1. CCBASE. This package provides the base macros and Lua module for my LuaTEX setup.

2. The package files.

⟨ *{ccbase.tex} 2 ⟩ ≡
⟨ TEX package preamble 4 ⟩
⟨ TEX global variables 16 ⟩
⟨ TEX macros 6 ⟩
⟨ TEX package postamble 5 ⟩

3. ⟨ *{ccbase.lua} 3 ⟩ ≡
```
local exports = {}
```
⟨ Lua global variables 20 ⟩
⟨ Lua functions 21 ⟩
```
return exports
```

4. ⟨ TEX package preamble 4 ⟩ ≡
⟨ Include guards 7 ⟩
```
\input eplain
\directlua{ccbase = dofile(kpse.find_file("ccbase.lua"))}
\makeatletter
```
This code is used in section 2.

5. ⟨ TEX package postamble 5 ⟩ ≡
```
\makeatother
\endinput
```
This code is used in section 2.

6. ⟨ TEX macros 6 ⟩ ≡
```
\def\typeout{\immediate\write17}
```
See also sections 8, 9, 10, 12, 15, 18, and 19.
This code is used in section 2.

7. Include guards. We define a command `\pragmaonce{id}` that expands to nothing the first time it is called, and to `\endinput` otherwise. This is useful to provide include guards to our package files. Since we want to include guard `CCBASE` itself, this is the first thing we define, and the first thing we use.

⟨ Include guards 7 ⟩ ≡

```
\def\pragmaonce#1{
  \csname pragmaonce#1\endcsname%
  \global\expandafter\let\csname pragmaonce#1\endcsname=\endinput
}
\pragmaonce{ccbase}
```

This code is used in section 4.

8. Catcodes and verbatim. The definitions below are copied from `tugboat.cmn`, including the documentation: The following allow for easier changes of category. These require that the character be addressed as a control-sequence: e.g. `\makeescape\/` will make the `/` an escape character.

⟨ TEX macros 6 ⟩ +≡

```
\def\makeescape#1{\catcode`#1=0 }
\def\makebgroup#1{\catcode`#1=1 }
\def\makeegroup#1{\catcode`#1=2 }
\def\makemath#1{\catcode`#1=3 }
\def\makealign#1{\catcode`#1=4 }
\def\makeeol#1{\catcode`#1=5 }
\def\makeparm#1{\catcode`#1=6 }
\def\makesup#1{\catcode`#1=7 }
\def\makesub#1{\catcode`#1=8 }
\def\makeignore#1{\catcode`#1=9 }
\def\makespace#1{\catcode`#1=10 }
\def\makeletter#1{\catcode`#1=11 }
\def\makeother#1{\catcode`#1=12 }
\def\makeactive#1{\catcode`#1=13 }
\def\makecomment#1{\catcode`#1=14 }
```

9. Eplain defines the `\verbatim .. |endverbatim` construction, but it is much more convenient to use the LaTeX-like `\verb+...+` one.

⟨ TEX macros 6 ⟩ +≡

```
\def\verb{\begingroup\uncatcodespecials\@ccbaseverb}
\def\@ccbaseverb#1{\tt\def\@@ccbaseverb##1#1{##1\endgroup}\@@ccbaseverb}
```

10. Cross-references. Eplain defines a couple of macros for cross-referencing pages, but none of them produce a page number that can be used as an argument to `\ifodd`, for example. Using Eplain's internal `\xrlabel`, however, we can easily create one.

The first argument to `\ccxrefn` is what the macro should expand to if the label is not defined.

⟨ TEX macros 6 ⟩ +≡

```
\def\ccxrefn[#1]#2{
  \expandafter \ifx\csname\xrlabel{#2}\endcsname\relax#1
  \else\csname\xrlabel{#2}\endcsname
  \fi
}
```

11. Bibliography. There are situations when I need to cite multiple references, while pointing to different page or section numbers for each. The optional argument to `\cite` only gives me one note, so I can't use that. The macros below allow me to add optional arguments to each of the `\cite` arguments. They are built by reverse-engineering `btxmac`'s `\@cite` macro, which does all the processing.

12. The first thing we need to modify is the `\@onecitation` macro, which is the one that does the printing. We redefine it to accept an optional argument, just like `\cite`, and print it after the reference itself.

⟨ TEX macros 6 ⟩ +≡
```
\let\@btxonecitation\@onecitation%
\def\@onecitation#1\@@{%
  \expandafter\@btxonecitation%
    \directlua{⟨ Remove optional argument from #1 list 13 ⟩}\@@%
  \directlua{⟨ Get optional argument from #1 14 ⟩}%
}
```

13. String processing is done better in Lua:

⟨ Remove optional argument from #1 list 13 ⟩ ≡
```
local p=string.char(37)
local r,n=string.gsub("#1",p.."[[^]]+"..p.."]","")
tex.print(r)
```
This code is used in sections 12 and 15.

14. ⟨ Get optional argument from #1 14 ⟩ ≡
```
local p=string.char(37)
local r,n=string.gsub("#1",p.."[([^]]+)"..p.."].*",p.."1")
if n > 0 then tex.print(r) end
```
This code is used in section 12.

15. Next, we need to modify the `\nocite` macro, which is called by `\@cite` with the raw list of arguments. All we have to do here is to remove the optional arguments from the list.

⟨ TEX macros 6 ⟩ +≡
```
\let\@btxnocite=\nocite
\def\nocite#1{%
  \@btxnocite{\directlua{⟨ Remove optional argument from #1 list 13 ⟩}}%
}
```

16. Two-sided printing. To enable two-sided printint layout, where horiziontal margins alternate between odd and even pages, issue `\twosidetrue` after including `CCBASE`.

