

Toggle menu
Blue Gold Program Wiki

Navigation

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

Tools

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

Personal tools

- [Log in](#)

personal-extra

Toggle search

Search

Random page

Views

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

Actions

Module:Citation/CS1/Identifiers

From Blue Gold Program Wiki

< [Module:Citation/CS1](#)

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

`{{#lst:Module:Citation/CS1/doc|header}}` This page renders and performs error checking on the various named identifiers supported by [Module:Citation/CS1](#).

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

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

```
local is_set, in_array, set_error, select_one, add_maint_cat, substitute,  
make_wikilink;      -- functions in Module:Citation/CS1/Utilities
```

```
local z;  
-- table of tables defined in Module:Citation/CS1/Utilities
```

```
local cfg;  
-- table of configuration tables that are defined in  
Module:Citation/CS1/Configuration
```

```
-----<< H E L P E R   F U N C T I O N S  
>>-----
```

```
--[[-----< W I K I D A T A _ A R T I C L E _ N A M E _ G  
E T >-----
```

as an aid to internationalizing identifier-label wikilinks, gets identifier
article names from wikidata.

returns :<lang code>:<article title> when <q> has an <article title> for
<lang code>; nil else

for identifiers that do not have q, returns nil

for wikis that do not have mw.wikibase installed, returns nil

```
]]
```

```
local function wikidata_article_name_get (q)  
    if not is_set (q) or (q and not mw.wikibase) then  
-- when no q number or when a q number but mw.wikibase not installed on this  
wiki  
        return nil;  
-- abandon  
    end  
  
    local wd_article;  
    local this_wiki_code = cfg.this_wiki_code;  
-- wikipedia subdomain; 'en' for en.wikipedia.org  
  
    wd_article = mw.wikibase.getSitelink (q, this_wiki_code .. 'wiki');
```

```

-- fetch article title from wd; nil when no title available at this wiki

    if wd_article then
        wd_article = table.concat ( {':', this_wiki_code, ':',
wd_article});          -- interwiki-style link without brackets if
taken from wd; leading colon required
    end

    return wd_article;
-- article title from wd; nil else
end

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

Formats a wiki style external link

]]

local function external_link_id(options)
    local url_string = options.id;
    local ext_link;
    local this_wiki_code = cfg.this_wiki_code;
-- wikipedia subdomain; 'en' for en.wikipedia.org
    local wd_article;
-- article title from wikidata
    if options.encode == true or options.encode == nil then
        url_string = mw.uri.encode( url_string );
    end

    ext_link = mw.ustring.format ( '[%s%s%s %s]', options.prefix,
url_string, options.suffix or "", mw.text.nowiki(options.id));
    if is_set(options.access) then
        ext_link = substitute (cfg.presentation['ext-link-access-
signal'], {cfg.presentation[options.access].class,
cfg.presentation[options.access].title, ext_link});          -- add the free-
to-read / paywall lock
    end

    if not (cfg.use_identifier_redirects and is_set (options.redirect))
then          -- redirect has priority so if enabled and available don't fetch
from wikidata because expensive
        wd_article = wikidata_article_name_get (options.q);
-- if wikidata has an article title for this wiki, get it;
    end
    local label_link = (cfg.use_identifier_redirects and is_set
(options.redirect) and options.redirect) or wd_article or options.link;

    return table.concat          ( {
        make_wikilink (label_link, options.label),

```

```

-- redirect, wikidata link, or locally specified link (in that order)
    options.separator or '&nbsp;',
    ext_link
    });
end

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

Formats a wiki style internal link

]]

local function internal_link_id(options)
    local id = mw.ustr.gsub (options.id, '%d',
cfg.date_names.local_digits);    -- translate 'local' digits to Western
0-9

    if not (cfg.use_identifier_redirects and is_set (options.redirect))
then    -- redirect has priority so if enabled and available don't fetch
from wikidata because expensive
        wd_article = wikidata_article_name_get (options.q);
-- if wikidata has an article title for this wiki, get it;
        end

        local label_link = (cfg.use_identifier_redirects and is_set
(options.redirect) and options.redirect) or wd_article or options.link;

        return table.concat (
            {
                make_wikilink (label_link, options.label),
-- wiki link the identifier label
                options.separator or '&nbsp;',
-- add the separator
                make_wikilink (
                    table.concat (
                        {
                            options.prefix,
                            id,
-- translated to western digits
                            options.suffix or ''
                        },
                        substitute (cfg.presentation['bdi'], {'',
mw.text.nowiki (options.id)})    -- bdi tags to prevent Latn script
identifiers from being reversed at rtl language wikis
                    );
-- nowiki because MediaWiki still has magic links for ISBN and the like;
TODO: is it really required?
                });
end

```

```
--[[-----< I S _ E M B A R G O E D >-----  
-----
```

Determines if a PMC identifier's online version is embargoed. Compares the date in |embargo= against today's date. If embargo date is in the future, returns the content of |embargo=; otherwise, returns an empty string because the embargo has expired or because |embargo= was not set in this cite.

```
]]
```

```
local function is_embargoed (embargo)  
    if is_set (embargo) then  
        local lang = mw.getContentLanguage();  
        local good1, embargo_date, good2, todays_date;  
        good1, embargo_date = pcall( lang.formatDate, lang, 'U',  
embargo );  
        good2, todays_date = pcall( lang.formatDate, lang, 'U' );  
        if good1 and good2 then  
-- if embargo date and today's date are good dates  
            if tonumber( embargo_date ) >= tonumber( todays_date  
) then  
                -- is embargo date is in the future?  
                return embargo;  
-- still embargoed  
            else  
                add_maint_cat ('embargo')  
                return '';  
-- unset because embargo has expired  
            end  
        end  
    end  
    return '';  
-- |embargo= not set return empty string  
end
```

```
--[=[-----< I S _ V A L I D _ B I O R X I V _ D A T E >---  
-----
```

returns true if:

```
2019-12-11T00:00Z <= biorxiv_date < today + 2 days
```

The dated form of biorxiv identifier has a start date of 2019-12-11. The unix timestamp for that date is `{{#time:U|2019-12-11}}` = 1576022400

biorxiv_date is the date provided in those |biorxiv= parameter values that are dated at time 00:00:00 UTC

today is the current date at time 00:00:00 UTC plus 48 hours

```
    if today is 2015-01-01T00:00:00 then
```

adding 24 hours gives 2015-01-02T00:00:00 – one second more than today

```
        adding 24 hours gives 2015-01-03T00:00:00 – one second more
```

than tomorrow

This function does not work if it is fed month names for languages other than English. Wikimedia #time: parser apparently doesn't understand non-English date month names. This function will always return false when the date contains a non-English month name because good1 is false after the call to lang.formatDate(). To get around that call this function with YYYY-MM-DD format dates.

]=]

```
local function is_valid_biorxiv_date (biorxiv_date)
    local good1, good2;
    local biorxiv_ts, tomorrow_ts;
-- to hold unix time stamps representing the dates
    local lang_object = mw.getContentLanguage();

    good1, biorxiv_ts = pcall (lang_object.formatDate, lang_object, 'U',
biorxiv_date );          -- convert biorxiv_date value to unix
timesatmp
    good2, tomorrow_ts = pcall (lang_object.formatDate, lang_object, 'U',
'today + 2 days' );      -- today midnight + 2 days is one second more than
all day tomorrow
    if good1 and good2 then
-- lang.formatDate() returns a timestamp in the local script which which
tonumber() may not understand
        biorxiv_ts = tonumber (biorxiv_ts) or
lang_object:parseFormattedNumber (biorxiv_ts);      -- convert to numbers
for the comparison;
        tomorrow_ts = tonumber (tomorrow_ts) or
lang_object:parseFormattedNumber (tomorrow_ts);
    else
        return false;
-- one or both failed to convert to unix time stamp
    end

    return ((1576022400 <= biorxiv_ts) and (biorxiv_ts < tomorrow_ts))
-- 2012-12-11T00:00Z <= biorxiv_date < tomorrow's date
end
```

--[[-----< IS _ V A L I D _ I S X N >-----

ISBN-10 and ISSN validator code calculates checksum across all isbn/issn digits including the check digit.
ISBN-13 is checked in isbn().

