LuaCJK 宏包
The LuaCJK package*

Mingyu Xia†

Released 2026-11-26 v0.1.1

摘要 | Abstract

LuaCJK 为 LuaTEX 提供中日韩（CJK）文字的排版。它把当前的 CJK 字体记录在一个 LuaTEX 属性里，并在文字定形之前把每个 CJK 字形放进该字体。

LuaCJK typesets Chinese, Japanese, and Korean (CJK) with LuaTEX. It records the current CJK font in a LuaTEX attribute and sets each CJK glyph in that font before the text is shaped.

目录 Contents

1 简介 **4**

2 加载 **4**

3 字体 **6**

3.1 子区块 7
3.2 后备字体 8
3.3 基线调整 8
3.4 加载 LXGW 9
3.5 伪粗体与伪斜体 9
3.6 数学模式中的 CJK 字符 10

*https://ctan.org/pkg/luacjk, https://github.com/myhsia/LuaCJK
†xiamingyu@westlake.edu.cn

4 文本排版 11
4.1 间距 . . . 11
4.2 标点 . . . 13
4.3 断行 . . . 17
4.4 暂停与钩子 . . . 18
4.5 抄录与 listings . . . 18
4.6 装饰 . . . 19

5 与 xeCJK 的比较 23
5.1 行为差异 . . . 23
5.2 未提供的功能 . . . 24

1 Introduction 25

2 Loading 25

3 Fonts 27
3.1 Sub-blocks . . . 28
3.2 Fallback fonts . . . 29
3.3 Baseline shift . . . 30
3.4 Loading LXGW . . . 30
3.5 Fake bold and slant . . . 30
3.6 CJK in mathematics . . . 32

4 Text layout 33
4.1 Spacing . . . 33
4.2 Punctuation . . . 35
4.3 Line breaking . . . 40
4.4 Suspending and hooks . . . 41
4.5 Verbatim and listings . . . 42
4.6 Decorations . . . 43

5 Comparison with xeCJK 47
5.1 Different behavior . . . 47
5.2 Not available . . . 48

A Implementation **49**
A.1 Loading and conflicts . . . 49
A.2 Registers, styles, and options . . . 50
A.3 Fonts . . . 75
A.4 Names and configuration . . . 97
A.5 The Lua module . . . 100
A.6 Settings and data . . . 103
A.7 Node rewriting . . . 119
A.8 The passes . . . 137
A.9 Decorations . . . 191
A.10 Listings and ctex . . . 238

Index **247**

Change History **267**

1 简介

LuaCJK 是一个只用于 LuaTeX 的中日韩（CJK）排版宏包。它的主要接口是 `\setCJKmainfont`、`\setCJKsansfont`、`\setCJKmonofont`、`\setCJKfamilyfont`、`\CJKfamily` 与 `\LuaCJKsetup`。

当前的 CJK 字体记录在一个 LuaTeX 属性里，每个 CJK 字形在文字定形之前就放进该字体。

中文 日本語 **한국어**	中文日本語한국어

2 加载

`\usepackage[`⟨*options*⟩`]{LuaCJK}`

LuaCJK 需要 LuaTeX 引擎。它与 luatexja 冲突：无论先后加载哪一个，都会报错。

`\LuaCJKversion` 打印 LuaCJK 的版本。

`\LuaCJKsetup` `\LuaCJKsetup {`⟨*key-value list*⟩`}`

配置 LuaCJK。这些键就是随后各节中记录的那些键。

这些键分为两组。

描述文字如何排版的设置只作用于当前分组，并且记录在字符本身上，所以分组中的文字即使在该分组已经结束之后才断行，也仍然保留它的设置：

```
\vbox{\begingroup\LuaCJKsetup{CJKglue = 10pt}中中\endgroup\par}
```

用的就是 10pt 的胶，尽管分组在 `\par` 之前已经关闭。这一组包含

- `CJKglue`, `CJKecglue`, `CJKspace` 与 `xCJKecglue`;
- `PunctStyle`, `PunctKinsoku`, `AllowBreakBetweenPuncts`, `PunctWidth` 与 `PunctBoundWidth`;
- `LatinPunct` 与 `CJKmath`;
- `CheckSingle`, `WidowPenalty` 与 `CJLineBreak`;
- `AutoFallBack` 与 `Verb`;
- 两个命令 `\LuaCJKCancelSubCJKBlock` 与 `\LuaCJKRestoreSubCJKBlock`。

其余的键是全局的，或者在加载宏包时起作用：

- KaiMingPunct，MiddlePunct，LongPunct 与 NoBreakLongPunct 修改字符集合；
- AutoFakeBold，AutoFakeSlant，EmboldenFactor 与 SlantFactor 在注册字族时起作用；
- LocalConfig 在加载宏包时起作用，LoadLXGW 在 \begin{document} 处起作用。

声明 \LuaCJKDeclareCharClass、\LuaCJKDeclarePunctClass、\LuaCJKDeclareSubCJKBlock、\LuaCJKsetwidth 与 \LuaCJKsetkern 同样是全局的，这一点在各自的说明里都提到了。

同样的键也可以作为宏包选项给出：

```
\usepackage[PunctStyle = banjiao]{LuaCJK}
```

LocalConfig = ⟨**true**|*false*|*name*⟩ 从配置文件读取本地设置。true 加载 luacjk.cfg，⟨*name*⟩ 加载 luacjk-⟨*name*⟩.cfg，而 false（初始值）什么都不加载。文件在宏包末尾读取，所以它可以使用 LuaCJK 定义的每一个命令。文件不存在时只给出警告。

要在 ctex 中使用 LuaCJK，把它的 engine 选项设为 ⟨*luacjk*⟩：

```
\documentclass[engine = luacjk]{ctexart}
```

ctex 于是载入 LuaCJK 的 ctex-engine-luacjk.def 而不是它自己的 LuaTEX 后端，这个文件同时载入宏包，LuaCJK 不必再单独载入。

LuaCJK 与 LATEX 的标记 PDF（tagged PDF）项目协同工作。文档开头声明

```
\DocumentMetadata{tagging = on}
```

时，装饰的墨迹标记为 artifact，装饰符号不会出现在复制和选择的文本中；selectable = true 保留符号的文字。在 l3doc 文档中还要载入 latex-lab-testphase-l3doc，本手册就是这样生成的。选区的高度由 PDF 阅读器按字体的 hhea 上伸部和下伸部决定，对 Noto Serif CJK SC 是 1.437em，所以选区比 1em 的方块更高；这不是 LuaCJK 能改变的，也不影响字形本身。

3 字体

主要的字体命令接受一个可选的 fontspec 选项列表。选项既可以写在字体名之前，也可以写在字体名之后。无论西文使用哪一种编码，CJK 字体都在 Unicode 编码下选取，所以用 fontenc 把西文设为 T1 之类的旧编码不会影响 CJK 字符。

\setCJKmainfont
\setCJKsansfont
\setCJKmonofont

`\setCJKmainfont [⟨options⟩] {⟨font⟩} [⟨options⟩]`

为罗马、无衬线与等宽字族设置 CJK 字体。

```
\setCJKmainfont[Scale = 1.2]
  {LXGWNeoZhiSong.ttf}
English 中文字体\\
\setCJKsansfont[Scale = 1.2]
  {LXGWWenKaiGBLite-Regular.ttf}
\sffamily English 中文字体
```

English 中文字体
English 中文字体

\setCJKfamilyfont

`\setCJKfamilyfont{⟨family⟩}[⟨options⟩]{⟨font⟩}[⟨options⟩]`

定义一个具名的 CJK 字族。字族可以用 \CJKrmdefault 之类的名字给出。

\CJKfamily

`\CJKfamily {⟨family⟩}`

切换到具名的 CJK 字族。字族可以用 \CJKrmdefault 之类的名字给出。

```
\CJKfamily{zhsong}English 中文字体\\
\CJKfamily{zhhei}English 中文字体\\
\CJKfamily{zhkai}English 中文字体\\
\CJKfamily{zhfs}English 中文字体
```

