_set (Language) then
                Language = language_parameter (Language);
-- format, categories, name from ISO639-1, etc.
        else
                Language='';
-- language not specified so make sure this is an empty string;
        --[[ TODO: need to extract the wrap_msg from language_parameter
        so that we can solve parentheses bunching problem with
Format/Language/TitleType
        ]]
        end

        --[[
        Handle the oddity that is cite speech.  This code overrides whatever may
be the value assigned to TitleNote (through |department=) and forces it to be "
(Speech)" so that
        the annotation directly follows the |title= parameter value in the
citation rather than the |event= parameter value (if provided).
        ]]
        if "speech" == config.CitationClass then
-- cite speech only
                TitleNote = TitleType;
-- move TitleType to TitleNote so that it renders ahead of |event=
                TitleType = '';
-- and unset

                if utilities.is_set (Periodical) then
-- if Periodical, perhaps because of an included |website= or |journal= parameter
                        if utilities.is_set (Conference) then
-- and if |event= is set
```

```
                                Conference = Conference .. sepc .. " ";
-- then add appropriate punctuation to the end of the Conference variable before
rendering
                        end
                end
        end

        -- Piece all bits together at last.  Here, all should be non-nil.
        -- We build things this way because it is more efficient in LUA
        -- not to keep reassigning to the same string variable over and over.

        local tcommon;
        local tcommon2;
-- used for book cite when |contributor= is set

        if utilities.in_array (config.CitationClass, {"journal", "citation"}) and
utilities.is_set (Periodical) then
                if utilities.is_set (Others) then Others = safe_join ({Others,
sepc .. " "}, sepc) end         -- add terminal punctuation & space; check for
dup sepc; TODO why do we need to do this here?
                tcommon = safe_join( {Others, Title, TitleNote, Conference,
Periodical, Format, TitleType, Series, Language, Edition, Publisher, Agency,
Volume}, sepc );
        elseif utilities.in_array (config.CitationClass, {"book", "citation"})
and not utilities.is_set (Periodical) then              -- special cases for book
cites
                if utilities.is_set (Contributors) then
-- when we are citing foreword, preface, introduction, etc.
                        tcommon = safe_join( {Title, TitleNote}, sepc );
-- author and other stuff will come after this and before tcommon2
                        tcommon2 = safe_join( {Conference, Periodical, Format,
TitleType, Series, Language, Volume, Others, Edition, Publisher, Agency}, sepc );
                else
                        tcommon = safe_join( {Title, TitleNote, Conference,
Periodical, Format, TitleType, Series, Language, Volume, Others, Edition,
Publisher, Agency}, sepc );
                end

        elseif 'map' == config.CitationClass then
-- special cases for cite map
                if utilities.is_set (Chapter) then
-- map in a book; TitleType is part of Chapter
                        tcommon = safe_join( {Title, Format, Edition, Scale,
Series, Language, Cartography, Others, Publisher, Volume}, sepc );
                elseif utilities.is_set (Periodical) then
-- map in a periodical
                        tcommon = safe_join( {Title, TitleType, Format,
Periodical, Scale, Series, Language, Cartography, Others, Publisher, Volume},
sepc );
                else
-- a sheet or stand-alone map
                        tcommon = safe_join( {Title, TitleType, Format, Edition,
Scale, Series, Language, Cartography, Others, Publisher}, sepc );
                end

        elseif 'episode' == config.CitationClass then
-- special case for cite episode
                tcommon = safe_join( {Title, TitleNote, TitleType, Series,
Language, Edition, Publisher}, sepc );

        else
-- all other CS1 templates
```

```
                tcommon = safe_join( {Title, TitleNote, Conference, Periodical,
Format, TitleType, Series, Language,
                        Volume, Others, Edition, Publisher, Agency}, sepc );
        end

        if #ID_list > 0 then
                ID_list = safe_join( { sepc .. " ",  table.concat( ID_list, sepc
.. " " ), ID }, sepc );
        else
                ID_list = ID;
        end