If the number is valid the result will be 0. Before calling this function, issbn/issn must be checked for length

and stripped of dashes, spaces and other non-isxn characters.

```
]]

local function is_valid_isxn (isxn_str, len)
    local temp = 0;
    isxn_str = { isxn_str:byte(1, len) };
-- make a table of byte values '0' → 0x30 .. '9' → 0x39, 'X' → 0x58
    len = len+1;
-- adjust to be a loop counter
    for i, v in ipairs( isxn_str ) do
-- loop through all of the bytes and calculate the checksum
        if v == string.byte( "X" ) then
-- if checkdigit is X (compares the byte value of 'X' which is 0x58)
            temp = temp + 10*( len - i );
-- it represents 10 decimal
        else
            temp = temp + tonumber( string.char(v) )*(len-i);
        end
    end
    return temp % 11 == 0;
-- returns true if calculation result is zero
end

--[[-----< IS _ V A L I D _ I S X N _ 1 3 >-----
-----

ISBN-13 and ISMN validator code calculates checksum across all 13 isbn/ismn
digits including the check digit.
If the number is valid, the result will be 0. Before calling this function,
isbn-13/ismn must be checked for length
and stripped of dashes, spaces and other non-isxn-13 characters.

]]

local function is_valid_isxn_13 (isxn_str)
    local temp=0;
    isxn_str = { isxn_str:byte(1, 13) };
-- make a table of byte values '0' → 0x30 .. '9' → 0x39
    for i, v in ipairs( isxn_str ) do
        temp = temp + (3 - 2*(i % 2)) * tonumber( string.char(v) );
-- multiply odd index digits by 1, even index digits by 3 and sum; includes
check digit
    end
    return temp % 10 == 0;
-- sum modulo 10 is zero when isbn-13/ismn is correct
end

--[[-----< N O R M A L I Z E _ L C C N >-----
```

lccn normalization

(<http://www.loc.gov/marc/lccn-namespace.html#normalization>)

1. Remove all blanks.
2. If there is a forward slash (/) in the string, remove it, and remove all characters to the right of the forward slash.
3. If there is a hyphen in the string:
 - a. Remove it.
 - b. Inspect the substring following (to the right of) the (removed) hyphen. Then (and assuming that steps 1 and 2 have been carried out):
 1. All these characters should be digits, and there should be six or less. (not done in this function)
 2. If the length of the substring is less than 6, left-fill the substring with zeroes until the length is six.

Returns a normalized lccn for lccn() to validate. There is no error checking (step 3.b.1) performed in this function.

]]

local function normalize_lccn (lccn)

 lccn = lccn:gsub ("%s", "");

-- 1. strip whitespace

 if nil ~= string.find (lccn, '/') then

 lccn = lccn:match ("(-)/");

-- 2. remove forward slash and all character to the right of it
 end

 local prefix

 local suffix

 prefix, suffix = lccn:match ("(.+)%-(.+)");

-- 3.a remove hyphen by splitting the string into prefix and suffix

 if nil ~= suffix then

-- if there was a hyphen

 suffix=string.rep("0", 6-string.len (suffix)) .. suffix;

-- 3.b.2 left fill the suffix with 0s if suffix length less than 6

 lccn=prefix..suffix;

-- reassemble the lccn

 end

 return lccn;

 end

=====<< I D E N T I F I E R F U N C T I O N S
>>=====

--[[-----< A R X I V >-----

See: http://arxiv.org/help/arxiv_identifier

format and error check arXiv identifier. There are three valid forms of the identifier:

the first form, valid only between date codes 9108 and 0703 is:

```
arXiv:<archive>.<class>/<date code><number><version>
```

where:

<archive> is a string of alpha characters - may be hyphenated; no other punctuation

<class> is a string of alpha characters - may be hyphenated; no other punctuation; not the same as |class= parameter which is not supported in this form

<date code> is four digits in the form YYYY where YY is the last two digits of the four-digit year and MM is the month number January = 01

first digit of YY for this form can only 9 and 0

<number> is a three-digit number

<version> is a 1 or more digit number preceded with a lowercase v; no spaces (undocumented)

the second form, valid from April 2007 through December 2014 is:

```
arXiv:<date code>.<number><version>
```

where:

<date code> is four digits in the form YYYY where YY is the last two digits of the four-digit year and MM is the month number January = 01

<number> is a four-digit number

<version> is a 1 or more digit number preceded with a lowercase v; no spaces

the third form, valid from January 2015 is:

```
arXiv:<date code>.<number><version>
```

where:

<date code> and <version> are as defined for 0704-1412

<number> is a five-digit number

]]

```
local function arxiv (id, class)
```

```
    local handler = cfg.id_handlers['ARXIV'];
```

```
    local year, month, version;
```

```
    local err_cat = false;
```

```
-- assume no error message
```

```
    local text;
```

```
-- output text
```

```
    if id:match("^%a[%a%.%-]+/[90]%d[01]%d%d%d%d$") or
id:match("^%a[%a%.%-]+/[90]%d[01]%d%d%d%dv%d+$") then          -- test for the
9108-0703 format w/ & w/o version
```

```
        year, month = id:match("^%a[%a%.%-]+/([90]%d)([01]%d)%d%d%d[v%d]*$");
```

```
        year = tonumber(year);
```

```
        month = tonumber(month);
```

```
        if ((not (90 < year or 8 > year)) or (1 > month or 12 <
month)) or          -- if invalid year or invalid month
```

```
            ((91 == year and 7 > month) or (7 == year and 3 <
month)) then          -- if years ok, are starting and ending months
```

```
ok?
```

```

err_cat = true;
-- flag for error message
end

elseif id:match("^%d%d[01]%d%.%d%d%d%d$d$") or
id:match("^%d%d[01]%d%.%d%d%d%dv%d+$") then -- test for the 0704-1412
w/ & w/o version
year, month = id:match("^(%d%d)([01]%d)%.%d%d%d%d[v%d]*$");
year = tonumber(year);
month = tonumber(month);
if ((7 > year) or (14 < year) or (1 > month or 12 < month))
or -- is year invalid or is month invalid? (doesn't
test for future years)
((7 == year) and (4 > month)) then --or
-- when year is 07, is month invalid (before April)?
err_cat = true;
-- flag for error message
end

elseif id:match("^%d%d[01]%d%.%d%d%d%d%d$d$") or
id:match("^%d%d[01]%d%.%d%d%d%d%dv%d+$") then -- test for the 1501-
format w/ & w/o version
year, month = id:match("^(%d%d)([01]%d)%.%d%d%d%d%d[v%d]*$");
year = tonumber(year);
month = tonumber(month);
if ((15 > year) or (1 > month or 12 < month)) then
-- is year invalid or is month invalid? (doesn't test for future years)
err_cat = true;
-- flag for error message
end

else
err_cat = true;
-- not a recognized format; flag for error message
end

err_cat = err_cat and table.concat ({' ', set_error ('bad_arxiv')})
or ''; -- set error message if flag is true
text = external_link_id ({link=handler.link, label=handler.label,
q=handler.q, redirect=handler.redirect,
prefix=handler.prefix,id=id,separator=handler.separator,
encode=handler.encode, access=handler.access}) .. err_cat;

if is_set (class) then
if id:match ('^%d+') then
text = table.concat ({text, '
[[//arxiv.org/archive/', class, ' ', class, ']]'}); -- external link
within square brackets, not wikilink
else
text = table.concat ({text, ' ', set_error
('class_ignored')});

```

```

        end
    end
    return text;
end

```

```

--[[-----< B I B C O D E >-----
-----

```

Validates (sort of) and formats a bibcode id.