English 中文字体
English 中文字体
English 中文字体
English 中文字体

\newCJKfontfamily

`\newCJKfontfamily[⟨family⟩]⟨command⟩ [⟨options⟩]{⟨font⟩}[⟨options⟩]`

定义一个切换到新声明字族的命令。与 \newfontfamily 一样，该命令只作用于当前分组。选项可以写在字体名之前，也可以写在之后。

\CJKfontspec

`\CJKfontspec [⟨options⟩] {⟨font⟩}`

为当前分组选择一个 CJK 字体。与 \fontspec 一样，选项写在字体名之前：这个命令用在正文中间，若它还要读取字体名之后的方括号，正文紧接着的那个方括号就会被当成选项而消失。

\songti
\heiti
\fangsong
\kaishu
\lishu
\youyuan

名为 zhsong、zhhei、zhfs、zhkai、 zhli 与 zhyou 的字族的快捷方式。

```
\newCJKfontfamily[kai]\mykai
  {LXGWWenKaiGBLite-Regular.ttf}
\mykai English 中文字体\\
\songti English 中文字体
```

English 中文字体
English 中文字体

\CJKrmdefault
\CJKsfdefault
\CJKttdefault
\CJKfamilydefault

由 \rmfamily、\sffamily 与 \ttfamily 选择的 CJK 字族，以及由 \normalfont 选择的字族。\CJKfamilydefault 默认跟随 \familydefault：主字族是三者之一时，它就是对应的那一族，否则它取 \CJKrmdefault。它在文档开始时选取，所以在导言区重定义它或 \familydefault，正文第一行就用新的字族。

3.1 子区块

一种 CJK 字体很少覆盖 Unicode 的每一个区块。子区块让一个范围之内的字符在一个字族里拥有自己的字体；这个区块随后成为字体命令的一个选项。

\LuaCJKDeclareSubCJKBlock \LuaCJKDeclareSubCJKBlock {⟨*block*⟩} {⟨*range list*⟩}

用给定的范围声明 ⟨*block*⟩，写法与 \LuaCJKDeclareCharClass 相同。这个名字随后成为一个字体选项，正如下例所示：设置了它的字族对该区块的字符使用第二个字体，没有设置它的字族则把这些字符留在主字体里。声明的范围同时并入 CJK 字符类别：字符即使不在内置的 CJK 范围内（例如带圈数字 U+2460 到 U+2473），只要属于某个区块，就按 CJK 字符排版。重叠的区块按声明顺序解决。

```
\LuaCJKDeclareSubCJKBlock{Kana}{3040 -> 30FF}
\setCJKmainfont[Kana = LXGWWenKaiGBLite-
Regular.ttf]
  {LXGWNeoZhiSong.ttf}
汉字 かんじ
```

汉字かんじ

\LuaCJKCancelSubCJKBlock \LuaCJKCancelSubCJKBlock {⟨*block list*⟩}
\LuaCJKRestoreSubCJKBlock \LuaCJKRestoreSubCJKBlock {⟨*block list*⟩}

在当前分组的剩余部分关闭或重新打开列出的区块。因为字体的选择是按字体标识符记录的，所以提供同一字体的两个字族共享同一个设置：最后选择该字体的字族说了算。

3.2 后备字体

一种 CJK 字体很少覆盖需要的每一个字符。当所选的 CJK 字体没有某个字符的字形时，后备字体按声明的顺序依次尝试。

\setCJKfallbackfamilyfont \setCJKfallbackfamilyfont {⟨*family*⟩} [⟨*options*⟩] {⟨*font*⟩} [⟨*options*⟩]

把 ⟨*font*⟩ 加入 ⟨*family*⟩ 的后备列表。字族可以用 \CJKrmdefault 之类的名字给出。多次调用该命令可以建立一条链；第一个拥有该字符的字体胜出。这条链按文字的字重、字形与字号解析，所以粗体里的字符取自后备字体的粗体字面，放大后的字符取自对应字号的字面。打开 CJKmath 时，排在数学模式里的 CJK 字符也走同一条链：列表取自负则这个数学字体的 CJK 字族，字号取数学样式的字号。

AutoFallBack = ⟨*true*|**false**⟩ 是否使用声明过的后备字体。这个开关像间距的那些键一样跟随文字，所以可以只在一个分组里打开它。

```
\setCJKmainfont{LXGWNeoZhiSong.ttf}
\setCJKfallbackfamilyfont{\CJKrmdefault}
  {LXGWWenKaiGBLite-Regular.ttf}
\LuaCJKsetup{AutoFallBack = true}中あ
```

中あ

3.3 基线调整

两种 CJK 字体很少是为彼此设计的：混排在同一行里，一种字体的字总显得偏高或偏低。一种字体可以带一个 Baseline 值，把它的字整体向上或向下移动，使两种字体立在同一条基线上。这个值属于字体而不属于文字，所以它是声明这个字体的那条命令的一个选项。

Baseline = ⟨*dimension*⟩ 把这个字体的字向上移动 ⟨*dimension*⟩，负值则向下移动。它和其他字体选项一样可以写在后备字体的声明里，这也是它最常见的用处：让后备字体贴住主字体的基线。单位可以用任何 TEX 长度单位，em 随字号一起变化。这个偏移记在字族排出的字上：两条声明给出同一个字体时，各自保留自己的值，在一个编组里选用的字族也只移动这个编组里的字。

```
\setCJKfamilyfont{high}[Baseline = .3em]
  {LXGWWenKaiGBLite-Regular.ttf}
\songti 基线\quad\CJKfamily{high}基线
```

基线　基线

3.4 加载 LXGW

LoadLXGW = ⟨**true**|*false*⟩ 默认打开。当文档没有声明任何 CJK 字体时，在 \begin{document} 处加载 LXGW 字体：霞鹜新致宋作为主字体，霞鹜新晰黑作为无衬线字体，朱雀仿宋作为等宽字体，并且把霞鹜新致宋、霞鹜新晰黑、霞鹜文楷、朱雀仿宋全部注册为具名字族。新致宋一族以文楷作为斜体，以霞鹜臻楷作为粗斜体，所以 \emph 与 \itshape 会把汉字改成楷体。新致宋与新晰黑都没有粗体，朱雀仿宋也没有，所以这些声明要求伪粗体，新晰黑与朱雀仿宋另外要求伪斜体；文楷一族则把霞鹜臻楷作为粗体。已经声明了字体的文档保留自己的字体，所以 LXGW 不会混进来；但若它恰好没有声明 \CJKfamilydefault 所用的那一族，正文就会用 \CJKrmdefault 那一族，若也没有则用最先声明的一族，并且给出警告。什么字体都没声明又关掉这个键时，同样会给出警告。

3.5 伪粗体与伪斜体

一种 CJK 字族常常没有粗体或斜体。下面这些键让 LuaCJK 请求 fontspec 加粗或倾斜常规字体，而不去使用粗体或斜体。它们作用于此后注册的字族，并且在字族自己设置了 BoldFont 或 ItalicFont 时不生效。它们是全局的：分组不会改变一个已经注册的字族所用的设置。一个既没有设置 ItalicFont 也没有打开 AutoFakeSlant 的字族没有斜体形状：强调的文字保持正体，fontspec 会报告缺少这个形状。

AutoFakeBold = ⟨*true*|**false**|*number*⟩ 在请求粗体形状时加粗常规字体。给出一个数字时同时设置 EmboldenFactor。

AutoFakeSlant = ⟨*true*|**false**|*number*⟩ 在请求斜体形状时倾斜常规字体。给出一个数字时同时设置 SlantFactor。

EmboldenFactor = ⟨*number*⟩: 加粗的量。初始值是 4。

SlantFactor = ⟨*number*⟩: 倾斜的量，介于 −0.999 与 0.999 之间。初始值是 0.167。

```
\LuaCJKsetup{AutoFakeBold = true}
\setCJKmainfont{LXGWNeoZhiSong.ttf}
\textbf{English 中文字体}\\
English 中文字体
```