        local Via = A['Via'];
        Via = utilities.is_set (Via) and  wrap_msg ('via', Via) or '';
        local idcommon;
        if 'audio-visual' == config.CitationClass or 'episode' ==
config.CitationClass then       -- special case for cite AV media & cite episode
position transcript
                idcommon = safe_join( { ID_list, URL, Archived, Transcript,
AccessDate, Via, Lay, Quote }, sepc );
        else
                idcommon = safe_join( { ID_list, URL, Archived, AccessDate, Via,
Lay, Quote }, sepc );
        end

        local text;
        local pgtext = Position .. Sheet .. Sheets .. Page .. Pages .. At;

        local OrigDate = A['OrigDate'];
        OrigDate = utilities.is_set (OrigDate) and wrap_msg ('origdate',
OrigDate) or '';
        if utilities.is_set (Date) then
                if utilities.is_set (Authors) or utilities.is_set (Editors) then
-- date follows authors or editors when authors not set
                        Date = " (" .. Date .. ")" .. OrigDate .. sepc .. " ";
-- in parentheses
                else
-- neither of authors and editors set
                        if (string.sub(tcommon, -1, -1) == sepc) then
-- if the last character of tcommon is sepc
                                Date = " " .. Date .. OrigDate;
-- Date does not begin with sepc
                        else
                                Date = sepc .. " " .. Date .. OrigDate;
-- Date begins with sepc
                        end
                end
        end
        if utilities.is_set (Authors) then
                if (not utilities.is_set (Date)) then
-- when date is set it's in parentheses; no Authors termination
                        Authors = terminate_name_list (Authors, sepc);
-- when no date, terminate with 0 or 1 sepc and a space
                end
                if utilities.is_set (Editors) then
                        local in_text = " ";
                        local post_text = "";
                        if utilities.is_set (Chapter) and 0 == #c then
                                in_text = in_text .. cfg.messages['in'] .. " "
                                if (sepc ~= '.') then
                                        in_text = in_text:lower()
-- lowercase for cs2
```

```
                                end
                        end
                        if EditorCount <= 1 then
                                post_text = " (" .. cfg.messages['editor'] ..
")";                            -- be consistent with no-author, no-date case
                        else
                                post_text = " (" .. cfg.messages['editors'] ..
")";
                        end
                        Editors = terminate_name_list (in_text .. Editors ..
post_text, sepc);       -- terminate with 0 or 1 sepc and a space
                end
                if utilities.is_set (Contributors) then
-- book cite and we're citing the intro, preface, etc.
                        local by_text = sepc .. ' ' .. cfg.messages['by'] .. ' ';
                        if (sepc ~= '.') then by_text = by_text:lower() end
-- lowercase for cs2
                        Authors = by_text .. Authors;
-- author follows title so tweak it here
                        if utilities.is_set (Editors) and utilities.is_set (Date)
then            -- when Editors make sure that Authors gets terminated
                                Authors = terminate_name_list (Authors, sepc);
-- terminate with 0 or 1 sepc and a space
                        end
                        if (not utilities.is_set (Date)) then
-- when date is set it's in parentheses; no Contributors termination
                                Contributors = terminate_name_list (Contributors,
sepc);          -- terminate with 0 or 1 sepc and a space
                        end
                        text = safe_join( {Contributors, Date, Chapter, tcommon,
Authors, Place, Editors, tcommon2, pgtext, idcommon }, sepc );
                else
                        text = safe_join( {Authors, Date, Chapter, Place,
Editors, tcommon, pgtext, idcommon }, sepc );
                end
        elseif utilities.is_set (Editors) then
                if utilities.is_set (Date) then
                        if EditorCount <= 1 then
                                Editors = Editors .. ", " ..
cfg.messages['editor'];
                        else
                                Editors = Editors .. ", " ..
cfg.messages['editors'];
                        end
                else
                        if EditorCount <= 1 then
                                Editors = Editors .. " (" ..
cfg.messages['editor'] .. ")" .. sepc .. " "
                        else
                                Editors = Editors .. " (" ..
cfg.messages['editors'] .. ")" .. sepc .. " "
                        end
                end
                text = safe_join( {Editors, Date, Chapter, Place, tcommon,
pgtext, idcommon}, sepc );
        else
                if utilities.in_array (config.CitationClass, {"journal",
"citation"}) and utilities.is_set (Periodical) then
                        text = safe_join( {Chapter, Place, tcommon, pgtext, Date,
idcommon}, sepc );
                else
                        text = safe_join( {Chapter, Place, tcommon, Date, pgtext,
```