Format for bibcodes is specified here:

http://adsabs.harvard.edu/abs_doc/help_pages/data.html#bibcodes

But, this: 2015arXiv151206696F is apparently valid so apparently, the only things that really matter are length, 19 characters and first four digits must be a year. This function makes these tests:

```

    length must be 19 characters
    characters in position
        1-4 must be digits and must represent a year in the range of
1000 - next year
        5 must be a letter
        6-8 must be letter, digit, ampersand, or dot (ampersand
cannot directly precede a dot; &. )
        9-18 must be letter, digit, or dot
        19 must be a letter or dot

```

```

]]

```

```

local function bibcode (id, access)

```

```

    local handler = cfg.id_handlers['BIBCODE'];
    local err_type;
    local year;

```

```

    local text = external_link_id ({link=handler.link,
label=handler.label, q=handler.q, redirect=handler.redirect,
        prefix=handler.prefix, id=id, separator=handler.separator,
encode=handler.encode,
        access=access});

```

```

    if 19 ~= id:len() then
        err_type = cfg.err_msg_supl.length;
    else

```

```

        year = id:match
        ("^(%d%d%d%d)[%a][%w&%%.][%w&%%.][%w&%%.][%w.]+[%a%.]$") --

```

```

        if not year then
-- if nil then no pattern match
            err_type = cfg.err_msg_supl.value;

```

```

-- so value error

```

```

        else
            local next_year = tonumber(os.date ('%Y'))+1;

```

```

-- get the current year as a number and add one for next year
    year = tonumber (year);
-- convert year portion of bibcode to a number
    if (1000 > year) or (year > next_year) then
        err_type = cfg.err_msg_supl.year;
-- year out of bounds
    end
    if id:find('&%.') then
        err_type = cfg.err_msg_supl.journal;
-- journal abbreviation must not have '&.' (if it does its missing a letter)
    end
    end
end

    if is_set (err_type) then
-- if there was an error detected
        text = text .. ' ' .. set_error( 'bad_bibcode', {err_type});
    end
    return text;
end

```

```

--[[-----< B I O R X I V >-----
-----

```

Format bioRxiv id and do simple error checking. Before 2019-12-11, biorXiv ids were 10.1101/ followed by exactly 6 digits. After 2019-12-11, biorXiv ids retained the six-digit identifier but prefixed that with a yyyy.mm.dd. date and suffixed with an optional version identifier.

The bioRxiv id is the string of characters:

<https://doi.org/10.1101/078733> -> 10.1101/078733

or a date followed by a six-digit number followed by an optional version indicator 'v' and one or more digits:

<https://www.biorxiv.org/content/10.1101/2019.12.11.123456v2> -> 10.1101/2019.12.11.123456v2

see <https://www.biorxiv.org/about-biorxiv>

```

]]

```

```

local function biorxiv(id)

```

```

    local handler = cfg.id_handlers['BIORXIV'];

```

```

    local err_cat = true;

```

```

-- flag; assume that there will be an error

```

```

    local patterns = {

```

```

        '^10.1101/%d%d%d%d%d%d$',

```

```

-- simple 6-digit identifier (before 2019-12-11)

```

```

'^10.1101/(20[1-9]%d)%.[01]%d).[0-3]%d).%d%d%d%d%d%dv%d+$',

```

```

-- y.m.d. date + 6-digit identifier + version (after 2019-12-11)

```

```

'^10.1101/(20[1-9]%d)%.[01]%d).[0-3]%d).%d%d%d%d%d$',

```

```

-- y.m.d. date + 6-digit identifier (after 2019-12-11)
    }
    for _, pattern in ipairs (patterns) do
-- spin through the patterns looking for a match
        if id:match (pattern) then
            local y, m, d = id:match (pattern);
-- found a match, attempt to get year, month and date from the identifier

                if m then
-- m is nil when id is the six-digit form
                    if not is_valid_biorxiv_date (y .. '-' .. m
.. '-' .. d) then -- validate the encoded date; TODO: don't ignore
leapyear and actual month lengths ({{#time:}} is a poor date validator)
                        break;
-- date fail; break out early so we don't unset the error message
                    end
                end
                err_cat = nil;
-- we found a match so unset the error message
                break;
-- and done
            end
        end
    end
-- err_cat remains set here when no match

    return external_link_id ({link=handler.link, label=handler.label,
q=handler.q, redirect=handler.redirect,
prefix=handler.prefix,id=id,separator=handler.separator,
                        encode=handler.encode, access=handler.access}) ..
(err_cat and (' ' .. set_error( 'bad_biorxiv')) or '');
end

```

```

--[[-----< C I T E S E E R X >-----
-----

```

CiteSeerX use their own notion of "doi" (not to be confused with the identifiers resolved via doi.org).

The description of the structure of this identifier can be found at [Help_talk:Citation_Style_1/Archive_26#CiteSeerX_id_structure](https://en.wikipedia.org/wiki/Help_talk:Citation_Style_1/Archive_26#CiteSeerX_id_structure)]]

```

local function citeseerx (id)
    local handler = cfg.id_handlers['CITeseerX'];
    local matched;
    local text = external_link_id ({link=handler.link,
label=handler.label, q=handler.q, redirect=handler.redirect,
                        prefix=handler.prefix, id=id, separator=handler.separator,
encode=handler.encode,
                        access=handler.access});

```

```

        matched = id:match ("^10%.1%.1%.[1-9]%d?%d?%d?%.[1-9]%d?%d?%d?$");
        if not matched then
            text = text .. ' ' .. set_error( 'bad_citeseerx' );
        end
        return text;
    end
end

```

```

--[[-----< D O I >-----
-----

```

Formats a DOI and checks for DOI errors.

DOI names contain two parts: prefix and suffix separated by a forward slash.

Prefix: directory indicator '10.' followed by a registrant code

Suffix: character string of any length chosen by the registrant

This function checks a DOI name for: prefix/suffix. If the doi name contains spaces or endashes, or, if it ends

with a period or a comma, this function will emit a bad_doi error message.

DOI names are case-insensitive and can incorporate any printable Unicode characters so the test for spaces, endash, and terminal punctuation may not be technically correct but it appears, that in practice these characters are rarely if ever used in doi names.

```

]]

```

```

local function doi(id, inactive, access)
    local cat = ""
    local handler = cfg.id_handlers['DOI'];
    local text;
    if is_set(inactive) then
        local inactive_year = inactive:match("%d%d%d%d") or '';
-- try to get the year portion from the inactive date
        local inactive_month, good;

        if is_set (inactive_year) then
            if 4 < inactive:len() then
-- inactive date has more than just a year (could be anything)
                local lang_obj = mw.getContentLanguage();
-- get a language object for this wiki
                good, inactive_month = pcall
(lang_obj.formatDate, lang_obj, 'F', inactive); -- try to get the
month name from the inactive date
                if not good then
                    inactive_month = nil;
-- something went wrong so make sure this is unset
                end
            end
        end
    end
end