English 中文字体
English 中文字体

\defaultCJKfontfeatures `\defaultCJKfontfeatures {⟨font features⟩}`

设置前置到此后注册的每一个 CJK 字族及其子区块与后备字体的特性。初始值是 `Script = CJK`。在调用之前注册的字族保留它们当时得到的特性。

\addCJKfontfeatures `\addCJKfontfeatures [⟨ignored⟩] {⟨font features⟩}`

为当前 CJK 字族添加特性，直到分组结束，例如 `\addCJKfontfeatures{Scale = 1.2}`。开头的一个 `*` 也可以接受；两种形式作用于同一个字体。

```
\defaultCJKfontfeatures{Scale = 1.5}
\setCJKfamilyfont{big}
  {LXGWNeoZhiSong.ttf}
\CJKfamily{big}中文字体\\
\CJKfamily{zhsong}中文字体\\
\addCJKfontfeatures{Scale = 1.2}
\CJKfamily{zhsong}中文字体
```

中文字体
中文字体
中文字体

3.6 数学模式中的 CJK 字符

CJKmath = ⟨*true*|**false**⟩：是否允许在数学模式里直接输入 CJK 字符。打开时，这样的字符用它自己的一个数学字族的 CJK 字体排出，而不是到数学字体里去找，所以上标或分数里的字符会取那个数学样式的尺寸。可以用 `\LuaCJKsetup{CJKmath = true}` 打开它，也可以用宏包选项 `CJKmath = true`。此时字符取数学版本（见 `\mathversion`）的字体：在 `bold` 版本里用字族的粗体，字族没有粗体时保持正体；字形取字族的正体，不随周围的正文。不打开它时，数学模式里的 CJK 字符会到数学字体里去找，通常就此消失，所以这种情况下 LuaCJK 给出一次性的警告。

\setCJKmathfont `\setCJKmathfont [⟨options⟩] {⟨font⟩} [⟨options⟩]`

选择数学模式里的字符所用的 CJK 字体。不用这个命令时，使用 CJK 主字体。本手册在前导区打开 CJKmath 并设置了这个字体。与其他的字体命令一样，这个选择只作用于当前分组：分组里的公式使用该分组里选择的字族。

数学模式里出现的每一个 CJK 字族都被注册成一个独立的数学字族，这样 TEX 就能取它的文本、脚本或脚脚本尺寸。字族按数学版本选字体：`bold` 版本用 `\bfdefault`，其余版本用 `\mddefault`，字形的形状取 `\shapedefault`。所以 `\boldmath` 会把公式里的 CJK 字符换成粗体，而且只作用于当前分组；字族没有粗体时保持正体，也不报警告。字族号取自 240–255 这一段，是 LuaTEX 允许的 256 个字族里的最后十六个，正常文档不会用到那里；数学里用到的每一个后备字体也各占一个号，声明了它的字族共用这个号，一个文档如果需要十六个以上

的号就会报错停止。脚本尺寸是当前正文尺寸的百分之七十五，也就是标准尺寸表里的比例。这个正文尺寸是公式周围文字的尺寸，而不是公式里 \text 临时切换到的尺寸，所以含有 \text 的公式与不含它的公式一样大。数学参数不取自 CJK 字体，因为它没有 MathConstants 表：这些参数仍然是文档数学字体的，这就是为什么两个 CJK 字符组成的分数用的是周围数学的横线与间距。

`$ 天地玄黄 $ and 天地玄黄`	$ 天地玄黄 $ and 天地玄黄

4 文本排版

4.1 间距

CJKglue = ⟨*glue*⟩: 插入相邻 CJK 字符之间的胶。初始值是 0pt plus 0.08\baselineskip，在每次选择字体时重新计算，所以它跟随字号；文档给出的值按原样使用，不随字号变化，连同它的伸缩阶（例如 plus 1filll）一起保留。

CJKecglue = ⟨*glue*⟩: 插入 CJK 与拉丁字符之间的胶。标点两侧的胶没有宽度：标点的进距本身就带有间距。半角的 ASCII 括号只留下开头括号之前的这一份胶，括号内侧不加，于是成对的括号贴住它括起来的文字。只排出一个开头或收尾定界符的公式同样如此：定界符贴住它括起来的文字，胶只加在成对定界符的外侧。初始值是 0.3333em plus 0.1667em minus 0.1111em，它同样跟随字号，字号取自与它相邻的 CJK 字符而不是拉丁文本：拉丁文本用 \scriptsize 或 \large 排时，两侧的间距仍是周围 CJK 字符的间距。

CJKspace = ⟨*true*|**false**⟩: 是否保留源文件中 CJK 字符之间的空格。不保留时，空格换成字符之间的胶，断行与伸缩都随之保留。用 \ 或 \nobreakspace 写出的空格不是源文件中的空格，它的宽度就是一个空格的宽度。

xCJKecglue = ⟨*true*|**false**⟩: 是否把源文件中 CJK 与拉丁字符之间的空格替换为 CJKecglue。

`中文 English 中文`	中文 English 中文
`\LuaCJKsetup{CJKspace = true}天 地 玄 黄 宇 宙`	天 地 玄 黄 宇 宙

\normalspacedchars `\normalspacedchars {⟨character list⟩}`

列出的字符两侧都不加胶，这样拉丁文字中的连字符或斜线不会把周围的文字拉开。初始列表是 / 与 -。这个设置是全局的。反斜线不在列表里：它紧邻 CJK 字符时通常是排出的命令名的开头，那就该像其他拉丁字符一样加胶，而不像路径中的分隔符那样不加。

中x中\par \normalspacedchars{x}中x中	中 x 中 中x中

只用于控制文字、本身没有进距的字符分两类。 U+200B、U+2060、U+180E 与 U+FEFF 不含字体能用的信息，在输入时就已删除，因此它们既不占胶，也不把两个 CJK 字符分开。连接符（U+200C、U+200D）与各个变体选择符留给字体：它们跟随前一个字符的字体，所以一个 emoji 序列不会被拆成两个图形，一个变体序列也能选到它的字形；字体用不到它们时留下的占位字形在断行之前就去掉，因此它们既不占进距，也不在两个 CJK 字符之间加上 CJKecglue。用 \char 请求的字符不会被删除，因为 \char 要的就是字形本身。盒子同样按它占的进距计：不占进距的盒子不把两侧的文字分开，所以 \rlap、\llap 与 \clap 覆盖在文字上的内容仍属于同一串。这样的盒子两侧的源码空格也属于同一个边界：紧邻后一个字符的空格成为这个边界的胶，它前面的空格不再占宽，所以两个空格与直接相邻时相同，仍是一个空格。盒边的类别取自盒内最靠近这条边的文字：盒内的字距、间距胶与标尺都不遮住它，所以中西文胶按文字本身的类别加在盒外；只有无限伸长的胶（目录里 \numberline 的 \hfil）让文字到不了盒边，那时盒外不加胶。竖排的盒子同样按这条边读：\underline 这样的盒子末尾是它下面的标尺，标尺不遮住它盖着的文字，所以这条文字两侧都加胶。盒边上的另一个盒子自己读不出类别时，它后面的文字就是最靠近这条边的文字： xcolor 在盒外另画的边框不遮住盒内的文字。彩色盒子的底色是它内容的背景，胶加在底色之前，所以底色只覆盖盒内的文字与留白，两侧的中西文胶都在底色之外。段落行的行首有一个记录段落设置的标记，它同样不遮住行首的字，所以 \parbox 的左边缘按其中第一行文字计；行末是 \parfillskip，文字到不了右边缘，那里不加胶。只有墨迹、没有文字的盒子按西文材料计，两侧都加胶。