```
idcommon}, sepc );
                end
        end

        if utilities.is_set (PostScript) and PostScript ~= sepc then
                text = safe_join( {text, sepc}, sepc );
-- Deals with italics, spaces, etc.
                text = text:sub(1, -sepc:len() - 1);
        end

        text = safe_join( {text, PostScript}, sepc );

        -- Now enclose the whole thing in a <cite> element
        local options_t = {};
        options_t.class = cite_class_attribute_make (config.CitationClass, Mode);

        local Ref = is_valid_parameter_value (A['Ref'], A:ORIGIN('Ref'),
cfg.keywords_lists['ref'], nil, true);  -- nil when |ref=harv; A['Ref'] else

        if 'none' ~= cfg.keywords_xlate[(Ref and Ref:lower()) or ''] then
                local namelist_t = {};
-- holds selected contributor, author, editor name list
                local year = first_set ({Year, anchor_year}, 2);
-- Year first for legacy citations and for YMD dates that require disambiguation

                if #c > 0 then
-- if there is a contributor list
                        namelist_t = c;
-- select it
                elseif #a > 0 then
-- or an author list
                        namelist_t = a;
                elseif #e > 0 then
-- or an editor list
                        namelist_t = e;
                end
                local citeref_id;
                if #namelist_t > 0 then
-- if there are names in namelist_t
                        citeref_id = make_citeref_id (namelist_t, year);
-- go make the CITEREF anchor
                        if mw.uri.anchorEncode (citeref_id) == ((Ref and
mw.uri.anchorEncode (Ref)) or '') then  -- Ref may already be encoded (by
{{sfnref}}) so citeref_id must be encoded before comparison
                                utilities.set_message
('maint_ref_duplicates_default');
                        end
                else
                        citeref_id = '';
-- unset
                end
                options_t.id = Ref or citeref_id;
        end

        if string.len (text:gsub('%b<>', '')) <= 2 then
-- remove html and html-like tags; then get length of what remains;
                z.error_cats_t = {};
-- blank the categories list
                z.error_msgs_t = {};
-- blank the error messages list
                OCinSoutput = nil;
-- blank the metadata string
```

```
                text = '';
-- blank the the citation
                utilities.set_message ('err_empty_citation');
-- set empty citation message and category
        end

        local render_t = {};
-- here we collect the final bits for concatenation into the rendered citation

        if utilities.is_set (options_t.id) then
-- here we wrap the rendered citation in <cite ...>...</cite> tags
                table.insert (render_t, utilities.substitute
(cfg.presentation['cite-id'], {mw.uri.anchorEncode(options_t.id),
mw.text.nowiki(options_t.class), text}));       -- when |ref= is set or when
there is a namelist
        else
                table.insert (render_t, utilities.substitute
(cfg.presentation['cite'], {mw.text.nowiki(options_t.class), text}));   -- when
|ref=none or when namelist_t empty and |ref= is missing or is empty
        end

        if OCinSoutput then
-- blanked when citation is 'empty' so don't bother to add boilerplate metadata
span
                table.insert (render_t, utilities.substitute
(cfg.presentation['ocins'], OCinSoutput));      -- format and append metadata to
the citation
        end

        local template_name = ('citation' == config.CitationClass) and 'citation'
or 'cite ' .. (cfg.citation_class_map_t[config.CitationClass] or
config.CitationClass);
        local template_link = '[[Template:' .. template_name .. '|' ..
template_name .. ']]';
        local msg_prefix = '<code class="cs1-code">{{' .. template_link .. '}}
</code>: ';

        if 0 ~= #z.error_msgs_t then
                mw.addWarning (utilities.substitute (cfg.messages.warning_msg_e,
template_link));

                table.insert (render_t, ' ');
-- insert a space between citation and its error messages
                table.sort (z.error_msgs_t);
-- sort the error messages list; sorting includes wrapping <span> and <code>
tags; hidden-error sorts ahead of visible-error

                local hidden = true;
-- presume that the only error messages emited by this template are hidden
                for _, v in ipairs (z.error_msgs_t) do
-- spin through the list of error messages
                        if v:find ('cs1-visible-error', 1, true) then
-- look for the visible error class name
                                hidden = false;
-- found one; so don't hide the error message prefix
                                break;
-- and done because no need to look further
                        end
                end

                z.error_msgs_t[1] = table.concat ({utilities.error_comment
(msg_prefix, hidden), z.error_msgs_t[1]});      -- add error message prefix to
```

