

```
--[[
```

Lua source code for the Lua/APR binding.

Author: Peter Odding <peter@peterodding.com>

Last Change: January 8, 2011

Homepage: <http://peterodding.com/code/lua/apr/>

License: MIT

Version: 0.9.29

This Lua script is executed on `require("apr")`, loads the binary module using `require("apr.core")`, defines several library functions implemented on top of the binary module and returns the module table as the result of `require()`.

```
--]]
```

```
local apr = require 'apr.core'
```

```
apr._VERSION = '0.9.29'
```

```
-- apr.md5(input [, binary]) -> digest {{{1
```

```
--  
-- Calculate the [MD5] [md5] message digest of the string @input. On success  
-- the digest is returned as a string of 32 hexadecimal characters, or a string  
-- of 16 bytes if @binary evaluates to true. Otherwise a nil followed by an  
-- error message is returned.
```

```
-- *This function is binary safe.*
```

```
-- Part of the "Cryptography routines" module.
```

```
function apr.md5(input, binary)
```

```
  assert(type(input) == 'string', "bad argument #1 to apr.md5() (string expected)")
```

```
  local context, digest, status, errmsg, errcode
```

```
  context, errmsg, errcode = apr.md5_init()
```

```
  if context then
```

```
    status, errmsg, errcode = context:update(input)
```

```
    if status then
```

```
      digest, errmsg, errcode = context:digest(binary)
```

```
      if digest then return digest end
```

```
    end
```

```
  end
```

```
  return nil, errmsg, errcode
```

```
end
```

```
-- apr.sha1(input [, binary]) -> digest {{{1
```

```
--  
-- Calculate the [SHA1] [sha1] message digest of the string @input. On success  
-- the digest is returned as a string of 40 hexadecimal characters, or a string  
-- of 20 bytes if @binary evaluates to true. Otherwise a nil followed by an  
-- error message is returned.
```

```
-- *This function is binary safe.*
```

```
-- Part of the "Cryptography routines" module.
```

```
function apr.sha1(input, binary)
```

```
  assert(type(input) == 'string', "bad argument #1 to apr.sha1() (string expected)")
```

```
  local context, digest, status, errmsg, errcode
```

```
  context, errmsg, errcode = apr.sha1_init()
```

```
  if context then
```

```
    status, errmsg, errcode = context:update(input)
```

```

    if status then
        digest, errmsg, errcode = context:digest(binary)
        if digest then return digest end
    end
end
return nil, errmsg, errcode
end

-- apr.filepath_which(program [, find_all]) -> pathname {{{1
--
-- Find the full pathname of @program by searching the directories in the
-- [$PATH] [path_var] environment variable and return the pathname of the
-- first program that's found. If @find_all is true then a list with the
-- pathnames of all matching programs is returned instead.
--
-- [path_var]: http://en.wikipedia.org/wiki/PATH\_%28variable%29
--
-- Part of the "File path manipulation" module.

function apr.filepath_which(program, find_all)
    local split = apr.filepath_list_split
    local is_windows = apr.platform_get() == 'WIN32'
    local extensions = is_windows and split(apr.env_get 'PATHEXT')
    local results = find_all and {}
    for _, directory in ipairs(split(apr.env_get 'PATH')) do
        local candidate = apr.filepath_merge(directory, program)
        if apr.stat(candidate, 'type') == 'file' then
            -- TODO if not is_windows check executable bits
            if not find_all then return candidate end
            results[#results + 1] = candidate
        end
        if is_windows and #extensions >= 1 then
            for _, extension in ipairs(extensions) do
                candidate = apr.filepath_merge(directory, program .. '.' .. extension)
                if apr.stat(candidate, 'type') == 'file' then
                    if not find_all then return candidate end
                    results[#results + 1] = candidate
                end
            end
        end
    end
    return results
end

-- apr.glob(pattern [, ignorecase]) -> iterator {{{1
--
-- Split @pattern into a directory path and a filename pattern and return an
-- iterator which returns all filenames in the directory that match the
-- extracted filename pattern. The 'apr.fnmatch()' function is used for
-- filename matching so the documentation there applies.
--
-- *This function is not binary safe.*
--
-- Part of the "Filename matching" module.

function apr.glob(pattern, ignorecase)
    local fnmatch = apr.fnmatch
    local yield = coroutine.yield
    local directory, pattern = apr.filepath_parent(pattern)
    local handle = assert(apr.dir_open(directory))
    return coroutine.wrap(function()

```

```

for path, name in handle:entries('path', 'name') do
    if fnmatch(pattern, name, ignorecase) then
        yield(path)
    end
end
end
handle:close()
end)
end

-- apr.uri_encode(string) -> encoded {{{1
--
-- Encode all unsafe bytes in @string using [percent-encoding] [percenc] so
-- that the string can be embedded in a [URI] [uri] query string.
--
-- [percenc]: http://en.wikipedia.org/wiki/Percent-encoding
--
-- Part of the "Uniform resource identifier parsing" module.

function apr.uri_encode(s)
    local byte = string.byte
    local format = string.format
    return (s:gsub('[^A-Za-z0-9_.-]', function(c)
        if c == ' ' then
            return '+'
        else
            return format('%%%02x', byte(c))
        end
    end))
end
end

-- apr.uri_decode(encoded) -> string {{{1
--
-- Decode all [percent-encoded] [percenc] bytes in the string @encoded.
--
-- [percenc]: http://en.wikipedia.org/wiki/Percent-encoding
--
-- Part of the "Uniform resource identifier parsing" module.

function apr.uri_decode(s)
    local char = string.char
    local tonumber = tonumber
    s = s:gsub('+', ' ')
    return (s:gsub('%%%?x?', function(code)
        return char(tonumber(code, 16))
    end))
end
end

-- }}}1

return apr

-- vim: ts=2 sw=2 et tw=79 fen fdm=marker

```