```

```

else
    inactive_year = nil;
-- |doi-broken= has something but it isn't a date
end
    if is_set(inactive_year) and is_set (inactive_month) then
        table.insert( z.error_categories, 'Pages with DOIs
inactive as of ' .. inactive_year .. ' ' .. inactive_month);      -- use
inactive month in category name
    elseif is_set(inactive_year) then
        table.insert( z.error_categories, 'Pages with DOIs
inactive as of ' .. inactive_year);
    else
        table.insert( z.error_categories, 'Pages with
inactive DOIs');          -- when inactive doesn't contain a
recognizable date
    end
    inactive = " (" .. cfg.messages['inactive'] .. ' ' ..
inactive .. ')';
end

    text = external_link_id ({link=handler.link, label=handler.label,
q=handler.q, redirect=handler.redirect,
        prefix=handler.prefix,id=id,separator=handler.separator,
encode=handler.encode, access=access}) .. (inactive or '')

    local registrant = id:match ('^10%.([/+)]/[^%s-]-[^%.,$]');
-- registrant set when doi has the proper basic form
    registrant_err_patterns = {
-- these patterns are for code ranges that are not supported
        '^1-3]d%d%d%d%.%d*d*$',
-- 5 digits with subcode (0xxxx, 40000+); accepts: 10000-39999
        '^1-4]d%d%d%d$d$',
-- 5 digits without subcode (0xxxx, 40000+); accepts: 10000-49999
        '^1-9]d%d%d%d%.%d*d*$',
-- 4 digits with subcode (0xxx); accepts: 1000-9999
        '^1-9]d%d%d$d$',
-- 4 digits without subcode (0xxx); accepts: 1000-9999
        '^d%d%d%d%d+d+',
-- 6 or more digits
        '^d%d?%d?$',
-- less than 4 digits without subcode (with subcode is legitimate)
        '^5555$',
-- test registrant will never resolve
        '%s',
-- any space character in registrant
    }
    if registrant then
-- when doi has proper form
        for i, pattern in ipairs (registrant_err_patterns) do
-- spin through error patterns
            if registrant:match (pattern) then

```

```

-- to validate registrant codes
                                cat = ' ' .. set_error ('bad_doi');
-- when found, mark this doi as bad
                                break;
-- and done
                                end
                                end
                                else
                                cat = ' ' .. set_error ('bad_doi');
-- invalid directory or malformed
                                end

                                return text .. cat
end

```

```

--[[-----< H D L >-----
-----

```

Formats an HDL with minor error checking.

HDL names contain two parts: prefix and suffix separated by a forward slash.

Prefix: character string using any character in the UCS-2 character set except '/'

Suffix: character string of any length using any character in the UCS-2 character set chosen by the registrant

This function checks a HDL name for: prefix/suffix. If the HDL name contains spaces, endashes, or, if it ends with a period or a comma, this function will emit a bad_hdl error message.

HDL names are case-insensitive and can incorporate any printable Unicode characters so the test for endashes and terminal punctuation may not be technically correct but it appears, that in practice these characters are rarely if ever used in HDLs.

Query string parameters are named here:

http://www.handle.net/proxy_servlet.html. query strings are not displayed but since '?' is an allowed character in an hdl, '?' followed by one of the query parameters is the only way we have to detect the query string so that it isn't url encoded with the rest of the identifier.

```
]]
```

```

local function hdl(id, access)
    local handler = cfg.id_handlers['HDL'];
    local query_params = {
-- list of known query parameters from
http://www.handle.net/proxy\_servlet.html

```



```

        'noredirect',
        'ignore_aliases',
        'auth',
        'cert',
        'index',
        'type',
        'urlappend',
        'locatt',
        'action',
    }
    local hdl, suffix, param = id:match ('(.-)(%?(%a+).+)$');
-- look for query string
    local found;

    if hdl then
-- when there are query strings, this is the handle identifier portion
        for _, q in ipairs (query_params) do
-- spin through the list of query parameters
            if param:match ('^' .. q) then
-- if the query string begins with one of the parameters
                found = true;
-- announce a find
                break;
-- and stop looking
            end
        end
    end

    if found then
        id = hdl;
-- found so replace id with the handle portion; this will be url encoded,
suffix will not
    else
        suffix = '';
-- make sure suffix is empty string for concatenation else
    end

    local text = external_link_id ({link=handler.link,
label=handler.label, q=handler.q, redirect=handler.redirect,
        prefix=handler.prefix, id=id, suffix=suffix,
separator=handler.separator, encode=handler.encode, access=access})

    if nil == id:match ("^[^%s-]-/[^%s-]-[^\%.,]$" ) then
-- hdl must contain a fwd slash, must not contain spaces, endashes, and must
not end with period or comma
        text = text .. ' ' .. set_error( 'bad_hdl' );
    end
    return text;
end
end

```

```
--[[-----< I S B N >-----  
-----
```

Determines whether an ISBN string is valid

```
]]  
  
local function isbn( isbn_str )  
    if nil ~= isbn_str:match("[^%s-0-9X]") then  
        return false, cfg.err_msg_supl.char;  
    -- fail if isbn_str contains anything but digits, hyphens, or the uppercase X  
    end  
    isbn_str = isbn_str:gsub( "-", "" ):gsub( " ", "" );  
    -- remove hyphens and spaces  
    local len = isbn_str:len();  
  
    if len ~= 10 and len ~= 13 then  
        return false, cfg.err_msg_supl.length;  
    -- fail if incorrect length  
    end  
  
    if len == 10 then  
        if isbn_str:match( "^%d*X?$" ) == nil then  
            -- fail if isbn_str has 'X' anywhere but last position  
            return false, cfg.err_msg_supl.form;  
        end  
        return is_valid_isxn(isbn_str, 10), cfg.err_msg_supl.check;  
    else  
        if isbn_str:match( "^%d+$" ) == nil then  
            -- fail if isbn13 is not all digits  
            end  
            if isbn_str:match( "^97[89]%d*$" ) == nil then  
                -- fail when isbn13 does not begin with 978 or 979  
                end  
                if isbn_str:match ( '^9790' ) then  
                    -- group identifier '0' is reserved to ismn  
                    end  
                    return is_valid_isxn_13 ( isbn_str ), cfg.err_msg_supl.check;  
                end  
            end  
        end  
    end  
end
```

```
--[[-----< A M A Z O N >-----  
-----
```

Formats a link to Amazon. Do simple error checking: asin must be mix of 10 numeric or uppercase alpha characters. If a mix, first character must be uppercase alpha; if all

numeric, asins must be 10-digit isbn. If 10-digit isbn, add a maintenance category so a bot or awb script can replace |asin= with |isbn=. Error message if not 10 characters, if not isbn10, if mixed and first character is a digit.