```
first error message to prevent extraneous punctuation
                table.insert (render_t, table.concat (z.error_msgs_t, '; '));
-- make a big string of error messages and add it to the rendering
        end

        if 0 ~= #z.maint_cats_t then
                mw.addWarning (utilities.substitute (cfg.messages.warning_msg_m,
template_link));

                table.sort (z.maint_cats_t);
-- sort the maintenance messages list

                local maint_msgs_t = {};
-- here we collect all of the maint messages

                if 0 == #z.error_msgs_t then
-- if no error messages
                        table.insert (maint_msgs_t, msg_prefix);
-- insert message prefix in maint message livery
                end

                for _, v in ipairs( z.maint_cats_t ) do
-- append maintenance categories
                        table.insert (maint_msgs_t,
-- assemble new maint message and add it to the maint_msgs_t table
                                table.concat ({v, ' (', utilities.substitute
(cfg.messages[':cat wikilink'], v), ')'})
                                );
                end
                table.insert (render_t, utilities.substitute
(cfg.presentation['hidden-maint'], table.concat (maint_msgs_t, ' ')));  -- wrap
the group of maint messages with proper presentation and save
        end

        if not no_tracking_cats then
                for _, v in ipairs (z.error_cats_t) do
-- append error categories
                        table.insert (render_t, utilities.substitute
(cfg.messages['cat wikilink'], v));
                end
                for _, v in ipairs (z.maint_cats_t) do
-- append maintenance categories
                        table.insert (render_t, utilities.substitute
(cfg.messages['cat wikilink'], v));
                end
                for _, v in ipairs (z.prop_cats_t) do
-- append properties categories
                        table.insert (render_t, utilities.substitute
(cfg.messages['cat wikilink'], v));
                end
        end

        return table.concat (render_t);
-- make a big string and done
end


--[[--------------------------< V A L I D A T E >--------------------------------
------------------------------

Looks for a parameter's name in one of several whitelists.
```

```
Parameters in the whitelist can have three values:
        true - active, supported parameters
        false - deprecated, supported parameters
        nil - unsupported parameters

]]

local function validate (name, cite_class, empty)
        local name = tostring (name);
        local enum_name;
-- for enumerated parameters, is name with enumerator replaced with '#'
        local state;
        local function state_test (state, name)
-- local function to do testing of state values
                if true == state then return true; end
-- valid actively supported parameter
                if false == state then
                        if empty then return nil; end
-- empty deprecated parameters are treated as unknowns
                        deprecated_parameter (name);
-- parameter is deprecated but still supported
                        return true;
                end
                if 'tracked' == state then
                        local base_name = name:gsub ('%d', '');
-- strip enumerators from parameter names that have them to get the base name
                        utilities.add_prop_cat ('tracked-param', {base_name},
base_name);     -- add a properties category; <base_name> modifies <key>
                        return true;
                end
                return nil;
        end

        if name:find ('#') then
-- # is a cs1|2 reserved character so parameters with # not permitted
                return nil;
        end

        if utilities.in_array (cite_class, whitelist.preprint_template_list )
then    -- limited parameter sets allowed for these templates
                state = whitelist.limited_basic_arguments[name];
                if true == state_test (state, name) then return true; end

                state = whitelist.preprint_arguments[cite_class][name];
-- look in the parameter-list for the template identified by cite_class
                if true == state_test (state, name) then return true; end


-- limited enumerated parameters list
                enum_name = name:gsub("%d+", "#" );
-- replace digit(s) with # (last25 becomes last#) (mw.ustring because non-Western
'local' digits)
                state = whitelist.limited_numbered_arguments[enum_name];
                if true == state_test (state, name) then return true; end

                return false;
-- not supported because not found or name is set to nil
        end
-- end limited parameter-set templates

        if utilities.in_array (cite_class, whitelist.unique_param_template_list)
then    -- experiment for template-specific parameters for templates that accept
```

