1 Minim

minim.opm

```
\_def\_optexminim_version{0.1}
\_codedecl \optexminim_loaded  {Minim compatibility for \OpTeX/ (v\_optexminim_version)}
```

When we want to use minim with OpTeX, we need to accomodate for their differences. Particularly in allocations. Because of Esger Renkema's graciousness, minim has been modified in ways that make it easier to integrate with OpTeX. So even when the two formats normally disagree on a lot of core things (allocations, callbacks) they can end up working together.

OpTeX defines most allocation macros in `alloc.opm` and some Lua allocation functions in `optex.lua`. Minim "packages" are not standalone, they all depend on core routines defined in `minim-alloc.tex` and `minim-alloc.lua` respectively. Minim as a format preloads a stripped version of `etex.src` so the Lua code makes some assumptions about that (i.e. expects local allocators).

Both OpTeX and minim want to make it possible to register more functions for a single callback, by chaining their calls and `callback.register()`ing only a proxy function. While minim stays close to the `callback` interface from LuaTeX, OpTeX is a subset of the LaTeX `luatexbase` interface. But minim now can work on top of ltluatex by using its functions instead of `callback.register()`, so we don't have to do anything to support it in OpTeX!

Firstly, because we only change what is defined by others, we actually need a dummy macro for `\_codedecl`, to prevent loading this file more than once.

minim.opm

```
\_catcode`\@=11
% dummy macro to signalize that we are loaded
\_let\optexminim_loaded=\empty
```

In general, there are four allocator types expected by minim:

- Knuth allocators from `plain.tex` (like `\newcount`). These are already defined by OpTeX. Although minim itself sets the old `\allocationnumber` counter, which is not even defined in OpTeX.
- Global allocators from `etex.src` (like `\globcount`). These are not defined by OpTeX, since they no longer make sense (LuaTeX doesn't use sparse arrays for registers). Minim defines them to be the classic Knuth allocators if it doesn't find them on the TeX side, but expects them from the Lua side.
- Local allocators from `etex.src` (like `\loccount`). Concept of local allocators is completely missing in OpTeX. The semantics of local allocation in Lua is weird too, so we try to avoid these, since minim also doesn't define their Lua variants.
- LuaTeX allocators from `ltluatex.tex` (like `\newattribute`). Subset of these is in OpTeX (only atributes, which are also allocatable in Lua, and catcode tables). But minim tries to be compatible with LaTeX and patches its routines if it detects them.

For defining TeX commands implemented in Lua, OpTeX has `define_lua_command`, which actually does the allocation and definition at the same time, and allows to do so only from Lua end.

Historically (in my opinion unfortunately) LaTeX made the allocations of these functions available from TeX end, and the "lua define" operation is thus a two step process which involves synchronization with TeX.

`minim-alloc` actually defines a `luadef` function which is like `define_lua_command`, but is backed by the minim allocator. To make this work, we just need to set the LaTeX register to the index of last allocated function, since it allocates at counter plus one. Then minim will start where OpTeX stopped, and we will later define `define_lua_command` to be just minim's `luadef`.

We tell the number of allocated function by going through the table of actually used functions. This is not that robust, because while `define_lua_command` allocates sequentially, the provided functions may be `nil`, which breaks the code below.

minim.opm

```
\_newcount\allocationnumber

% for synchronisation of allocated Lua functions
\_ea\_newcount\_csname e@alloc@luafunction@count\_endcsname

\directlua{
    local function_table = lua.get_functions_table()
    % minim allocates at count + 1 for "new" allocators, so this works
    tex.setcount("global", "e@alloc@luafunction@count", \csstring\#function_table)
}
```

We also get PDF resources out of the way now. Minim is ready to use OpTeX's PDF resource management, but also has compatibility layer for PGF, which is not needed in OpTeX. We prevent loading the problematic TeX file.

minim.opm

```
\_let\minimpdfresourcesloaded=\_empty
```

The preparations are over. We load `minim-alloc.tex`.

minim.opm

```
\input minim-alloc
```

Both LaTeX and the minim inspired catcode table allocators initialize the catcode tables with `\initcatcodetable` (i.e. iniTeX catcodes). OpTeX merely allocates the registers. LuaTeX doesn't allow to activate unitialized catcode table, therefore activation with either `\initcatcodetable` or `\savecatcodetable` is necessary before use. To ensure compatibility with foreign macros, we also issue `\initcatcodetable` on allocation in the public version of `\newcatcodetable`.

minim.opm

```
\_def\newcatcodetable#1{\_newcatcodetable#1\_initcatcodetable#1}
```

By now, the Knuthian allocators are dealt with. ε-TeX global and local allocators are undefined, but the global ones are expected in minim's Lua code with their hardcoded counter register numbers. This is unacceptable, since in this range (`\count260` to `\count266`) OpTeX has already made allocations. Thus we need to replace these Lua functions with similar definitions. For some, OpTeX also has a different idea whether the counter represent the last or next allocated register number, so we correct that as well.