This function is positioned here because it calls isbn()

```
]]

local function asin(id, domain)
    local err_cat = ""

    if not
id:match("^[%d%u][%d%u][%d%u][%d%u][%d%u][%d%u][%d%u][%d%u][%d%u][%d%u]$")
then
        err_cat = ' ' .. set_error ('bad_asin');
-- asin is not a mix of 10 uppercase alpha and numeric characters
    else
        if id:match("^%d%d%d%d%d%d%d%d%d[dX]$") then
-- if 10-digit numeric (or 9 digits with terminal X)
            if isbn( id ) then
-- see if asin value is isbn10
                add_maint_cat ('ASIN');
                elseif not is_set (err_cat) then
                    err_cat = ' ' .. set_error ('bad_asin');
-- asin is not isbn10
            end
            elseif not id:match("^%u[%d%u]+$") then
                err_cat = ' ' .. set_error ('bad_asin');
-- asin doesn't begin with uppercase alpha
            end
        end
        if not is_set(domain) then
            domain = "com";
        elseif in_array (domain, {'jp', 'uk'}) then
Japan, United Kingdom
            domain = "co." .. domain;
        elseif in_array (domain, {'au', 'br', 'mx'}) then
Australia, Brazil, Mexico
            domain = "com." .. domain;
        end
        local handler = cfg.id_handlers['ASIN'];
        return external_link_id ({link=handler.link, label=handler.label,
q=handler.q, redirect=handler.redirect,
            prefix=handler.prefix .. domain .. "/dp/",
            id=id, encode=handler.encode, separator = handler.separator})
        .. err_cat;
    end
end
```

```

--[[-----< I S M N >-----
-----

Determines whether an ISMN string is valid. Similar to isbn-13, ismn is 13
digits begining 979-0-... and uses the
same check digit calculations. See
http://www.ismn-international.org/download/Web_ISMN_Users_Manual_2008-6.pdf
section 2, pages 9–12.

]]

local function ismn (id)
    local handler = cfg.id_handlers['ISMN'];
    local text;
    local valid_ismn = true;
    local id_copy;

    id_copy = id;
-- save a copy because this testing is destructive
    id=id:gsub( "[%s--]", "" );
-- strip spaces, hyphens, and endashes from the ismn

    if 13 ~= id:len() or id:match( "^9790%d*$" ) == nil then
-- ismn must be 13 digits and begin 9790
        valid_ismn = false;
    else
        valid_ismn=is_valid_isxn_13 (id);
-- validate ismn
    end

--    text = internal_link_id ({link=handler.link, label=handler.label,
q=handler.q, redirect=handler.redirect, -- use this (or
external version) when there is some place to link to
--    prefix=handler.prefix, id=id_copy,
separator=handler.separator, encode=handler.encode})

    local label_link = (cfg.use_identifier_redirects and is_set
(handler.redirect) and handler.redirect) or wd_article or handler.link;
-- because no place to link to yet

    text = table.concat (
-- because no place to link to yet
    {
        make_wikilink (label_link, handler.label),
        handler.separator,
        id_copy
    });

    if false == valid_ismn then
        text = text .. ' ' .. set_error( 'bad_ismn' )
-- add an error message if the ismn is invalid

```

```
        end
        return text;
end
```

```
--[[-----< I S S N >-----
-----
```

Validate and format an issn. This code fixes the case where an editor has included an ISSN in the citation but has separated the two groups of four digits with a space. When that condition occurred, the resulting link looked like this:

```
    |issn=0819 4327 gives: [http://www.worldcat.org/issn/0819 4327 0819
4327]    -- can't have spaces in an external link
```

This code now prevents that by inserting a hyphen at the issn midpoint. It also validates the issn for length and makes sure that the checkdigit agrees with the calculated value. Incorrect length (8 digits), characters other than 0-9 and X, or checkdigit / calculated value mismatch will all cause a check issn error message. The issn is always displayed with a hyphen, even if the issn was given as a single group of 8 digits.

```
]]
```

```
local function issn(id, e)
    local issn_copy = id;
    -- save a copy of unadulterated issn; use this version for display if issn
does not validate
    local handler;
    local text;
    local valid_issn = true;
    if e then
        handler = cfg.id_handlers['EISSN'];
    else
        handler = cfg.id_handlers['ISSN'];
    end

    id=id:gsub( "[%s--]", "" );
    -- strip spaces, hyphens, and endashes from the issn

    if 8 ~= id:len() or nil == id:match( "^%d*X?$" ) then
    -- validate the issn: 8 digits long, containing only 0-9 or X in the last
position
        valid_issn=false;
    -- wrong length or improper character
    else
        valid_issn=is_valid_issn(id, 8);
    -- validate issn
```

```

end

    if true == valid_issn then
        id = string.sub( id, 1, 4 ) .. "-" .. string.sub( id, 5 );
-- if valid, display correctly formatted version
    else
        id = issn_copy;
-- if not valid, use the show the invalid issn with error message
    end
    text = external_link_id ({link=handler.link, label=handler.label,
q=handler.q, redirect=handler.redirect,
        prefix=handler.prefix,id=id,separator=handler.separator,
encode=handler.encode})

    if false == valid_issn then
        text = text .. ' ' .. set_error( 'bad_issn', e and 'e' or ''
)
        -- add an error message if the issn is invalid
    end
    return text
end
end

```

```

--[[-----< J F M >-----
-----

```

A numerical identifier in the form nn.nnnn.nn

```

]]

```

```

local function jfm (id)
    local handler = cfg.id_handlers['JFM'];
    local id_num;
    local err_cat = '';
    id_num = id:match ('^[Jj][Ff][Mm](.*)$');
-- identifier with jfm prefix; extract identifier

    if is_set (id_num) then
        add_maint_cat ('jfm_format');
    else
-- plain number without mr prefix
        id_num = id;
-- if here id does not have prefix
    end

    if id_num and id_num:match('^%d%d%.%d%d%d%d%.%d%d$') then
        id = id_num;
-- jfm matches pattern
    else
        err_cat = ' ' .. set_error( 'bad_jfm' );
-- set an error message
    end
end

```

```

        return external_link_id ({link=handler.link, label=handler.label,
q=handler.q, redirect=handler.redirect,
prefix=handler.prefix,id=id,separator=handler.separator,
encode=handler.encode}) .. err_cat;
end

```

```

--[[-----< L C C N >-----
-----

```

Format LCCN link and do simple error checking. LCCN is a character string 8-12 characters long. The length of the LCCN dictates the character type of the first 1-3 characters; the rightmost eight are always digits.
<http://info-uri.info/registry/OAIHandler?verb=GetRecord&metadataPrefix=reg&identifier=info:lccn/>

```

length = 8 then all digits
length = 9 then lccn[1] is lower case alpha
length = 10 then lccn[1] and lccn[2] are both lower case alpha or both digits
length = 11 then lccn[1] is lower case alpha, lccn[2] and lccn[3] are both
lower case alpha or both digits
length = 12 then lccn[1] and lccn[2] are both lower case alpha

```

```

]]

```

```

local function lccn(lccn)
    local handler = cfg.id_handlers['LCCN'];
    local err_cat = '';
-- presume that LCCN is valid
    local id = lccn;
-- local copy of the lccn

    id = normalize_lccn (id);
-- get canonical form (no whitespace, hyphens, forward slashes)
    local len = id:len();
-- get the length of the lccn

    if 8 == len then
        if id:match("[^%d]") then
-- if LCCN has anything but digits (nil if only digits)
            err_cat = ' ' .. set_error( 'bad_lccn' );
-- set an error message
        end
    elseif 9 == len then
-- LCCN should be adddddddd
        if nil == id:match("%l%d%d%d%d%d%d%d") then
-- does it match our pattern?
            err_cat = ' ' .. set_error( 'bad_lccn' );
-- set an error message
        end
    end

```

```

elseif 10 == len then
-- LCCN should be aaaaaaaaa or ddddddddd
    if id:match("[^%d]") then
-- if LCCN has anything but digits (nil if only digits) ...
        if nil == id:match("^%l%l%d%d%d%d%d%d%d") then
-- ... see if it matches our pattern
            err_cat = ' ' .. set_error( 'bad_lccn' );
-- no match, set an error message
        end
    end
elseif 11 == len then
-- LCCN should be aaaaaaaaaa or aaaaaaaaaa
    if not (id:match("^%l%l%l%d%d%d%d%d%d%d") or
id:match("^%l%d%d%d%d%d%d%d%d%d")) then -- see if it matches one of
our patterns
        err_cat = ' ' .. set_error( 'bad_lccn' );
-- no match, set an error message
    end
elseif 12 == len then
-- LCCN should be aaaaaaaaaa
    if not id:match("^%l%l%d%d%d%d%d%d%d%d") then
-- see if it matches our pattern
        err_cat = ' ' .. set_error( 'bad_lccn' );
-- no match, set an error message
    end
else
    err_cat = ' ' .. set_error( 'bad_lccn' );
-- wrong length, set an error message
end

if not is_set (err_cat) and nil ~= lccn:find ('%s') then
    err_cat = ' ' .. set_error( 'bad_lccn' );
-- lccn contains a space, set an error message
end

return external_link_id ({link=handler.link, label=handler.label,
q=handler.q, redirect=handler.redirect,
prefix=handler.prefix,id=lccn,separator=handler.separator,
encode=handler.encode}) .. err_cat;
end

```

--[[-----< M R >-----

A seven digit number; if not seven digits, zero-fill leading digits to make seven digits.