```
parameters from the basic argument list
                state = whitelist.unique_arguments[cite_class][name];
-- look in the template-specific parameter-lists for the template identified by
cite_class
                if true == state_test (state, name) then return true; end
        end
-- if here, fall into general validation

        state = whitelist.basic_arguments[name];
-- all other templates; all normal parameters allowed
        if true == state_test (state, name) then return true; end


-- all enumerated parameters allowed
        enum_name = name:gsub("%d+", "#" );
-- replace digit(s) with # (last25 becomes last#) (mw.ustring because non-Western
'local' digits)
        state = whitelist.numbered_arguments[enum_name];
        if true == state_test (state, name) then return true; end

        return false;
-- not supported because not found or name is set to nil
end


--[=[-------------------------< I N T E R _ W I K I _ C H E C K >----------------
------------------------------

check <value> for inter-language interwiki-link markup.  <prefix> must be a
MediaWiki-recognized language
code.  when these values have the form (without leading colon):
        [[<prefix>:link|label]] return label as plain-text
        [[<prefix>:link]] return <prefix>:link as plain-text

return value as is else

]=]

local function inter_wiki_check (parameter, value)
        local prefix = value:match ('%[%[(%a+):');
-- get an interwiki prefix if one exists
        local _;

        if prefix and cfg.inter_wiki_map[prefix:lower()] then
-- if prefix is in the map, needs preceding colon so
                utilities.set_message ('err_bad_paramlink', parameter);
-- emit an error message
                _, value, _ = utilities.is_wikilink (value);
-- extract label portion from wikilink
        end
        return value;
end


--[[-------------------------< M I S S I N G _ P I P E _ C H E C K >------------
------------------------------

Look at the contents of a parameter. If the content has a string of characters
and digits followed by an equal
sign, compare the alphanumeric string to the list of cs1|2 parameters.  If found,
then the string is possibly a
parameter that is missing its pipe.  There are two tests made:
```

```
        {{cite ... |title=Title access-date=2016-03-17}}        -- the first
parameter has a value and whitespace separates that value from the missing pipe
parameter name
        {{cite ... |title=access-date=2016-03-17}}                      -- the
first parameter has no value (whitespace after the first = is trimmed by
MediaWiki)
cs1|2 shares some parameter names with XML/HTML attributes: class=, title=, etc.
To prevent false positives XML/HTML
tags are removed before the search.

If a missing pipe is detected, this function adds the missing pipe maintenance
category.

]]

local function missing_pipe_check (parameter, value)
        local capture;
        value = value:gsub ('%b<>', '');
-- remove XML/HTML tags because attributes: class=, title=, etc.

        capture = value:match ('%s+(%a[%w%-]+)%s*=') or value:match ('^(%a[%w%-
]+)%s*=');      -- find and categorize parameters with possible missing pipes
        if capture and validate (capture) then
-- if the capture is a valid parameter name
                utilities.set_message ('err_missing_pipe', parameter);
        end
end


--[[--------------------------< H A S _ E X T R A N E O U S _ P U N C T >--------
------------------------------

look for extraneous terminal punctuation in most parameter values; parameters
listed in skip table are not checked

]]

local function has_extraneous_punc (param, value)
        if 'number' == type (param) then
                return;
        end

        param = param:gsub ('%d+', '#');
-- enumerated name-list mask params allow terminal punct; normalize
        if cfg.punct_skip[param] then
                return;
-- parameter name found in the skip table so done
        end

        if value:match ('[,;:]$') then
                utilities.set_message ('maint_extra_punct');
-- has extraneous punctuation; add maint cat
        end
        if value:match ('^=') then
-- sometimes an extraneous '=' character appears ...
                utilities.set_message ('maint_extra_punct');
-- has extraneous punctuation; add maint cat
        end
end


--[[--------------------------< H A S _ E X T R A N E O U S _ U R L >------------
```

```
------------------------------

look for extraneous url parameter values; parameters listed in skip table are not
checked

]]

local function has_extraneous_url (url_param_t)
	local url_error_t = {};

	check_for_url (url_param_t, url_error_t);
-- extraneous url check
	if 0 ~= #url_error_t then
-- non-zero when there are errors
		table.sort (url_error_t);
		utilities.set_message ('err_param_has_ext_link',
{utilities.make_sep_list (#url_error_t, url_error_t)}); -- add this error message
	end
end


--[[--------------------------< C I T A T I O N >----------------------------------
------------------------------

This is used by templates such as {{cite book}} to create the actual citation
text.

