

☐ 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:Unicode data

From Blue Gold Program Wiki

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

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

```

local p = {}

local floor = math.floor

local function errorf(level, ...)
    if type(level) == "number" then
        return error(string.format(...), level + 1)
    else -- level is actually the format string.
        return error(string.format(level, ...), 2)
    end
end

local function binary_range_search(codepoint, ranges)
    local low, mid, high
    low, high = 1, ranges.length or require
"Module:TableTools".length(ranges)
    while low <= high do
        mid = floor((low + high) / 2)
        local range = ranges[mid]
        if codepoint < range[1] then
            high = mid - 1
        elseif codepoint <= range[2] then
            return range, mid
        else
            low = mid + 1
        end
    end
    return nil, mid
end

p.binary_range_search = binary_range_search

--[[
local function linear_range_search(codepoint, ranges)
    for i, range in ipairs(ranges) do
        if range[1] <= codepoint and codepoint <= range[2] then
            return range
        end
    end
end

end
--]]

-- Load a module by indexing "loader" with the name of the module minus the
-- "Module:Unicode data/" part. For instance, loader.blocks returns
-- [[Module:Unicode data/blocks]]. If a module cannot be loaded, false will
be
-- returned.
local loader = setmetatable({}, {
    __index = function (self, key)
        local success, data = pcall(mw.loadData, "Module:Unicode
data/" .. key)
        if not success then

```

```

        data = false
    end
    self[key] = data
    return data
end
}))

-- For the algorithm used to generate Hangul Syllable names,
-- see "Hangul Syllable Name Generation" in section 3.12 of the
-- Unicode Specification:
-- https://www.unicode.org/versions/Unicode11.0.0/ch03.pdf
local name_hooks = {
    { 0x00, 0x1F, "<control-%04X>" }, -- C0 control characters
    { 0x7F, 0x9F, "<control-%04X>" }, -- DEL and C1 control
characters
    { 0x3400, 0x4DB5, "CJK UNIFIED IDEOGRAPH-%04X" }, -- CJK
Ideograph Extension A
    { 0x4E00, 0x9FEF, "CJK UNIFIED IDEOGRAPH-%04X" }, -- CJK
Ideograph
    { 0xAC00, 0xD7A3, function (codepoint) -- Hangul Syllables
        local Hangul_data = loader.Hangul
        local syllable_index = codepoint - 0xAC00

        return ("HANGUL SYLLABLE %s%s%s"):format(
            Hangul_data.leads[floor(syllable_index /
Hangul_data.final_count)],
            Hangul_data.vowels[floor((syllable_index %
Hangul_data.final_count)
                / Hangul_data.trail_count)],
            Hangul_data.trails[syllable_index %
Hangul_data.trail_count]
        )
    end },
    -- High Surrogates, High Private Use Surrogates, Low Surrogates
    { 0xD800, 0xDFFF, "<surrogate-%04X>" },
    { 0xE000, 0xF8FF, "<private-use-%04X>" }, -- Private Use
    -- CJK Compatibility Ideographs
    { 0xF900, 0xFA6D, "CJK COMPATIBILITY IDEOGRAPH-%04X" },
    { 0xFA70, 0xFAD9, "CJK COMPATIBILITY IDEOGRAPH-%04X" },
    { 0x17000, 0x187F1, "TANGUT IDEOGRAPH-%04X" }, -- Tangut
    { 0x18800, 0x18AF2, function (codepoint)
        return ("TANGUT COMPONENT-%03d"):format(codepoint - 0x187FF)
    end },
    { 0x1B170, 0x1B2FB, "NUSHU CHARACTER-%04X" }, -- Nushu
    { 0x20000, 0x2A6D6, "CJK UNIFIED IDEOGRAPH-%04X" }, -- CJK
Ideograph Extension B
    { 0x2A700, 0x2B734, "CJK UNIFIED IDEOGRAPH-%04X" }, -- CJK
Ideograph Extension C
    { 0x2A740, 0x2B81D, "CJK UNIFIED IDEOGRAPH-%04X" }, -- CJK
Ideograph Extension D
    { 0x2B820, 0x2CEA1, "CJK UNIFIED IDEOGRAPH-%04X" }, -- CJK