]]

local function mr (id)


```

        local handler = cfg.id_handlers['MR'];
        local id_num;
        local id_len;
        local err_cat = '';
        id_num = id:match ('^[Mm][Rr](%d+)$');
-- identifier with mr prefix

        if is_set (id_num) then
            add_maint_cat ('mr_format');
        else
-- plain number without mr prefix
            id_num = id:match ('^%d+$');
-- if here id is all digits
            end

            id_len = id_num and id_num:len() or 0;
            if (7 >= id_len) and (0 ~= id_len) then
                id = string.rep ('0', 7-id_len ) .. id_num;
-- zero-fill leading digits
            else
                err_cat = ' ' .. set_error( 'bad_mr' );
-- set an error message
            end
            return external_link_id ({link=handler.link, label=handler.label,
q=handler.q, redirect=handler.redirect,
prefix=handler.prefix,id=id,separator=handler.separator,
encode=handler.encode}) .. err_cat;
end

--[[-----< O C L C >-----
-----

```

Validate and format an oclc id.

<https://www.oclc.org/batchload/controlnumber.en.html> {{dead link}}

archived at:

<https://web.archive.org/web/20161228233804/https://www.oclc.org/batchload/controlnumber.en.html>

]]

local function oclc (id)

```

        local handler = cfg.id_handlers['OCLC'];
        local number;
        local err_msg = '';
-- empty string for concatenation
        if id:match ('^ocm%d%d%d%d%d%d%d$d$') then
-- ocm prefix and 8 digits; 001 field (12 characters)
            number = id:match ('ocm(%d+)');
-- get the number
        elseif id:match ('^ocn%d%d%d%d%d%d%d%d$d$') then

```

```

-- ocn prefix and 9 digits; 001 field (12 characters)
    number = id:match('ocn(%d+)');
-- get the number
    elseif id:match('^on%d%d%d%d%d%d%d%d%d+d+') then
-- on prefix and 10 or more digits; 001 field (12 characters)
    number = id:match('^on(%d%d%d%d%d%d%d%d%d+d+)$');
-- get the number
    elseif id:match('^%(0CoLC%)[1-9]d*$') then
-- (0CoLC) prefix and variable number digits; no leading zeros; 035 field
    number = id:match('%(0CoLC%)([1-9]d*)');
-- get the number
    if 9 < number:len() then
        number = nil;
-- constrain to 1 to 9 digits; change this when oclc issues 10-digit numbers
    end
    elseif id:match('^%d+$') then
-- no prefix
    number = id;
-- get the number
    if 10 < number:len() then
        number = nil;
-- constrain to 1 to 10 digits; change this when oclc issues 11-digit numbers
    end
end

    if number then
-- proper format
        id = number;
-- exclude prefix, if any, from external link
    else
        err_msg = ' ' .. set_error( 'bad_oclc' )
-- add an error message if the id is malformed
    end
    local text = external_link_id ({link=handler.link,
label=handler.label, q=handler.q, redirect=handler.redirect,
prefix=handler.prefix, id=id, separator=handler.separator,
encode=handler.encode}) .. err_msg;

    return text;
end

```

```

--[[-----< O P E N L I B R A R Y >-----
-----

```

Formats an OpenLibrary link, and checks for associated errors.

```
]]
```

```

local function openlibrary(id, access)
    local code;

```

```

    local handler = cfg.id_handlers['OL'];
    local ident;
    ident, code = id:gsub('^OL', ''):match("^(%d+([AMW]))$");
-- optional OL prefix followed immediately by digits followed by 'A', 'M', or
'W'; remove OL prefix

    if not is_set (ident) then
-- if malformed return an error
        return external_link_id ({link=handler.link,
label=handler.label, q=handler.q, redirect=handler.redirect,
        prefix=handler.prefix .. 'OL',
        id=id, separator=handler.separator,          encode =
handler.encode,
        access = access}) .. ' ' .. set_error( 'bad_ol' );
    end
    id = ident;
-- use ident without the optional OL prefix (it has been removed)
    if ( code == "A" ) then
        return external_link_id ({link=handler.link,
label=handler.label, q=handler.q, redirect=handler.redirect,
        prefix=handler.prefix .. 'authors/OL',
        id=id, separator=handler.separator,          encode =
handler.encode,
        access = access})
    end
    if ( code == "M" ) then
        return external_link_id ({link=handler.link,
label=handler.label, q=handler.q, redirect=handler.redirect,
        prefix=handler.prefix .. 'books/OL',
        id=id, separator=handler.separator,          encode =
handler.encode,
        access = access})
    end
    if ( code == "W" ) then
        return external_link_id ({link=handler.link,
label=handler.label, q=handler.q, redirect=handler.redirect,
        prefix=handler.prefix .. 'works/OL',
        id=id, separator=handler.separator,          encode =
handler.encode,
        access = access})
    end
end
end

```

```

--[[-----< P M C >-----
-----

```

Format a PMC, do simple error checking, and check for embargoed articles.

The embargo parameter takes a date for a value. If the embargo date is in the

future the PMC identifier will not be linked to the article. If the embargo date is today or in the past, or if it is empty or omitted, then the PMC identifier is linked to the article through the link at `cfg.id_handlers['PMC'].prefix`.

PMC embargo date testing is done in function `is_embargoed ()` which is called earlier because when the citation has `|pmc=<value>` but does not have a `|url=` then `|title=` is linked with the PMC link. Function `is_embargoed ()` returns the embargo date if the PMC article is still embargoed, otherwise it returns an empty string.

PMCs are sequential numbers beginning at 1 and counting up. This code checks the PMC to see that it contains only digits and is less than `test_limit`; the value in local variable `test_limit` will need to be updated periodically as more PMCs are issued.

```
]]
```

```
local function pmc(id, embargo)
    local handler = cfg.id_handlers['PMC'];
    local err_cat = '';
-- presume that PMC is valid
    local id_num;
    local text;
    id_num = id:match ('^[Pp][Mm][Cc](%d+)$');
-- identifier with pmc prefix

    if is_set (id_num) then
        add_maint_cat ('pmc_format');
    else
-- plain number without pmc prefix
        id_num = id:match ('^%d+$');
-- if here id is all digits
        end

        if is_set (id_num) then
-- id_num has a value so test it
            id_num = tonumber(id_num);
-- convert id_num to a number for range testing
            if 1 > id_num or handler.id_limit < id_num then
-- if PMC is outside test limit boundaries
                err_cat = ' ' .. set_error( 'bad_pmc' );
-- set an error message
            else
                id = tostring (id_num);
-- make sure id is a string
            end
        else
-- when id format incorrect
```

```

                err_cat = ' ' .. set_error( 'bad_pmc' );
-- set an error message
            end
            if is_set (embargo) then
-- is PMC is still embargoed?
                text = table.concat (
-- still embargoed so no external link
                    {
                        make_wikilink (handler.link, handler.label),
                        handler.separator,
                        id,
                        err_cat
                    });
            else
                text = external_link_id ({link=handler.link,
label=handler.label, q=handler.q, redirect=handler.redirect,      -- no
embargo date or embargo has expired, ok to link to article
prefix=handler.prefix,id=id,separator=handler.separator,
encode=handler.encode, access=handler.access}) .. err_cat;
            end
            return text;
        end
    end
end

```

```

--[[-----< P M I D >-----
-----

```

Format PMID and do simple error checking. PMIDs are sequential numbers beginning at 1 and counting up. This code checks the PMID to see that it contains only digits and is less than test_limit; the value in local variable test_limit will need to be updated periodically as more PMIDs are issued.

```

]]