]]

local function citation(frame)
	Frame = frame;
-- save a copy in case we need to display an error message in preview mode
	local sandbox = '/sandbox'
-- i18n: replace this rvalue with the name that your wiki uses to identify
sandbox subpages
	is_sandbox = nil ~= string.find (frame:getTitle(), sandbox, 1, true);
-- is this invoke the sandbox module?
	sandbox = is_sandbox and sandbox or '';
-- use i18n sandbox to load sandbox modules when this module is the sandox; live
modules else

	local pframe = frame:getParent()
	local styles;

	cfg = mw.loadData ('Module:Citation/CS1/Configuration' .. sandbox);
-- load sandbox versions of support modules when
{{#invoke:Citation/CS1/sandbox|...}}; live modules else
	whitelist = mw.loadData ('Module:Citation/CS1/Whitelist' .. sandbox);
	utilities = require ('Module:Citation/CS1/Utilities' .. sandbox);
	validation = require ('Module:Citation/CS1/Date_validation' .. sandbox);
	identifiers = require ('Module:Citation/CS1/Identifiers' .. sandbox);
	metadata = require ('Module:Citation/CS1/COinS' .. sandbox);
	styles = 'Module:Citation/CS1' .. sandbox .. '/styles.css';

	utilities.set_selected_modules (cfg);
-- so that functions in Utilities can see the selected cfg tables
	identifiers.set_selected_modules (cfg, utilities);
-- so that functions in Identifiers can see the selected cfg tables and selected
Utilities module
	validation.set_selected_modules (cfg, utilities);
-- so that functions in Date validataion can see selected cfg tables and the
selected Utilities module
```

```
        metadata.set_selected_modules (cfg, utilities);
-- so that functions in COinS can see the selected cfg tables and selected
Utilities module

        z = utilities.z;
-- table of error and category tables in Module:Citation/CS1/Utilities

        is_preview_mode = not utilities.is_set (frame:preprocess
('{{REVISIONID}}'));

        local args = {};
-- table where we store all of the template's arguments
        local suggestions = {};
-- table where we store suggestions if we need to loadData them
        local error_text;
-- used as a flag

        local config = {};
-- table to store parameters from the module {{#invoke:}}
        for k, v in pairs( frame.args ) do
-- get parameters from the {{#invoke}} frame
                config[k] = v;
        --      args[k] = v;
-- crude debug support that allows us to render a citation from module
{{#invoke:}}; skips parameter validation; TODO: keep?
        end

        local capture;
-- the single supported capture when matching unknown parameters using patterns
        local empty_unknowns = {};
-- sequence table to hold empty unknown params for error message listing
        for k, v in pairs( pframe.args ) do
-- get parameters from the parent (template) frame
                v = mw.ustring.gsub (v, '^%s*(.-)%s*$', '%1');
-- trim leading/trailing whitespace; when v is only whitespace, becomes empty
string
                if v ~= '' then
                        if ('string' == type (k)) then
                                k = mw.ustring.gsub (k, '%d',
cfg.date_names.local_digits);           -- for enumerated parameters, translate
'local' digits to Western 0-9
                        end
                        if not validate( k, config.CitationClass ) then
                                if type (k) ~= 'string' then
-- exclude empty numbered parameters
                                        if v:match("%S+") ~= nil then
                                                error_text =
utilities.set_message ('err_text_ignored', {v});
                                        end
                                elseif validate (k:lower(), config.CitationClass)
then
                                        error_text = utilities.set_message
('err_parameter_ignored_suggest', {k, k:lower()});      -- suggest the lowercase
version of the parameter
                                else
                                        if nil == suggestions.suggestions then
-- if this table is nil then we need to load it
                                                if is_sandbox then
-- did the {{#invoke:}} use sandbox version?
                                                        suggestions =
mw.loadData( 'Module:Citation/CS1/Suggestions/sandbox' );       -- use the
sandbox version
```