CJKfilltwosides (*env.*) 把各行排进一个 minipage，盒中的行两端对齐到 ⟨*width*⟩，各行之间用 \\ 分开。它的 ⟨*position*⟩ 参数像 \parbox 一样决定盒子落在基线上的方式： t 让第一行的基线与周围的行齐平， c 表示居中（默认）， b 让最后一行的基线与周围的行齐平。环境像 \parbox 一样先离开竖直模式，所以它后面的文字接着盒子排在同一行，而不是另起一段；环境之后的空格与 xeCJK 一样被忽略。盒内的段落设置也是 \parbox 的：段首缩进和段间距为零，所以分段之后的每一段都从盒子左端起排，并与第一段一样两端对齐，行距是文档自己的行距，而不是外面某个对齐环境留下的值。正文里的 \footnote 排在盒子底部、横线之下，用盒子自己的计数器编号，与它在 minipage 里一样。宽度为零或为负时沿用周围的行宽。带星号的形式行为相同；它不与内容的自然宽度作比较。

```
\begin{CJKfilltwosides}[⟨position⟩]{⟨width⟩}
⟨lines⟩
\end{CJKfilltwosides}
```

```
\begin{CJKfilltwosides}{6em}
  瞻彼阕者，\\虚室生白
\end{CJKfilltwosides}
```

瞻彼阕者，
虚室生白

```
\begin{CJKfilltwosides*}{0pt}
  瞻彼阕者，\\虚室生白
\end{CJKfilltwosides*}
```

瞻 彼 阕 者 ，
虚 室 生 白

4.2 标点

PunctStyle = ⟨*style*⟩: 选择一种标点样式。已经实现的样式有 quanjiao（默认）、 plain、banjiao、kaiming、CCT 与 hangmobanjiao。未知的名字给出警告并回退到 plain。带固定宽度的样式不会切进墨迹：标点保留自己的墨迹与两侧较小的一份空白，所以一个墨迹几乎占满进距的标点（例如 U+FF05）保持全角。

PunctWidth = ⟨*length*|*false*⟩: 不管样式怎么说，给每个标点同样的进距。 0pt 也是一个进距，标点因此不占任何进距； false（初始值）让标点回到样式给出的宽度。用 \LuaCJKsetwidth 为某个标点设置的宽度仍然优先。

PunctBoundWidth = ⟨*length*|*false*⟩: 标点出现在行尾或行首时，它的进距中有多少留在行内；其余的墨迹悬挂到页边，或者悬挂到段落开头或列表项开头的缩进里。默认悬挂标点自带的空白，所以它的墨迹正好落在版心边上；断行时这一份悬挂已经算进去，行尾的标点因此能随它所在的行一起放下。居中的段落与右对齐的段落不从版心边上起行，所以它的开头与 \\ 强制换行之后的一行都不悬挂，否则整行会一起挪动；行尾照旧悬挂，段落最后一行也悬挂，墨迹落在右侧的对齐边上，于是这种段落的每一行都停在同一条边上。标点在一个盒子里时，盒子在标点之后若有墨迹（例如 \fbox 的框线），行的边就是盒子自己的边，标点不悬挂；盒子正好止于标点的进距时悬挂照旧，\mbox 即属此类。 0pt 把整个进距都悬挂出去，false 恢复这个默认值。

LatinPunct = ⟨*true*|**false**⟩: 把引号（U+2018、U+2019、U+201C、U+201D）、间隔号（U+00B7）、两点前导符（U+2025）、省略号（U+2026、U+22EF）与连字点（U+2027）当作拉丁标点：它们留在当前的拉丁字体里，也不被压缩。这样英文单词里的撇号就不会变成一个全角形式。破折号不受影响。

PunctFamily = ⟨*family*|*false*⟩: 为标点使用另一个 CJK 字族。该字族必须用 \setCJKfamilyfont 声明，可以用 \CJKsfdefault 之类的名字给出； false（初始值）让标点重新跟随文字。字族尚未声明时，标点仍用正文字体，并给出一条警告。该字族的 Baseline 同样移动它排出的标点。

AllowBreakBetweenPuncts = ⟨*true*|**false**⟩: 两个相邻的标点是否可以被断行分开。破折号与省略号不在此列，一串连续的破折号或省略号由 NoBreakLongPunct 控制。收尾的、中间的标点与破折号、省略号不能出现在行首，开头的标点不能出现在行尾，这两条禁则不由本选项放开，所以一个收尾的引号不会被单独排在行首。标点仍然会被压缩。

PunctKinsoku = ⟨**true**|*false*⟩: 是否对标点施加断行禁则。收尾的、中间的标点与破折号、省略号不能出现在行首，开头的标点不能出现在行尾，半角的 ASCII 括号同样如此：开头的不能出现在行尾，收尾的不能出现在行首；两个相邻的标点不会被分开，一串连续的破折号或省略号保持在一起。以空格隔开的两个标点不算相邻，那个空格仍是断行点，所以 `Try again. (Why?)` 可以在句点之后断行。盒子边缘的标点同样参与判断。这些禁则与 PunctStyle 无关，所以 `PunctStyle = plain` 时它们仍然有效。

```
不亦说乎，不亦乐乎\\
\LuaCJKsetup{PunctFamily = zhkai}
不亦说乎，不亦乐乎
```

不亦说乎，不亦乐乎
不亦说乎，不亦乐乎

```
\LuaCJKsetup{LatinPunct = false}%
a’a\quad 不亦说乎！\\
\LuaCJKsetup{LatinPunct, PunctWidth = .8em}
a’a\quad 不亦说乎！
```

a’ a　不亦说乎！
a’a　不亦说乎！

```
\fbox{甲，乙}\\
\LuaCJKsetup{PunctWidth = 0pt}%
\fbox{甲，乙}
```

甲，乙
甲乙

```
列位看官。\\
\LuaCJKsetup{PunctBoundWidth = .5em}
列位看官。\\
（列位看官
```

列位看官。
列位看官。
（列位看官

接下来是标点代码所用的字符集合。每个集合的键都有三种形式：不带符号的形式替换整个集合，带 + 的形式添加字符，带 - 的形式去掉字符。范围的写法与 \LuaCJKDeclareCharClass 相同。这些集合是全局的：分组不会把先前的成员带回来。

KaiMingPunct<+->=⟨*char list*⟩: 在 kaiming 样式里保持全角宽度的标点。初始为。、．、？、！。

MiddlePunct<+->=⟨*char list*⟩: 居中的标点，放在自己盒子的两半之间。居中的标点不能出现在行首。

LongPunct<+->=⟨*char list*⟩: 彼此之间从不压缩的标点，例如破折号与省略号。

NoBreakLongPunct<+->=⟨*char list*⟩: 不能被断行分开的长标点。初始与 LongPunct 相同。

\LuaCJKDeclareCharClass `\LuaCJKDeclareCharClass {⟨class⟩} {⟨range list⟩}`

改写一段码位的类别。范围用十六进制书写，前导的 " 可以省略，"START -> "END 表示一段范围；列表用逗号分隔。类别有：CJK 与 Default 让这段范围表现得像 CJK 或拉丁文字； FullLeft 与 HalfLeft 把它当作开头的标点， FullRight 与 HalfRight 把它当作收尾的标点，于是断行禁则适用；前者是全角标点，照样压缩与悬挂，后者只能是 ASCII 括号：它不压缩、不悬挂，成对的括号贴住括号内的文字，括号外则照常留中西文胶：开头括号之前一份，收尾括号之后一份。 NormalSpace 把这段范围移出 CJK/拉丁胶，就像 \normalspacedchars 那样； CJStarter、CM 与 Boundary 当作 CJK 处理，因为 LuaTEX 没有组合标记或文字段边界的模型。

\LuaCJKDeclarePunctClass `\LuaCJKDeclarePunctClass {⟨group⟩} {⟨range list⟩}`

改写一段范围的标点组。组有 open、close、 middle、dash 与 ellipsis，半角括号的两组 half-open 与 half-close，以及拉丁文收尾标点的一组 latin-close：这一组两侧的胶没有宽度，它的进距属于拉丁字体，所以不压缩、不悬挂， PunctWidth 也不改写它。 none 去掉标点处理，于是这些字符不再被压缩或悬挂。

\LuaCJKResetCharClass 丢弃迄今声明的每一个类别与标点改写，并把正常间距集合恢复为 \normalspacedchars 的字符。

```
中ABC中\par
\LuaCJKDeclareCharClass{CJK}{0041 -> 005A}
中ABC中\par
\LuaCJKDeclarePunctClass{none}{FF0C}%
字，，字\par
\LuaCJKResetCharClass 中ABC中
```