```

```

local function pmid(id)
    local handler = cfg.id_handlers['PMID'];
    local err_cat = '';
-- presume that PMID is valid
    if id:match("[^%d]") then
-- if PMID has anything but digits
        err_cat = ' ' .. set_error( 'bad_pmid' );
-- set an error message
    else
-- PMID is only digits
        local id_num = tonumber(id);
-- convert id to a number for range testing
        if 1 > id_num or handler.id_limit < id_num then
-- if PMID is outside test limit boundaries
            err_cat = ' ' .. set_error( 'bad_pmid' );
-- set an error message

```

```

        end
        end
        return external_link_id ({link=handler.link, label=handler.label,
q=handler.q, redirect=handler.redirect,
prefix=handler.prefix,id=id,separator=handler.separator,
encode=handler.encode}) .. err_cat;
end

--[[-----< S 2 C I D >-----
-----

Format an s2cid, do simple error checking

S2CIDs are sequential numbers beginning at 1 and counting up. This code
checks the s2cid to see that it is only
digits and is less than test_limit; the value in local variable test_limit
will need to be updated periodically
as more S2CIDs are issued.

]]

local function s2cid (id, access)
    local handler = cfg.id_handlers['S2CID'];
    local err_cat = '';
-- presume that S2CID is valid
    local id_num;
    local text;
    id_num = id:match ('^[1-9]%d*$');
-- id must be all digits; must not begin with 0; no open access flag

    if is_set (id_num) then
-- id_num has a value so test it
        id_num = tonumber(id_num);
-- convert id_num to a number for range testing
        if handler.id_limit < id_num then
-- if S2CID is outside test limit boundaries
            err_cat = ' ' .. set_error( 'bad_s2cid' );
-- set an error message
            end

        else
-- when id format incorrect
            err_cat = ' ' .. set_error( 'bad_s2cid' );
-- set an error message
            end

        text = external_link_id ({link=handler.link, label=handler.label,
q=handler.q, redirect=handler.redirect,
        prefix=handler.prefix, id=id:gsub ('%.%a%', ''),
separator=handler.separator, encode=handler.encode, access=access}) ..

```

```
err_cat;
```

```
    return text;
```

```
end
```

```
--[[-----< S B N >-----  
-----
```

9-digit form of isbn10; uses same check-digit validation when sbn is prefixed with an additional '0' to make 10 digits

```
]]
```

```
local function sbn (id)
```

```
    local check;
```

```
    local err_type = '';
```

```
    if nil ~= id:match("[^%s-0-9X]") then
```

```
        return false, cfg.err_msg_supl.char;
```

```
-- fail if sbn contains anything but digits, hyphens, or the uppercase X  
end
```

```
    id=id:gsub( "[%s-]", "" );
```

```
-- strip spaces and hyphens from the sbn
```

```
    if 9 ~= id:len() then
```

```
        return false, cfg.err_msg_supl.length;
```

```
-- fail if incorrect length  
end
```

```
    if id:match( "^%d*X?$" ) == nil then
```

```
-- fail if sbn has 'X' anywhere but last position
```

```
        return false, cfg.err_msg_supl.form;
```

```
    end
```

```
    return is_valid_isxn('0' .. id, 10), cfg.err_msg_supl.check;
```

```
-- prefix sbn with '0' and validate as isbn10
```

```
end
```

```
--[[-----< S S R N >-----  
-----
```

Format an ssrn, do simple error checking

SSRNs are sequential numbers beginning at 100? and counting up. This code checks the ssrn to see that it is only digits and is greater than 99 and less than test_limit; the value in local variable test_limit will need to be updated periodically as more SSRNs are issued.

```

]]

local function ssrn (id)
    local handler = cfg.id_handlers['SSRN'];
    local err_cat = '';
-- presume that SSRN is valid
    local id_num;
    local text;
    id_num = id:match ('^%d+$');
-- id must be all digits

    if is_set (id_num) then
-- id_num has a value so test it
        id_num = tonumber(id_num);
-- convert id_num to a number for range testing
        if 100 > id_num or handler.id_limit < id_num then
-- if SSRN is outside test limit boundaries
            err_cat = ' ' .. set_error( 'bad_ssrn' );
-- set an error message
            end
        else
-- when id format incorrect
            err_cat = ' ' .. set_error( 'bad_ssrn' );
-- set an error message
            end
        text = external_link_id ({link=handler.link, label=handler.label,
q=handler.q, redirect=handler.redirect,
            prefix=handler.prefix,id=id,separator=handler.separator,
encode=handler.encode, access=handler.access}) .. err_cat;

    return text;
end

--[[-----< U S E N E T _ I D >-----
-----

Validate and format a usenet message id.  Simple error checking, looks for
'id-left@id-right' not enclosed in
'<' and/or '>' angle brackets.

]]

local function usenet_id (id)
    local handler = cfg.id_handlers['USENETID'];

    local text = external_link_id ({link=handler.link,
label=handler.label, q=handler.q, redirect=handler.redirect,
        prefix=handler.prefix,id=id,separator=handler.separator,
encode=handler.encode})

```



```

        if not id:match('^.+@.+$') or not id:match('^[^<].*[^>]$') then
-- doesn't have '@' or has one or first or last character is '<' or '>'
            text = text .. ' ' .. set_error( 'bad_usenet_id' )
-- add an error message if the message id is invalid
        end
        return text
end
end

```

```

--[[-----< Z B L >-----
-----

```

A numerical identifier in the form nnnn.nnnnn - leading zeros in the first quartet optional

format described here: <http://emis.mi.sanu.ac.rs/ZMATH/zmath/en/help/search/>
temporary format is apparently eight digits. Anything else is an error

```
]]
```

```

local function zbl (id)
    local handler = cfg.id_handlers['ZBL'];
    local err_cat = '';
    if id:match('^%d%d%d%d%d%d%d%d$') then
-- is this identifier using temporary format?
        add_maint_cat ('zbl');
-- yes, add maint cat
    elseif not id:match('^%d?%d?%d?%d?%.%d%d%d%d$d$') then
-- not temporary, is it normal format?
        err_cat = ' ' .. set_error( 'bad_zbl' );
-- no, set an error message
    end
    return external_link_id ({link=handler.link, label=handler.label,
q=handler.q, redirect=handler.redirect,
prefix=handler.prefix,id=id,separator=handler.separator,
encode=handler.encode}) .. err_cat;
end

```

```

===== << I N T E R F A C E   F U N C T I O N S
>>=====

```

```

--[[-----< B U I L D _ I D _ L I S T >-----
-----

```

Takes a table of IDs created by extract_ids() and turns it into a table of formatted ID outputs.