```
                                                    else
                                                            suggestions =
mw.loadData( 'Module:Citation/CS1/Suggestions' );                       -- use
the live version
                                                    end
                                            end
                                            for pattern, param in pairs
(suggestions.patterns) do               -- loop through the patterns to see if we
can suggest a proper parameter
                                                    capture = k:match (pattern);
-- the whole match if no capture in pattern else the capture if a match
                                                    if capture then
-- if the pattern matches
                                                            param =
utilities.substitute (param, capture);          -- add the capture to the
suggested parameter (typically the enumerator)
                                                            if validate (param,
config.CitationClass) then              -- validate the suggestion to make sure
that the suggestion is supported by this template (necessary for limited
parameter lists)
                                                                    error_text =
utilities.set_message ('err_parameter_ignored_suggest', {k, param});    -- set
the suggestion error message
                                                            else
                                                                    error_text =
utilities.set_message ('err_parameter_ignored', {k});   -- suggested param not
supported by this template
                                                                    v = '';
-- unset
                                                            end
                                                    end
                                            end
                                            if not utilities.is_set (error_text) then
-- couldn't match with a pattern, is there an explicit suggestion?
                                                    if (suggestions.suggestions[
k:lower() ] ~= nil) and validate (suggestions.suggestions[ k:lower() ],
config.CitationClass) then
                                                            utilities.set_message
('err_parameter_ignored_suggest', {k, suggestions.suggestions[ k:lower() ]});
                                                    else
                                                            utilities.set_message
('err_parameter_ignored', {k});
                                                            v = '';
-- unset value assigned to unrecognized parameters (this for the limited
parameter lists)
                                                    end
                                            end
                                    end
                            end

                            args[k] = v;
-- save this parameter and its value

                    elseif not utilities.is_set (v) then
-- for empty parameters
                            if not validate (k, config.CitationClass, true) then
-- is this empty parameter a valid parameter
                                    k = ('' == k) and '(empty string)' or k;
-- when k is empty string (or was space(s) trimmed to empty string), replace with
descriptive text
                                    table.insert (empty_unknowns,
utilities.wrap_style ('parameter', k)); -- format for error message and add to
```

```
the list
                        end

-- crude debug support that allows us to render a citation from module
{{#invoke:}} TODO: keep?
        --      elseif args[k] ~= nil or (k == 'postscript') then
-- when args[k] has a value from {{#invoke}} frame (we don't normally do that)
        --              args[k] = v;
-- overwrite args[k] with empty string from pframe.args[k] (template frame); v is
empty string here
                end
-- not sure about the postscript bit; that gets handled in parameter validation;
historical artifact?
        end

        if 0 ~= #empty_unknowns then
-- create empty unknown error message
                utilities.set_message ('err_param_unknown_empty', {
                        1 == #empty_unknowns and '' or 's',
                        utilities.make_sep_list (#empty_unknowns, empty_unknowns)
                        });
        end

        local url_param_t = {};

        for k, v in pairs( args ) do
                if 'string' == type (k) then
-- don't evaluate positional parameters
                        has_invisible_chars (k, v);
-- look for invisible characters
                end
                has_extraneous_punc (k, v);
-- look for extraneous terminal punctuation in parameter values
                missing_pipe_check (k, v);
-- do we think that there is a parameter that is missing a pipe?
                args[k] = inter_wiki_check (k, v);
-- when language interwiki-linked parameter missing leading colon replace with
wiki-link label

                if 'string' == type (k) and not cfg.url_skip[k] then
-- when parameter k is not positional and not in url skip table
                        url_param_t[k] = v;
-- make a parameter/value list for extraneous url check
                end
        end

        has_extraneous_url (url_param_t);
-- look for url in parameter values where a url does not belong

        return table.concat ({
                frame:extensionTag ('templatestyles', '', {src=styles}),
                citation0( config, args)
        });
end


--[[--------------------------< E X P O R T E D   F U N C T I O N S >-----------
------------------------------
]]

return {citation = citation};
```