```

```

Ideograph Extension E
    { 0x2CEB0, 0x2EBE0, "CJK UNIFIED IDEOGRAPH-%04X" }, -- CJK
Ideograph Extension F
    -- CJK Compatibility Ideographs Supplement (Supplementary Ideographic
Plane)
    { 0x2F800, 0x2FA1D, "CJK COMPATIBILITY IDEOGRAPH-%04X" },
    { 0xE0100, 0xE01EF, function (codepoint) -- Variation Selectors
Supplement
    return ("VARIATION SELECTOR-%d"):format(codepoint - 0xE0100 +
17)
    end},
    { 0xF0000, 0xFFFFD, "<private-use-%04X>" }, -- Plane 15 Private Use
    { 0x100000, 0x10FFFFD, "<private-use-%04X>" } -- Plane 16 Private Use
}
name_hooks.length = #name_hooks

local name_range_cache

local function generate_name(data, codepoint)
    if type(data) == "string" then
        return data:format(codepoint)
    else
        return data(codepoint)
    end
end

end

--[[
-- Checks that the code point is a number and in range.
-- Does not check whether code point is an integer.
-- Not used
local function check_codepoint(funcName, argIdx, val)
    require 'libraryUtil'.checkType(funcName, argIdx, val, 'number')
    if codepoint < 0 or 0x10FFFF < codepoint then
        errorf("Codepoint %04X out of range", codepoint)
    end
end

end
--]]

-- https://www.unicode.org/versions/Unicode11.0.0/ch04.pdf, section 4.8
function p.lookup_name(codepoint)
    -- U+FDD0-U+FDEF and all code points ending in FFFE or FFFF are
Unassigned
    -- (Cn) and specifically noncharacters:
    -- https://www.unicode.org/faq/private_use.html#nonchar4
    if 0xFDD0 <= codepoint and (codepoint <= 0xFDEF
        or floor(codepoint % 0x10000) >= 0xFFFFE) then
        return ("<noncharacter-%04X>"):format(codepoint)
    end

    if name_range_cache -- Check if previously used "name hook" applies
to this code point.

```

```

        and codepoint >= name_range_cache[1]
        and codepoint <= name_range_cache[2] then
            return generate_name(name_range_cache[3], codepoint)
    end
    local range = binary_range_search(codepoint, name_hooks)
    if range then
        name_range_cache = range
        return generate_name(range[3], codepoint)
    end

    local data = loader[('names/%03X'):format(codepoint / 0x1000)]
    if data and data[codepoint] then
        return data[codepoint]
    -- Unassigned (Cn) consists of noncharacters and reserved characters.
    -- The character has been established not to be a noncharacter,
    -- and if it were assigned, its name would already been retrieved,
    -- so it must be reserved.
    else
        return ("<reserved-%04X>"):format(codepoint)
    end
end

--[[
-- No image data modules on Wikipedia yet.
function p.lookup_image(codepoint)
    local data = loader[('images/%03X'):format(codepoint / 0x1000)]
    if data then
        return data[codepoint]
    end
end
--]]

local planes = {
    [ 0] = "Basic Multilingual Plane";
    [ 1] = "Supplementary Multilingual Plane";
    [ 2] = "Supplementary Ideographic Plane";
    [14] = "Supplementary Special-purpose Plane";
    [15] = "Supplementary Private Use Area-A";
    [16] = "Supplementary Private Use Area-B";
}

-- Load [[Module:Unicode data/blocks]] if needed and assign it to this
variable.
local blocks

local function block_iter(blocks, i)
    i = i + 1
    local data = blocks[i]
    if data then
        -- Unpack doesn't work on tables loaded with mw.loadData.
        return i, data[1], data[2], data[3]
    end
end

```

```

        end
    end

    -- An ipairs-type iterator generator for the list of blocks.
    function p.enum_blocks()
        local blocks = loader.blocks
        return block_iter, blocks, 0
    end

    function p.lookup_plane(codepoint)
        local i = floor(codepoint / 0x10000)
        return planes[i] or ("Plane %u"):format(i)
    end

    function p.lookup_block(codepoint)
        local blocks = loader.blocks
        local range = binary_range_search(codepoint, blocks)
        if range then
            return range[3]
        else
            return "No Block"
        end
    end

    function p.get_block_info(name)
        for i, block in ipairs(loader.blocks) do
            if block[3] == name then
                return block
            end
        end
    end

    function p.is_valid_pagename(pagename)
        local has_nonws = false

        for cp in mw.ustr.string.gcodepoint(pagename) do
            if (cp == 0x0023) -- #
            or (cp == 0x005B) -- [
            or (cp == 0x005D) -- ]
            or (cp == 0x007B) -- {
            or (cp == 0x007C) -- |
            or (cp == 0x007D) -- }
            or (cp == 0x180E) -- MONGOLIAN VOWEL SEPARATOR
            or ((cp >= 0x2000) and (cp <= 0x200A)) -- spaces in General
Punctuation block
            or (cp == 0xFFFD) -- REPLACEMENT CHARACTER
            then
                return false
            end

            local printable, result = p.is_printable(cp)