中 ABC 中
中ABC中
字，，字
中 ABC 中

\LuaCJKEditPunctStyle `\LuaCJKEditPunctStyle {⟨style⟩} {⟨key-value list⟩}`

修改一个已有样式的参数。键是 compress-adjacent, adjacent-ratio, keep-open-close；没有给出的键保持它原来的值。

\LuaCJKsetwidth `\LuaCJKsetwidth {⟨punctuation list⟩} {⟨length⟩}`

给列出的标点一个固定的进距，覆盖所选的 PunctStyle 与 PunctWidth。这个设置是全局的：分组不会恢复先前的宽度，后一次调用是在表中添加，而不是替换。开头的一个 * 可以接受；它会在行首或行尾设置的宽度没有实现，星号只警告一次。

\LuaCJKsetkern `\LuaCJKsetkern {⟨before⟩} {⟨after⟩} {⟨length⟩}`

设置两个相邻标点之间的空白，覆盖样式的设置。这个长度是两处墨迹之间的空白，而不是从进距算起的 kern，所以 0pt 让两处墨迹相接，而不会把间距撑大；空白从字形的轮廓读出，读不到时按 kern 使用这个长度。每个参数只使用第一个字符。与 \LuaCJKsetwidth 一样，这个设置是全局的。与 xeCJK 不同，它也可以写在正文里：空白是在生成行时读取的，所以调用之后的行按新的空白排版。

```
\LuaCJKsetwidth{「}{1.5em}字「字\\
\LuaCJKsetkern{「}{」}{.3em}字「」字
```

字 「字
字 「」字

\LuaCJKDeclarePunctStyle `\LuaCJKDeclarePunctStyle {⟨name⟩} {⟨key-value list⟩}`

定义一种标点样式。

compress-adjacent = ⟨**true**|*false*⟩: 是否压缩相邻的标点。

adjacent-ratio = ⟨*number*⟩: 压缩的比例，相对于被压缩标点的宽度。压缩最多取走这个标点右侧的那份空白，所以它不会切进墨迹。

keep-open-close = ⟨**true**|*false*⟩: 开标点与闭标点相邻时是否保持不压缩。

```
\LuaCJKDeclarePunctStyle{mine}
  {adjacent-ratio = .25}
字，，字\par
\LuaCJKsetup{PunctStyle = mine}字，，字\par
\LuaCJKEditPunctStyle{mine}
  {adjacent-ratio = .75}
字，，字
```

字，，字
字，，字
字，，字

4.3 断行

一串破折号或省略号（U+2013、U+2014、U+2015、U+FF0D、U+2026 与 U+22EF）是一个整体：这些标点之间不加胶，所以两端对齐不会把一个两字宽的破折号拆开，它们之间也不提供断行机会。串中两个标点之间的源码空格同样不加胶，所以源文件里的换行不会把这一串标点拉开。这一串标点之前也不提供断行机会，所以一行不会以破折号或省略号开头。键 LongPunct 替换这个集合。

两个字符之间显式的 \nobreak 仍是断行禁止，它们之间的胶也照常插入：胶紧跟在惩罚之后，它提供的断行机会因而仍被禁止。若惩罚之前有一个源文件里的空格，这个空格也换成字符之间的胶，惩罚随之移到胶的前面。

基线被移动的盒子依附于它前面的字符：\textsuperscript 与 \textsubscript 就是这样排的。这种盒子两侧都不加胶，它前面也不提供断行机会，所以它的内容不会另起一行。有边框的盒子不是这种标记：\fbox、\framebox 与 \fcolorbox 把竖排的框围着内容放低，所以它像其他盒子一样按盒内文字加胶，两侧也为胶提供断行机会。

CheckSingle = ⟨*true*|**false**⟩ 打开时，末字符是 CJK 的段落的最后两个断行点被设为不可断开，这样单个字符就不会孤零零地落在最后一行上。初始值是 ⟨*false*⟩。段落以盒子结尾时，盒内末尾的字符就是段落的末字符，所以 \underline{乙} 排在段末也受到保护；盒子里的内容已经不能断开，于是若干汉字按它们所在的那一个整体计入，盒中已有三个汉字时不再插入惩罚。

WidowPenalty = ⟨*penalty*⟩: CheckSingle 使用的惩罚值。初始值是 10000；较小的值在值得这个代价时允许断开。

CJLineBreak = ⟨*normal*|*strict*⟩: 在 strict 模式下，Unicode 的 Line_Break 属性为 CJ 的字符（小假名之类）不能出现在行首。这个模式只增加禁则，不取消任何禁则。禁则属于不能起行的那个字符，起作用的是这个字符所在分组的设置。初始值是 normal，它把这些字符当作其他 CJK 字符。这个集合从 unicode-data 宏包集的 LineBreak.txt 读取，所以严格模式需要这个宏包集。

```
\parbox{3em}{天地玄黄宇宙洪荒\par}\quad
\LuaCJKsetup{CheckSingle = true}%
\parbox{3em}{天地玄黄宇宙洪荒\par}
```

天地玄　天地玄
黄宇宙　黄　宇
洪荒　　宙洪荒

\LuaCJKnobreak　\LuaCJKnobreak {⟨*text*⟩}

把 ⟨*text*⟩ 保持在一行之内。参数的节点被打上标记，相邻两个字符之间的断开被禁止，参数里由 \quad 或 \hspace 写出的胶也一样；参数与前后文之间的边界属于那段文字，照旧可以断开，两个相邻的参数是两个整体，它们之间也照旧可以断开。标记随当前分组结束。

```
\parbox{3em}{天地玄黄}\quad
\parbox{3em}{\LuaCJKnobreak{天地玄黄}}
```

天地玄黄　天地玄黄

4.4 暂停与钩子

\LuaCJKdisable
\LuaCJKenable

\LuaCJKdisable 停止排版调整：不再插入 CJKglue 或 CJKecglue，标点不再压缩，不再添加断行禁则，标点不再悬挂，数学模式里的 CJK 字符也不再处理。CJK 字符仍然用 CJK 字体排出，所以文字依然可读。这个开关只作用于当前分组；\LuaCJKenable 重新打开这些调整。

这就是抄录类环境使用的钩子，在测量原始节点列表时也用得上。

```
精致甲版 红楼梦\\
\LuaCJKdisable 精致甲版 红楼梦
```

精致甲版红楼梦
精致甲版 红楼梦

\LuaCJKShipoutHook

\LuaCJKShipoutHook {⟨*code*⟩}

在每一页输出之前运行 ⟨*code*⟩。代码被加入 shipout/before 钩子，所以它遵守通常的钩子顺序。

4.5 抄录与 listings

\LuaCJKVerbAddon
\LuaCJKOffVerbAddon

让一个 CJK 字符和当前等宽字体的两个空格一样宽，这样代码列表里含有 CJK 文字时各列仍然对齐。空隙以 CJKglue 的形式加入，所以 CJK 字体本身并没有被缩放；负的空隙只给出警告，什么也不加。 CJK 字符之间的断行被禁止，直到分组结束，与拉丁文的抄录文字一样。