inputs:
id_list – table of identifiers built by extract_ids()

options – table of various template parameter values used to modify some manually handled identifiers

```
]]

local function build_id_list( id_list, options )
    local new_list, handler = {};

    local function fallback(k) return { __index = function(t,i) return
cfg.id_handlers[k][i] end } end;
    for k, v in pairs( id_list ) do
-- k is uc identifier name as index to cfg.id_handlers; e.g.
cfg.id_handlers['ISBN'], v is a table
        -- fallback to read-only cfg
        handler = setmetatable( { ['id'] = v, ['access'] =
options.IdAccessLevels[k] }, fallback(k) );

        if handler.mode == 'external' then
            table.insert( new_list, {handler.label,
external_link_id( handler ) } );
        elseif handler.mode == 'internal' then
            table.insert( new_list, {handler.label,
internal_link_id( handler ) } );
        elseif handler.mode ~= 'manual' then
            error( cfg.messages['unknown_ID_mode'] );
        elseif k == 'ARXIV' then
            table.insert( new_list, {handler.label, arxiv( v,
options.Class ) } );
        elseif k == 'ASIN' then
            table.insert( new_list, {handler.label, asin( v,
options.ASINTLD ) } );
        elseif k == 'BIBCODE' then
            table.insert( new_list, {handler.label, bibcode( v,
handler.access ) } );
        elseif k == 'BIORXIV' then
            table.insert( new_list, {handler.label, biorxiv( v )
} );
        elseif k == 'CITeseerX' then
            table.insert( new_list, {handler.label, citeseerx( v
) } );
        elseif k == 'DOI' then
            table.insert( new_list, {handler.label, doi( v,
options.DoiBroken, handler.access ) } );
        elseif k == 'EISSN' then
            table.insert( new_list, {handler.label, issn( v, true
) } );
            -- true distinguishes eissn from issn
        elseif k == 'HDL' then
            table.insert( new_list, {handler.label, hdl( v,
handler.access ) } );
        elseif k == 'ISBN' then
            local ISBN = internal_link_id( handler );
```

```

        local check;
        local err_type = '';
        check, err_type = isbn( v );
        if not check then
            if is_set(options.IgnoreISBN) then
-- ISBN is invalid; if |ignore-isbn-error= set
                add_maint_cat ('ignore_isbn_err');
-- ad a maint category
            else
                ISBN = ISBN .. set_error( 'bad_isbn',
{err_type}, false, " ", "" );          -- else display an error message
            end
            elseif is_set(options.IgnoreISBN) then
-- ISBN is OK; if |ignore-isbn-error= set
                add_maint_cat ('ignore_isbn_err');
-- because |ignore-isbn-error= unnecessary
            end
            table.insert( new_list, {handler.label, ISBN } );
        elseif k == 'ISMN' then
            table.insert( new_list, {handler.label, ismn( v ) }
);
        elseif k == 'ISSN' then
            table.insert( new_list, {handler.label, issn( v ) }
);
        elseif k == 'JFM' then
            table.insert( new_list, {handler.label, jfm( v ) } );
        elseif k == 'LCCN' then
            table.insert( new_list, {handler.label, lccn( v ) }
);
        elseif k == 'MR' then
            table.insert( new_list, {handler.label, mr( v ) } );
        elseif k == 'OCLC' then
            table.insert( new_list, {handler.label, oclc( v ) }
);
        elseif k == 'OL' or k == 'OLA' then
            table.insert( new_list, {handler.label, openlibrary(
v, handler.access ) } );
        elseif k == 'PMC' then
            table.insert( new_list, {handler.label, pmc( v,
options.Embargo ) } );
        elseif k == 'PMID' then
            table.insert( new_list, {handler.label, pmid( v ) }
);
        elseif k == 'S2CID' then
            table.insert( new_list, {handler.label, s2cid( v,
handler.access ) } );
        elseif k == 'SBN' then
            local SBN = internal_link_id (handler);
            local check;
-- boolean validation result
            local err_type = '';

```

```

        check, err_type = sbn (v);
        if not check then
            SBN = SBN .. set_error( 'bad_sbn',
{err_type}, false, " ", "" );          -- display an error message
        end
        table.insert( new_list, {handler.label, SBN } );
    elseif k == 'SSRN' then
        table.insert( new_list, {handler.label, ssrn( v ) }
);
    elseif k == 'USENETID' then
        table.insert( new_list, {handler.label, usenet_id( v
) } );
    elseif k == 'ZBL' then
        table.insert( new_list, {handler.label, zbl( v ) } );
    else
        error( cfg.messages['unknown_manual_ID'] );
    end
end
local function comp( a, b )          -- used in following table.sort()
    return a[1]:lower() < b[1]:lower();
end
table.sort( new_list, comp );
for k, v in ipairs( new_list ) do
    new_list[k] = v[2];
end
return new_list;
end

```

```

--[[-----< E X T R A C T _ I D S >-----
-----

```

Populates ID table from arguments using configuration settings. Loops through `cfg.id_handlers` and searches args for any of the parameters listed in each `cfg.id_handlers['...'].parameters`. If found, adds the parameter and value to the identifier list. Emits redundant error message if more than one alias exists in args

```

]]

```

```

local function extract_ids( args )
    local id_list = {};
    -- list of identifiers found in args
    for k, v in pairs( cfg.id_handlers ) do
    -- k is uc identifier name as index to cfg.id_handlers; e.g.
    cfg.id_handlers['ISBN'], v is a table
        v = select_one( args, v.parameters, 'redundant_parameters' );
    -- v.parameters is a table of aliases for k; here we pick one from args if
    present
        if is_set(v) then id_list[k] = v; end
    end
end

```

```

-- if found in args, add identifier to our list
    end
    return id_list;
end

```

```

--[[-----< E X T R A C T _ I D _ A C C E S S _ L E V E L
S >-----

```

Fetches custom id access levels from arguments using configuration settings. Parameters which have a predefined access level (e.g. arxiv) do not use this function as they are directly rendered as free without using an additional parameter.

access-level values must match the case used in `cfg.keywords_lists['id-access']` (lowercase unless there is some special reason for something else)

```

]]

```

```

local function extract_id_access_levels( args, id_list )
    local id_accesses_list = {};
    for k, v in pairs( cfg.id_handlers ) do
        local access_param = v.custom_access;
-- name of identifier's access-level parameter
        if is_set(access_param) then
            local access_level = args[access_param];
-- get the assigned value if there is one
            if is_set (access_level) then
                if not in_array (access_level,
cfg.keywords_lists['id-access']) then -- exact match required
                    table.insert( z.message_tail, {
set_error( 'invalid_param_val', {access_param, access_level}, true ) } );
                    access_level = nil;
-- invalid so unset
                end
                if not is_set(id_list[k]) then
-- identifier access-level must have a matching identifier
                    table.insert( z.message_tail, {
set_error( 'param_access_requires_param', {k:lower()}, true ) } ); --
param name is uppercase in cfg.id_handlers (k); lowercase for error message
                end
                id_accesses_list[k] =
cfg.keywords_xlate[access_level]; -- get translated
keyword
            end
        end
    end
    return id_accesses_list;
end

```

```

--[[-----< S E T _ S E L E C T E D _ M O D U L E S >-----
-----
Sets local cfg table and imported functions table to same (live or sandbox)
as that used by the other modules.

]]

local function set_selected_modules (cfg_table_ptr, utilities_page_ptr)
    cfg = cfg_table_ptr;

    is_set = utilities_page_ptr.is_set;
-- import functions from select Module:Citation/CS1/Utilities module
    in_array = utilities_page_ptr.in_array;
    set_error = utilities_page_ptr.set_error;
    select_one = utilities_page_ptr.select_one;
    add_maint_cat = utilities_page_ptr.add_maint_cat;
    substitute = utilities_page_ptr.substitute;
    make_wikilink = utilities_page_ptr.make_wikilink;

    z = utilities_page_ptr.z;
-- table of tables in Module:Citation/CS1/Utilities
end

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

return {
    build_id_list = build_id_list,
    extract_ids = extract_ids,
    extract_id_access_levels = extract_id_access_levels,
    is_embargoed = is_embargoed;
    set_selected_modules = set_selected_modules;
}

```

Project Management Committee

one hundredth of an acre (0.004 ha)

accreted sediment in a river course or estuary, including both lateral (point-bars) and medial (braid-bars). Chars (or sand bars) emerge as islands within the river channel (island chars) or as attached land to the riverbanks (attached chars), create new opportunities for temporary settlements and agriculture.

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

intended function.

Retrieved from

"<https://www.bluegoldwiki.com/index.php?title=Module:Citation/CS1/Identifiers&oldid=3524>"

Namespaces

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

Variants

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

Blue Gold Program Wiki

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

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

Developed and maintained by Big Blue Communications for Blue Gold Program



[Blue Gold Program Wiki](#)