local M = {}

local defaults = { pandoc = {}, reader = {}, }

function M.setup(opts) if opts and opts.pandoc then defaults.pandoc.outfmt = opts.pandoc.outfmt or “pdf” defaults.pandoc.outfile_prefix = opts.pandoc.outfile_prefix or “MarkupPreviewer” else defaults.pandoc.outfmt = “pdf” defaults.pandoc.outfile_prefix = “MarkupPreviewer” end

```
if opts and opts.reader then
    defaults.reader.cmd = opts.reader.cmd
else
    defaults.reader.cmd = "zathura"
end


vim.api.nvim_create_user_command("MarkupPreviewer",
    function(t)
        local input
        local adhoc_output
        local adhoc_reader = {}
        local case = {
            [1] = function(arg) input = arg end,
            [2] = function(arg) adhoc_output = arg end,
            default = function(arg) table.insert(adhoc_reader, arg) end
        }
        for i, val in ipairs(t.fargs) do
            if case[i] then
                case[i](val)
            else
                case.default(val)
            end
        end

        M.preview(input, adhoc_output, adhoc_reader)
    end, {
    nargs = "+",
    complete = "file",
    desc = "require(\"mucv\").preview()",
    force = true,
})
```

end

— Convert a markup file into the given output format. —@param input string input markup file —@param outfmt? string pandoc file type —@return boolean|nil error —@return string errmsg the error message when the function returns false local function convert(input, outfmt) local pandoc = { cmd = { “pandoc”,

“–standalone” }, }

```
if outfmt then
    pandoc.outfmt = outfmt
else
    pandoc.outfmt = defaults.pandoc.outfmt
end

table.insert(pandoc.cmd, "--to")
table.insert(pandoc.cmd, pandoc.outfmt)

-- Needed by readers that depend on file extensions
local epub_ext = string.match(pandoc.outfmt, "epub%d*")
pandoc.outfile = table.concat({
    defaults.pandoc.outfile_prefix,
    ".",
    (epub_ext and "epub" or pandoc.outfmt)
})

table.insert(pandoc.cmd, "--output")
table.insert(pandoc.cmd, pandoc.outfile)
table.insert(pandoc.cmd, input)

pandoc.have_run, pandoc.pid = pcall(vim.fn.jobstart,
    pandoc.cmd,
    {
        on_exit = function(_, exitcode)
            pandoc.exitcode = exitcode
        end,
        stderr_buffered = true,
        on_stderr = function(_, data)
            pandoc.errmsg = data[1]
        end
    }
)

if not pandoc.have_run then return nil, "pandoc could not be executed" end

vim.fn.jobwait({ pandoc.pid }, -1)

if pandoc.exitcode ~= 0 then return nil, "pandoc: " .. pandoc.errmsg end

return true, pandoc.outfile
```

end

— Lauch the reader to read the pandoc generated file. —@param file string

pandoc generated file —@param adhoc_reader? table reader program —@return boolean|nil error —@return string errmsg local function read(file, adhoc_reader) local reader = {}

```
if adhoc_reader and adhoc_reader[1] then
    reader = { cmd = adhoc_reader[1], exec = adhoc_reader }
    table.insert(reader.exec, file)
else
    reader = { cmd = defaults.reader.cmd, exec = {}, }
    reader.exec = { reader.cmd, file }
end

reader.have_run, reader.pid = pcall(vim.fn.jobstart,
    reader.exec,
    {
        on_exit = function(_, exitcode)
            reader.exitcode = exitcode
        end,
        stderr_buffered = true,
        on_stderr = function(_, data)
            reader.errmsg = data[1]
        end
    }
)

if not reader.have_run then return nil, reader.cmd .. " could not be executed" end

vim.fn.jobwait({ reader.pid }, -1)

if reader.exitcode ~= 0 then
    return nil, table.concat({ reader.cmd, "--", reader.errmsg }, " ")
end

return true, _
```

end

— Check if a file is readable. —@param file string —@return boolean —@return string errmsg local function is_file_r(file) if not file then return false, “no file given” end

```
local f = io.open(file, "r")
if not f then return false, "could not read " .. file end
io.close(f)

return true, _
```

end

— Preview the given file with the given reader. —@param input string the file to be converted —@param adhoc_output? string overrides the default output type —@param adhoc_reader? table overrides the default reader function M.preview(input, adhoc_output, adhoc_reader)

```
local file_status, file_errmsg = is_file_r(input)
if not file_status then error(file_errmsg) end

local pandoc_status, retstr = convert(input, adhoc_output)
if not pandoc_status then error(retstr) end

local reader_status, reader_errmsg = read(retstr, adhoc_reader)
if not reader_status then error(reader_errmsg) end
```

end

return M