这条命令在最外层的分组里不起作用，所以它应当在需要它的环境内部使用，例如 fancyvrb 的 formatcom 选项里。它在自己的分组里重新打开排版调整：要补的间距只有经过调整才会加进去，所以在 Verb 为 true 或 env 的抄录环境里它同样有效。 \LuaCJKOffVerbAddon 在同一个分组里再次取消它，并恢复它打开调整之前暂停的状态。

```
\ttfamily abc汉字abc\\
\ttfamily\LuaCJKVerbAddon
abc汉字abc
```

abc 汉字 abc
abc汉字abc

`Verb=⟨true|false|env|env+⟩` `\verb` 与抄录类环境（verbatim、Verbatim、lstlisting 以及它们的近亲）里做什么。`true` 在两者中都暂停调整；`env` 在环境里保留调整，但在 `\verb` 里不保留，这样代码里含有 CJK 文字时各列仍然对齐；`env+` 在所有地方都保留调整；`false` 完全不碰抄录材料。既然 `env+` 与 `false` 都保留调整，它们在 LuaCJK 里行为相同。暂停只管材料自己的内部：材料里的字符不再被压缩、悬挂，也不再加字间胶，而材料与周围文字之间的边界照常加胶，按两侧字符的类别决定，与同样一串文字用文本字体排出时相同。所以紧挨着 CJK 字符的 `\verb` 与 `\texttt` 留出同样的中西文胶，紧挨着被暂停材料的标点仍按标点处理，断行机会也照旧。无论这个设置取什么值，`\LuaCJKVerbAddon` 都会在它所在的分组里把调整重新打开。fancyvrb 的行内命令 `\Verb` 与 `\UseVerb` 按 `\verb` 的规则处理；块命令 `\VerbatimInput` 与 `\UseVerbatim` 则与抄录环境一致。

listings 一次处理一个字符，并且在读到字符的那一刻就把字符表之外的字符排出来。CJK 字符在它那张表之外，于是会跑到周围拉丁文字的前面。配套的 LuaCJK-listings 修正了这一点：它把 CJK 字符加入 listings 的字符表，所以它们留在原处，把每个字符算作两列，并在列表内部关掉间距调整。一串 CJK 字符对 `emph` 与 `keywords` 选项来说是一个词，所以可以给一个 CJK 变量名着色；拉丁字母、数字和 CJK 标点都会结束这个词。开启 `breaklines` 时，一串 CJK 字符逐字排出，所以一行可以在两个汉字之间断开；这一串仍然是一个词，`emph` 与 `keywords` 依旧把整个词着色。使用 listings 时 LuaCJK 自动加载它，所以不需要额外的命令。

`\lstinline` 以改变了的类别码读取参数；补丁会破坏那个扫描器，所以行内代码保持不动，按通常的方式处理。

4.6 装饰

配套的 LuaCJKfntef 提供下划线、着重号与删除线。它建立在 lua-ul 之上，所以被装饰的文字可以包含任意命令、CJK 与拉丁文字，也仍然可以被断字并跨行断开。

\LuaCJKfntefsetup `\LuaCJKfntefsetup {⟨keyval list⟩}`

设置整个包共用的键。键 commute 取 true|false，初始 true：对易时每条装饰的墨迹各按自己的键放置，两条装饰的嵌套次序不改变结果；不对易时，一组嵌套装饰的墨迹排进它们各自位置合成的那一段里，次序按嵌套 —— 外层在上、里层在下，余下的空白在相邻两条之间均分。于是两种写法占的高度相同，只是先后不同，次序与 xeCJK 相同；外层键给出的 depth 比组里最浅的墨迹更深时，整组以最浅的那条为准。写在分组里时，它只作用于该分组中开始的装饰。波浪与斜删除线是画出来的图案，节点列表看不出墨迹在哪里，所以它们不参加排布，保持原位。装饰写在一个盒子里时也一样：盒子单独排版，包住它的装饰不在同一行里，盒里的墨迹不动；把盒子放在文字上而不是装饰上，两条装饰就都在同一行里。

\CJKunderline `\CJKunderline [⟨options⟩] {⟨text⟩}`

给 ⟨*text*⟩ 加下划线。初始几何是 depth = 0.2em 与 thickness = 0.4pt。开头的一个 * 或 - 可以接受，并且被忽略。

所有装饰命令共用下面这组键：与某个命令无关的键照样接受，只是不起作用。只给一个颜色名而不写键名，等价于 format = ⟨*color*⟩。键 top 与 bottom 只对 \CJKunderline 有效，并覆盖计算出来的几何，不论横线是否由 format 着色。画出的图案从不改变被装饰行的高度或深度，所以行距不会变大。下划线、双线、波浪、斜删除线与符号都按当前字号各建一个图案，所以字体变大时图案跟着变大。

thickness = ⟨*length*⟩: 横线的粗细，初始 0.4pt。

depth = ⟨*length*⟩: 从基线到横线顶部的距离，初始 0.2em；着重号的点是它在 em 盒底部之下的深度。

textformat = ⟨*code*⟩: 作用于文字的代码，例如 textformat = \bfseries。

format = ⟨*color name*⟩: 横线的颜色，例如 format = red；它只影响横线，不影响文字。

color = ⟨*color name*⟩: format 的另一个名字。

subtract = ⟨*length*⟩: 在横线两端各去掉的长度，初始关闭；true 表示半字宽，这样相邻的两个命令不会连在一起。

skip = ⟨*true*|**false**⟩: 把标点排除在横线之外。

hidden = ⟨*true*|**false**⟩: 隐藏文字而保留横线，横线取隐藏文字的宽度、高度与深度；文字照常断行，也不留在 PDF 里，所以读不到、选不中。

gap = ⟨*length*⟩: 双线两条横线之间的距离，初始 1.1pt。

height = ⟨*length*⟩: 删除线中心到基线的距离，初始 0.3em。

symbol = ⟨*code*⟩: 替代波浪或圆点的符号。

selectable = ⟨*true*|**false**⟩: 装饰符号是否保留为文字，初始 false；true 时 PDF 阅读器可以复制与选择它。

不给 format 时，横线使用它所装饰的文字的颜色，例如 \textcolor 设定的颜色。

sep 为了兼容而接受，但没有作用。

```
\CJKunderline[thickness = 1pt]{有涯}\\
\CJKunderline[format = red]{有涯}\\
\CJKunderline[textformat = \bfseries]
  {有涯}\\
\CJKunderline[hidden]{有涯}
```

有涯
有涯
有涯

```
今天是 \CJKunderline{\hspace*{3em}} 年\\
第 \CJKunderline{\hspace*{2em}} 题
```

今天是 _____ 年
第 ____ 题

```
\CJKunderline{有涯}\CJKunderline{有涯}\\
\CJKunderline[subtract]{吾生也有涯}\\
\CJKunderline[skip]{有涯，无涯}
```

有涯有涯
吾生也有涯
有涯，无涯

\CJKunderdblline \CJKunderdblline [⟨*options*⟩] {⟨*text*⟩}

在 ⟨*text*⟩ 下面画两条平行的横线，键见上面的键列表。两条横线都使用 thickness，上面那条从 depth 开始， gap 是两条横线之间的距离。开头的一个 * 或 - 可以接受，并且被忽略。

\CJKunderwave \CJKunderwave [⟨*options*⟩] {⟨*text*⟩}

在 ⟨*text*⟩ 下面画一条波浪线。波浪线用 l3draw 画出，各个周期首尾相接：一个周期宽 1em，上下各 .25em。一段整字的文字由整数个周期组成，波浪线从第一个字的边缘画到最后一个字的边缘；被对齐拉伸时，整段按最终宽度分成整数个周期，周期跟着整行的拉伸变长或变短。整个图案随字号缩放；与其他画出的图案一样，它不改变行的高度与深度。 symbol 用来把波浪线换成别的符号：每字宽一个符号，居中，符号顶部在基线之下半字宽处。

```
\CJKunderwave{吾生也有涯}\\
\CJKunderwave[symbol = \textbullet]
  {吾生也有涯}\\
\CJKunderdot{吾生也有涯}\\
\CJKunderdot[symbol = \textbullet]
  {吾生也有涯}
```

吾生也有涯
吾生也有涯
吾生也有涯
吾生也有涯

`\CJKxout` `\CJKxout [⟨options⟩] {⟨text⟩}`

用重复的斜线划掉 ⟨*text*⟩，像 ulem 的 `\xout` 那样。斜线从基线之上 `0.8em` 延伸到基线之下 `-0.15em`，所以它穿过一个 CJK 字形的整个字身。

`\LuaCJKemdescent` 当前字体 em 盒的深度，可以当作长度使用。CJK 的 em 盒是一个 em 的正方形，所以把高度为 `1em` 的对象放在这个深度上，就正好覆盖一个 CJK 字符所占的区域；字体没有 OpenType 度量时退回 `.12em`。