⟨ TEX global variables 16 ⟩ ≡
```
\newif\iftwoside
\twosidefalse
```
See also section 17.

This code is used in section 2.

17. We're going to replace the output routine so it can change margins at every page. To do that, we save the old ouptut routine in `\ccbase@prevoutput`.

⟨ TEX global variables 16 ⟩ +≡
```
\newtoks\ccbase@prevoutput
\ccbase@prevoutput=\expandafter{\the\output}
```

18. In TEX, the `\hoffset` value is how much, past 1in of the left border, does the text area start. The amount of horizontal text space available is `\hsize`. Eplain also keeps the total paper width dimension in `\paperwidth`. With that, we have

$$w = 1\text{in} + h_{\text{offset}} + h_{\text{size}} + h_{\text{right}},$$

so that the right margin of a page is given by

$$h_{\text{right}} = w - 1\text{in} - h_{\text{offset}} - h_{\text{size}}.$$

We want this to be the new left margin, which is $h_{\text{offset}} + 1\text{in}$, so what we have to do is to replace, at every page, `\hoffset` with

$$h_{\text{offset}} \leftarrow w - h_{\text{size}} - h_{\text{offset}} - 2\text{in}.$$

The resulting output routine follows.

⟨ TEX macros 6 ⟩ +≡
```
\output={%
  \the\ccbase@prevoutput%
  \iftwoside%
    \global\advance\hoffset by -2\hoffset%
    \global\advance\hoffset by \paperwidth%
    \global\advance\hoffset by -\hsize%
    \global\advance\hoffset by -2truein%
  \fi}
```

19. Inline Lua code. This comes directly from LuaTEX's "Writing Lua in TeX" page, using the catcode routines defined before for simplicity. It introduces two macros, `\luacode` and `\endluacode`, that are used as a begin-end environment.

To syntax highlight Lua code inside TeX, create a `$VIMFILES/after/syntax/plaintex.vim` file with the following contents:

```
unlet b:current_syntax
syn include @LUA syntax/lua.vim

syn region luatex matchgroup=contextIdentifier
    \ start='\\luacode'
    \ end='\\endluacode'
    \ contains=@LUA
```

The reason I define `\luacode` below with an `\expandafter` is to make Vim properly syntax-highlight `CCBASE` itself.

⟨ TEX macros 6 ⟩ +≡

```
\expandafter\def\csname luacode\endcsname{
  \bgroup
  \makeother\{
  \makeother\}
  \makeother\^^M
  \makeother\#
  \makeother\~
  \makeother\%
  \doluacode
}
\bgroup
\makeother\^^M %
\long\gdef\doluacode#1^^M#2\endluacode{\directlua{#2}\egroup}%
\egroup
```

20. LuaTeX nodes. TEX entities are represented in LuaTEX as nodes of different types. Here we define a few global variables that make type identification more efficient later on.

⟨ Lua global variables 20 ⟩ ≡

```
local GLUE_TYPE    = node.id("glue")
local GLYPH_TYPE   = node.id("glyph")
local HLIST_TYPE   = node.id("hlist")
local KERN_TYPE    = node.id("kern")
local MATH_TYPE    = node.id("math")
local RULE_TYPE    = node.id("rule")
local VLIST_TYPE   = node.id("vlist")
local WHATSIT_TYPE = node.id("whatsit")
```

See also sections 22 and 23.

This code is used in section 3.

21. ⟨ Lua functions 21 ⟩ ≡

See also sections 24, 25, and 26.

This code is used in section 3.

22. ⟨ Lua global variables 20 ⟩ +≡

```
exports["GLUE_TYPE"]    = GLUE_TYPE
exports["GLYPH_TYPE"]   = GLYPH_TYPE
exports["HLIST_TYPE"]   = HLIST_TYPE
exports["KERN_TYPE"]    = KERN_TYPE
exports["MATH_TYPE"]    = MATH_TYPE
exports["RULE_TYPE"]    = RULE_TYPE
exports["VLIST_TYPE"]   = VLIST_TYPE
exports["WHATSIT_TYPE"] = WHATSIT_TYPE
```

23. Dimensions.

⟨ Lua global variables 20 ⟩ +≡
```
local dims = {
  ["sp"] = 1,
  ["pt"] = 2^16,
  ["pc"] = 12*2^16,
  ["bp"] = 72*2^16,
  ["in"] = 72.27*2^16,
}
```

24. ⟨ Lua functions 21 ⟩ +≡
```
local function dim2str(value,from,to)
  return string.format("%f"..to,value*dims[from]/dims[to])
end
exports["dim2str"] = dim2str
```

25. ⟨ Lua functions 21 ⟩ +≡
```
local function str2dim(value,to)
  value = value:gsub("^[ \t]*","")
  value = value:gsub("[ \t].*$","")
  local from = value:gsub("[-0-9.]+","")
  value = value:gsub("[^-0-9.]+","")
  return tonumber(value)*dims[from]/dims[to]
end
exports["str2dim"] = str2dim
```

26. ⟨ Lua functions 21 ⟩ +≡
```
local function mkglue(w,st,sto,sh,sho)
  local glue = node.new(ccbase.GLUE_TYPE)
  glue.spec = node.new("glue_spec")
  glue.spec.width = w
  glue.spec.stretch = st
  glue.spec.stretch_order = sto
  glue.spec.shrink = sh
  glue.spec.shrink_order = sho
  return glue
end
exports["mkglue"] = mkglue
```

CCBASE