```

```

        if not printable then
            return false
        end

        if result ~= "space-separator" then
            has_nonws = true
        end
    end

    return has_nonws
end

local function manual_unpack(what, from)
    if what[from + 1] == nil then
        return what[from]
    end
    local result = {}
    from = from or 1
    for i, item in ipairs(what) do
        if i >= from then
            table.insert(result, item)
        end
    end
    return unpack(result)
end

local function compare_ranges(range1, range2)
    return range1[1] < range2[1]
end

-- Creates a function to look up data in a module that contains "singles" (a
-- code point-to-data map) and "ranges" (an array containing arrays that
-- contain
-- the low and high code points of a range and the data associated with that
-- range).
-- "loader" loads and returns the "singles" and "ranges" tables.
-- "match_func" is passed the code point and either the data or the "dots",
-- and
-- generates the final result of the function.
-- The varargs ("dots") describes the default data to be returned if there
-- wasn't
-- a match.
-- In case the function is used more than once, "cache" saves ranges that
-- have
-- already been found to match, or a range whose data is the default if there
-- was no match.
local function memo_lookup(data_module_subpage, match_func, ...)
    local dots = { ... }
    local cache = {}
    local singles, ranges

```

```

    return function (codepoint)
        if not singles then
            local data_module = loader[data_module_subpage]
            singles, ranges = data_module.singles,
data_module.ranges
        end

        if singles[codepoint] then
            return match_func(codepoint, singles[codepoint])
        end

        local range = binary_range_search(codepoint, cache)
        if range then
            return match_func(codepoint, manual_unpack(range, 3))
        end
        local range, index = binary_range_search(codepoint, ranges)
        if range then
            table.insert(cache, range)
            table.sort(cache, compare_ranges)
            return match_func(codepoint, manual_unpack(range, 3))
        end
        if ranges[index] then
            local dots_range
            if codepoint > ranges[index][2] then
                dots_range = {
                    ranges[index][2] + 1,
                    ranges[index + 1] and ranges[index +
1][1] - 1 or 0x10FFFF,
                    unpack(dots)
                }
            else -- codepoint < range[index][1]
                dots_range = {
                    ranges[index - 1] and ranges[index -
1][2] + 1 or 0,
                    ranges[index][1] - 1,
                    unpack(dots)
                }
            end
            table.sort(cache, compare_ranges)
        end
        return match_func(codepoint)
    end
end

```

```

-- Get a code point's combining class value in [[Module:Unicode
data/combining]],
-- and return whether this value is not zero. Zero is assigned as the default
-- if the combining class value is not found in this data module.
-- That is, return true if character is combining, or false if it is not.
-- See https://www.unicode.org/reports/tr44/#Canonical\_Combining\_Class\_Values
for

```



```

-- more information.
p.is_combining = memo_lookup(
    "combining",
    function (codepoint, combining_class)
        return combining_class and combining_class ~= 0 or false
    end,
    0)

function p.add_dotted_circle(str)
    return (mw.usttring.gsub(str, ".",
        function(char)
            if p.is_combining(mw.usttring.codepoint(char)) then
                return '◌' .. char
            end
        end))
end

local lookup_control = memo_lookup(
    "control",
    function (codepoint, ccc)
        return ccc or "assigned"
    end,
    "assigned")
p.lookup_control = lookup_control

function p.is_assigned(codepoint)
    return lookup_control(codepoint) ~= "unassigned"
end

function p.is_printable(codepoint)
    local result = lookup_control(codepoint)
    return (result == "assigned") or (result == "space-separator"),
result
end

function p.is_whitespace(codepoint)
    local result = lookup_control(codepoint)
    return (result == "space-separator"), result
end

p.lookup_category = memo_lookup(
    "category",
    function (codepoint, category)
        return category
    end,
    "Cn")

local lookup_script = memo_lookup(
    "scripts",
    function (codepoint, script_code)
        return script_code or 'Zzzz'
    end,
    "Zzzz")