`\CJKunderanyline` `\CJKunderanyline {⟨depth⟩} {⟨object⟩} {⟨text⟩}`

把 ⟨*object*⟩ 放在 ⟨*text*⟩ 下面，并用 `\leaders` 重复。⟨*object*⟩ 的底部放在基线之下 ⟨*depth*⟩ 处，所以对于一个（本身没有深度的）`\rule` 来说，横线从基线之下 ⟨*depth*⟩ 处向上占据。要覆盖一个 CJK 字形，就用字体的 em 盒：⟨*depth*⟩ 取 `\LuaCJKemdescent`，对象高度取 `1em`，正如下例所做的那样。对象填满的是 em 盒，也就是 PDF 阅读器划出的选区：横向是字身宽度，纵向是 em 盒的高度，字体墨迹比 em 盒窄时，四周会各露出一条边距。类型会被缓存，所以相同的参数复用一个下划线类型。

`\CJKunderanysymbol` `\CJKunderanysymbol[⟨options⟩]{⟨depth⟩}{⟨symbol⟩}{⟨text⟩}`

用法相同，只是每字宽放一个 ⟨*symbol*⟩：符号在一个和一个 CJK 字符一样宽的格子里居中，所以一排符号与它们上方的字符对齐。⟨*depth*⟩ 是符号顶部到 *em* 盒底部的距离，所以 `0pt` 留下的间隙正与字符两侧的边距相同；字体没有 OpenType 度量时按 `.12em` 计算。为了兼容，接受选项 `sep`，但它没有作用，因为格子固定为一个字宽。

```
\CJKunderanyline{\LuaCJKemdescent}
  {\color{yellow}\rule{1pt}{1em}}{有涯}\\
\CJKunderanysymbol{0pt}{\tiny$\triangle$}{有涯}
```

有涯
有涯
△ △

\CJKunderdot `\CJKunderdot [⟨options⟩] {⟨text⟩}`

在 ⟨*text*⟩ 下面加着重号。每个 CJK 字符下面一个点，点在字符的一字宽方格内居中，顶部在 em 盒底部之下 `0.2em` 处；拉丁字母、数字和空格下面没有点，一段文字断行以后，每一行的字符各有自己的点，粘连伸长时点随字符移动。用 `symbol = ⟨symbol⟩` 时，一个符号取代着重号，放置方式与 `\CJKunderanysymbol` 的符号相同：每字宽一个，居中，符号顶部在 em 盒底部之下 `0.3em` 处。

```
\CJKunderline{虚室生白，\CJKunderdot{吉祥}止止，
  虚室生白}
```

虚室生白，吉祥止止，
虚室生白

\CJKsout `\CJKsout [⟨options⟩] {⟨text⟩}`

给 ⟨*text*⟩ 加删除线，键见上面的键列表。删除线中心在基线上方，穿过一个 CJK 字形的中部，比拉丁字母 x 高度的中部要高；对于完全是拉丁文的文字，可以用 `height = 0.5ex`。

```
\CJKsout[thickness = 2pt, format = red]
  {吾生也有涯}\\
\CJKsout[height = .5ex]{吾生也有涯}
```

~~吾生也有涯~~
~~吾生也有涯~~

```
\CJKunderline   {吾生也有涯}\\
\CJKunderdblline{吾生也有涯}\\
\CJKunderwave   {吾生也有涯}\\
\CJKunderdot    {吾生也有涯}\\
\CJKsout        {吾生也有涯}\\
\CJKxout        {吾生也有涯}
```

吾生也有涯
吾生也有涯
吾生也有涯
吾生也有涯
~~吾生也有涯~~
~~吾生也有涯~~

5 与 xeCJK 的比较

LuaCJK 只要两个引擎允许，就沿用 xeCJK 的接口，所以一个已经在用 xeCJK 的文档大多不需要改动。这里只列出不同之处。

5.1 行为差异

- `Verb`：`env+` 的行为与 `false` 一样，因为两者都保留调整。
- 标点几何。样式把标点的尺寸给成一半，或者一半但句末标点保持全角，或者 0.7 倍，而进距与墨迹的位置取自标点自己的轮廓：样式只取走外侧的那份空白，所以墨迹几乎占满进距的标点保持全角。两个相邻标点之间的空白，以及 `\LuaCJKsetkern` 给出的长度，都是两处墨迹之间的空白。

- `\LuaCJKDeclarePunctStyle` 接受三个键 `compress-adjacent`、`adjacent-ratio` 与 `keep-open-close`；xeCJK 的三十个几何键描述它的侧边距模型，在这里没有对应物。
- `CJKfilltwosides*` 不与自然宽度作比较。
- CJK 字体在文字定形之前就应用到字符上，所以 CJK 字体的 OpenType 特性生效：组合记号附到它的基字上，变体序列选到它的字形，字体自己的连字（例如两个破折号并成一个两字宽的破折号）与 `locl` 形式也会出现；间距、标点进距与禁则仍然来自 LuaCJK 自己。
- 在 LuaCJKfntef 里，`format` 接受颜色名而不是代码，装饰由 lua-ul 画出、由本包的键配置，所以 `ulem` 的选项列表不存在。

5.2 未提供的功能

- `NewLineCS`、`EnvCS`、`InlineEnv` 与 `PlainEquation`：它们的存在是为了让 xeCJK 在扫描记号时认出 TEX 构造，而一个已经完成的节点列表不需要提示。
- `PoZheHaoLigature`：字体自带的两字宽破折号连字在定形时就会出现，所以没有开关它的选项。
- `RubberPunctSkip`：标点的进距是一个 kern，它既不是断行机会，也不参与两端对齐，而正是这一点让固定宽度的样式保持稳定。
- `CheckFullRight`：它需要标点之后的控制序列，节点列表并不显示这个；请改用 `\LuaCJKnobreak`。

1 Introduction

LuaCJK is a LuaTEX-only package for typesetting Chinese, Japanese, and Korean (CJK). Its main interface is `\setCJKmainfont`, `\setCJKsansfont`, `\setCJKmonofont`, `\setCJKfamilyfont`, `\CJKfamily`, and `\LuaCJKsetup`.

The current CJK font is recorded in a LuaTEX attribute, and each CJK glyph is set in that font before the text is shaped.

中文 日本語 한국어	中文日本語한국어

2 Loading

`\usepackage[`⟨*options*⟩`]{LuaCJK}`

LuaCJK requires the LuaTEX engine. It conflicts with luatexja: loading either package before or after the other raises an error.

`\LuaCJKversion` Print the version of LuaCJK.

`\LuaCJKsetup` `\LuaCJKsetup {`⟨*key-value list*⟩`}`

Configure LuaCJK. The keys are the ones documented in the following sections.

The keys fall into two groups.

The settings that describe how text is set are local to the current group, and they are recorded on the characters themselves, so text inside a group keeps its settings even when the group has already ended by the time the paragraph is broken:

```
\vbox{\begingroup\LuaCJKsetup{CJKglue = 10pt}中中\endgroup\par}
```

uses the 10pt glue, although the group is closed before `\par`. This group contains

- `CJKglue`, `CJKecglue`, `CJKspace`, and `xCJKecglue`;
- `PunctStyle`, `PunctKinsoku`, `AllowBreakBetweenPuncts`, `PunctWidth`, and `PunctBoundWidth`;
- `LatinPunct` and `CJKmath`;
- `CheckSingle`, `WidowPenalty`, and `CJLineBreak`;
- `AutoFallBack` and `Verb`;

- the two commands `\LuaCJKCancelSubCJKBlock` and `\LuaCJKRestoreSubCJKBlock`.

The other keys are global or act while the package is loaded:

- `KaiMingPunct`, `MiddlePunct`, `LongPunct`, and `NoBreakLongPunct` edit character sets;
- `AutoFakeBold`, `AutoFakeSlant`, `EmboldenFactor`, and `SlantFactor` act when a family is registered;
- `LocalConfig` acts while the package is loaded, and `LoadLXGW` at `\begin{document}`.

The declarations `\LuaCJKDeclareCharClass`, `\LuaCJKDeclarePunctClass`, `\LuaCJKDeclareSubCJKBlock`, `\LuaCJKsetwidth`, and `\LuaCJKsetkern` are global as well, as noted with each of them.