We also don't forget to actually set `define_lua_command` to be minim's `luadef`.

minim.opm

```
\directlua{
    local minimalloc = require("minim-alloc")

    define_lua_command = minimalloc.luadef

    % these are allocators already defined in OpTeX that we need to repair
    local toreplace = {
        "count",
        "dimen",
        "skip",
        "muskip",
        "box",
        "toks",
        "marks",
        "attribute",
        "catcodetable",
    }

    for _, alloc in ipairs(toreplace) do
        local cache = {}
        local countername = string.format("_\_pcent salloc", alloc)
        minimalloc["new_"..alloc] = function(id)
            local n = cache[id]
            if not n then
                n = tex.getcount(countername) + 1
                tex.setcount("global", countername, n)
                if id then
                    cache[id] = n
                end
                minimalloc.log(
                    "\_nbb\_pcent s\_pcent d : \_pcent s", alloc, n, id or "<unnamed>")
            end
            return n
        end
    end
}
```

We also need to do something about `minim-hooks.tex`, which hooks into `\shipout`, but the default OpTeX output routine (and perhaps also the user ones) use `\_shipout`.

Minim also adds to `\everypar`, but that is fine.

minim.opm

```
\_let\shipout\_shipout

\input minim-hooks

\_catcode`\:=11
\_let\_shipout\minim:shipout:new

% catcodes changes don't propage, since this file is loaded with \opinput
```

2 Minim-mp

minim-mp.opm

```
\_codedecl \_optexminimmp_used {Minim-PDF for \OpTeX/}
\_namespace{optexminimmp}

\_def\.used{}

\_load[minim]

\_input minim-mp

\_endnamespace
```

3 Minim-PDF

minim-pdf.opm

```
\_codedecl \nohyphlang {Minim-PDF for \OpTeX/}
\_namespace{optexminimpdf}
```

Before loading `minim-pdf` we do a few preparations. Most importantly adjusting core of minim, which is done in `minim.opm`.

minim-pdf.opm

```
\_load[minim]

\_catcode`\@=11
\_catcode`\:=11
```

Minim hooks into language mechanism with standard ε-TeX `\uselanguage@hook`. It can then translate language names (e.g "SwissGerman") into BCP 47 language identifiers (e.g. "de-ch-1901", the naming scheme of the `hyph-utf8` package) which are then embedded into tagged PDFs. For simplicity, we use what OpTeX calls "lang-tag" (e.g. "de") which may not be precise enough (i.e. the right thing for Swiss German would be "de-ch") and may even be wrong(?).

When e.g. `\gswlang` (Swiss German) is used, `\uselang{gsw}\_gswPatt`⟨...⟩ is called in OpTeX. We use the fact, the there is mapping from language register numbers (`\_gswPatt`) to "lang-tag"s ("de"). To avoid confusion with any other language mappings, we prefix what we give to minim with `o:`.

We also don't incorporate the ε-TeX `\uselanguage@hook`, but call minim's callback directly, maybe it will cause less trouble with other packages (since we don't pass standard names to the callback) and also gives a better idea of what happens.

minim-pdf.opm

```
\_let\.uselang=\_uselang
\_def\_uselang#1#2#3#4{%
   \.uselang{#1}{#2}{#3}{#4}%
   % just set this everytime, doesn't hurt
   \setlanguagecode{o:#1}{\_cs{_lan:\_the#2}}%
   \minim:uselanguagecallback{o:#1}%
}
```

If not detected, a few "dummy" languages would be (in erroneous ways) defined by minim: like "nohyph" and "undetermined". We define a few dummy control sequences, to make minim not define them, since we define them ourselves below. They are used in standard way, but their "ISO codes" are weird:

```
\nohyphlang
\nolanglang
\uncodedlang
\undeterminedlang
```

minim-pdf.opm

```
\_def\_tmp{}
\_let\lang@nohyph=\_tmp
\_let\lang@nolang=\_tmp
\_let\lang@uncoded=\_tmp
\_let\lang@undetermined=\_tmp

\_preplang nohyph       nohyph       und {} 11
\_preplang nolang       nolang       und {} 11
\_preplang uncoded      uncoded      und {} 11
\_preplang undetermined undetermined und {} 11
```

Now we actually load `minim-pdf`.

minim-pdf.opm

```
\_input minim-pdf
```

Users aren't supposed to define custom languages in OpTEX, forbid that.

minim-pdf.opm

```
\_def\_tmp{\errmessage{don't use this command with OpTeX}}
\_let\newnameddialect=\_tmp
\_let\newnameddllanguage=\_tmp
```

Since a language may already be set (at least the default Knuth english), then we need to tell minim about it, by reexecuting the language command (like `\enlang`), thus calling into minim through the above mentioned hook.

minim-pdf.opm

```
% set the current language again to let minim know what it is
\_cs{\_cs{_lan:\_the\_language}lang}


% catcodes changes don't propage, since this file is loaded with \opinput
\_endnamespace
```