```

```

        end,
        "Zzzz")
p.lookup_script = lookup_script

function p.get_best_script(str)
    -- Check type of argument, because mw.text.decode coerces numbers to
strings!
    require "libraryUtil".checkType("get_best_script", 1, str, "string")
    -- Convert HTML character references (including named character
references,
    -- or character entities) to characters.
    str = mw.text.decode(str, true)
    local scripts = {}
    for codepoint in mw.ustring.gcodepoint(str) do
        local script = lookup_script(codepoint)
        -- Ignore "Inherited", "Undetermined", or "Uncoded" scripts.
        if not (script == "Zyyy" or script == "Zinh" or script ==
"Zzzz") then
            scripts[script] = true
        end
    end
    -- If scripts does not contain two or more keys,
    -- return first and only key (script code) in table.
    if not next(scripts, next(scripts)) then
        return next(scripts)
    end -- else return majority script, or else "Zzzz"?
end

function p.is_Latin(str)
    require "libraryUtil".checkType("get_best_script", 1, str, "string")
    str = mw.text.decode(str, true)
    -- Search for the leading bytes that introduce the UTF-8 encoding of
the
    -- code points U+0340-U+10FFFF. If they are not found and there is at
least
    -- one Latin-script character, the string counts as Latin, because
the rest
    -- of the characters can only be Zyyy, Zinh, and Zzzz.
    -- The only scripts found below U+0370 (the first code point of the
Greek
    -- and Coptic block) are Latn, Zyyy, Zinh, and Zzzz.
    -- See the codepage in the [[UTF-8]] article.
    if not str:find "[\205-\244]" then
        for codepoint in mw.ustring.gcodepoint(str) do
            if lookup_script(codepoint) == "Latn" then
                return true
            end
        end
    end
    local Latn = false
    for codepoint in mw.ustring.gcodepoint(str) do

```

```

        local script = lookup_script(codepoint)
        if script == "Latn" then
            Latn = true
        elseif not (script == "Zyyy" or script == "Zinh"
                    or script == "Zzzz") then
            return false
        end
    end
    return Latn
end

-- Checks that a string contains only characters belonging to right-to-left
-- scripts, or characters of ignorable scripts.
function p.is_rtl(str)
    require "libraryUtil".checkType("get_best_script", 1, str, "string")
    str = mw.text.decode(str, true)
    -- Search for the leading bytes that introduce the UTF-8 encoding of
the
    -- code points U+0580-U+10FFFF. If they are not found, the string can
only
    -- have characters from a left-to-right script, because the first
code point
    -- in a right-to-left script is U+0591, in the Hebrew block.
    if not str:find "[\\214-\\244]" then
        return false
    end
    local result = false
    local rtl = loader.scripts.rtl
    for codepoint in mw.ustring.gcodepoint(str) do
        local script = lookup_script(codepoint)
        if rtl[script] then
            result = true
        elseif not (script == "Zyyy" or script == "Zinh"
                    or script == "Zzzz") then
            return false
        end
    end
    return result
end

local function get_codepoint(args, arg)
    local codepoint_string = args[arg]
        or errorf(2, "Parameter %s is required", tostring(arg))
    local codepoint = tonumber(codepoint_string, 16)
        or errorf(2, "Parameter %s is not a code point in hexadecimal
base",
        tostring(arg))
    if not (0 <= codepoint and codepoint <= 0x10FFFF) then
        errorf(2, "code point in parameter %s out of range",
tostring(arg))
    end
end

```

```

        return codepoint
end

local function get_func(args, arg, prefix)
    local suffix = args[arg]
        or errorf(2, "Parameter %s is required", tostring(arg))
    suffix = mw.text.trim(suffix)
    local func_name = prefix .. suffix
    local func = p[func_name]
        or errorf(2, "There is no function '%s'", func_name)
    return func
end

-- This function allows any of the "lookup" functions to be invoked. The
first
-- parameter is the word after "lookup_"; the second parameter is the code
point
-- in hexadecimal base.
function p.lookup(frame)
    local func = get_func(frame.args, 1, "lookup_")
    local codepoint = get_codepoint(frame.args, 2)
    local result = func(codepoint)
    if func == p.lookup_name then
        -- Prevent code point labels such as <control-0000> from
being
        -- interpreted as HTML tags.
        result = result:gsub("<", "&lt;")
    end
    return result
end

function p.is(frame)
    local func = get_func(frame.args, 1, "is_")
    -- is_Latin and is_valid_pagename take strings.
    if func == p.is_Latin or func == p.is_valid_pagename or func ==
p.is_rtl then
        return (func(frame.args[2]))
    else -- The rest take code points.
        local codepoint = get_codepoint(frame.args, 2)
        return (func(codepoint)) -- Adjust to one result.
    end
end

return p

```

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.

Retrieved from "https://www.bluegoldwiki.com/index.php?title=Module:Unicode_data&oldid=1816"

Namespaces

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

Variants

This page was last edited on 19 February 2020, at 09:51.

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](#)