The same keys may be given as package options:

```
\usepackage[PunctStyle = banjiao]{LuaCJK}
```

`LocalConfig` = ⟨**true**|*false*|*name*⟩ Read local settings from a configuration file. `true` loads `luacjk.cfg`, ⟨*name*⟩ loads `luacjk-`⟨*name*⟩`.cfg`, and `false` (the initial value) loads nothing. The file is read at the end of the package, so it may use every command LuaCJK defines. A missing file only warns.

To use LuaCJK with ctex, pass the `engine` option with the value ⟨*luacjk*⟩:

```
\documentclass[engine = luacjk]{ctexart}
```

ctex then inputs LuaCJK's `ctex-engine-luacjk.def` instead of its own LuaTEX backend, and that file loads the package, so LuaCJK is not loaded separately.

LuaCJK cooperates with the LATEX tagged-PDF project. A document that declares

```
\DocumentMetadata{tagging = on}
```

marks the ink of a decoration as `artifact`, and a decoration symbol is left out of the text a reader copies or selects; `selectable = true` keeps the symbol as text. Under the l3doc class, load latex-lab-testphase-l3doc as well; this manual is built that way. A PDF reader derives the height of a selection from the font's `hhea` ascent and descent, which is 1.437em for `Noto Serif CJK SC`; the selection is therefore taller than the 1em square. LuaCJK cannot change that, and it does not affect the glyph.

3 Fonts

The main font commands take an optional list of fontspec options. Options may be written before or after the font name. A CJK font is always selected in the Unicode encoding, whatever encoding the Latin text uses, so fontenc's `T1` option leaves the CJK text alone.

`\setCJKmainfont` `\setCJKsansfont` `\setCJKmonofont`

`\setCJKmainfont [⟨options⟩] {⟨font⟩} [⟨options⟩]`

Set the CJK font for the roman, sans serif, and monospaced families.

```
\setCJKmainfont[Scale = 1.2]
  {LXGWNeoZhiSong.ttf}
English 中文字体\\
\setCJKsansfont[Scale = 1.2]
  {LXGWWenKaiGBLite-Regular.ttf}
\sffamily English 中文字体
```

English 中文字体
English 中文字体

`\setCJKfamilyfont` `\setCJKfamilyfont{⟨family⟩}[⟨options⟩]{⟨font⟩}[⟨options⟩]`

Define a named CJK family. The family may be given as `\CJKrmdefault` and friends.

`\CJKfamily` `\CJKfamily {⟨family⟩}`

Switch to a named CJK family. The family may be given as `\CJKrmdefault` and friends.

```
\CJKfamily{zhsong}English 中文字体\\
\CJKfamily{zhhei}English 中文字体\\
\CJKfamily{zhkai}English 中文字体\\
\CJKfamily{zhfs}English 中文字体
```

English 中文字体
English 中文字体
English 中文字体
English 中文字体

`\newCJKfontfamily` `\newCJKfontfamily[⟨family⟩]⟨command⟩ [⟨options⟩]{⟨font⟩}[⟨options⟩]`

Define a command that switches to a newly declared family. The command is local to the current group, like `\newfontfamily`. The options may be given before or after the font name.

\CJKfontspec `\CJKfontspec [⟨options⟩] {⟨font⟩}`

Select a CJK font for the current group. The options go before the font name, as for `\fontspec`: the command is used inside the text, so if it also read a bracket after the font name, the bracket that the text begins with would be taken as an option list and would vanish.

`\songti`
`\heiti`
`\fangsong`
`\kaishu`
`\lishu`
`\youyuan`

Shortcuts for the families named `zhsong`, `zhhei`, `zhfs`, `zhkai`, `zhli`, and `zhyou`.

```
\newCJKfontfamily[kai]\mykai
  {LXGWWenKaiGBLite-Regular.ttf}
  \mykai English 中文字体\\
  \songti English 中文字体
```

English 中文字体
English 中文字体

`\CJKrmdefault`
`\CJKsfdefault`
`\CJKttdefault`
`\CJKfamilydefault`

The CJK families selected by `\rmfamily`, `\sffamily`, and `\ttfamily`, and the family selected by `\normalfont`. `\CJKfamilydefault` follows `\familydefault` by default: it is the CJK family that corresponds to the main family when that is one of the three, and `\CJKrmdefault` otherwise. It is selected when the document starts, so redefining it or `\familydefault` in the preamble sets the main text in that family from the first line.

3.1 Sub-blocks

One CJK font rarely covers every block of Unicode. A sub-block gives a range of characters its own font inside a family; the block then becomes an option of the font commands.

`\LuaCJKDeclareSubCJKBlock` `\LuaCJKDeclareSubCJKBlock {⟨block⟩} {⟨range list⟩}`

Declare ⟨*block*⟩ with the given ranges, in the same spelling as `\LuaCJKDeclareCharClass`. The name becomes a font option, as the example below shows: a family that sets it uses the second font for the characters of the block, and a family that does not leaves them in the main font. The declared range joins the CJK character class as well, so a character outside the built-in CJK ranges (the circled numbers `U+2460` to `U+2473`, say) is typeset as CJK once a block covers it. Overlapping blocks are resolved in declaration order.

```
\LuaCJKDeclareSubCJKBlock{Kana}{3040 -> 30FF}
\setCJKmainfont[Kana = LXGWWenKaiGBLite-
Regular.ttf]
  {LXGWNeoZhiSong.ttf}
汉字 かんじ
```

汉字かんじ

\LuaCJKCancelSubCJKBlock `\LuaCJKCancelSubCJKBlock {⟨block list⟩}`
\LuaCJKRestoreSubCJKBlock `\LuaCJKRestoreSubCJKBlock {⟨block list⟩}`

Turn the listed blocks off or on again for the rest of the group. Because the choice of font is recorded per font identifier, two families that supply the same font share one setting: the family that selected the font last decides.

3.2 Fallback fonts

A CJK font rarely covers every character one needs. Fallback fonts are tried in the order they were declared when the selected CJK font has no glyph for a character.

\setCJKfallbackfamilyfont `\setCJKfallbackfamilyfont {⟨family⟩} [⟨options⟩] {⟨font⟩} [⟨options⟩]`

Add ⟨*font*⟩ to the fallback list of ⟨*family*⟩. The family may be given as `\CJKrmdefault` and friends. Call the command several times to build a chain; the first font that has the character wins. The chain is resolved in the weight, shape, and size of the text, so a character in bold takes the bold face of the fallback font. With `CJKmath` on, a CJK character in math mode falls back through the same chain: the list of the CJK family that supplies the math font is tried, at the size of the math style.

`AutoFallBack` = ⟨*true*|**false**⟩ Whether to use the declared fallback fonts. The switch follows the text like the spacing keys, so it can be turned on for one group.

```
\setCJKmainfont{LXGWNeoZhiSong.ttf}
\setCJKfallbackfamilyfont{\CJKrmdefault}
  {LXGWWenKaiGBLite-Regular.ttf}
\LuaCJKsetup{AutoFallBack = true}中あ
```

中あ

3.3 Baseline shift

Two CJK fonts are rarely designed for each other: mixed in one line, the characters of one font sit too high or too low. A font may carry a `Baseline` value that moves its characters up or down, so that the two fonts stand on one baseline. The value belongs to the font rather than to the text, which is why it is an option of the command that declares the font.

`Baseline` = ⟨*dimension*⟩ Raise the characters of this font by ⟨*dimension*⟩; a negative value lowers them. It is an option like the others, so a fallback declaration may carry it as well, and that is its usual use: it makes a fallback font stand on the baseline of the main font. Any TEX unit works, and `em` follows the size. The offset is recorded on the characters the family sets, so two declarations that supply the same font keep their own values in one paragraph, and a family selected inside a group moves only that group.

```
\setCJKfamilyfont{high}[Baseline = .3em]
  {LXGWWenKaiGBLite-Regular.ttf}
\songti 基线\quad\CJKfamily{high}基线
```

基线　基线

3.4 Loading LXGW

`LoadLXGW` = ⟨**true**|*false*⟩ On by default. When the document declares no CJK font, load the LXGW faces at `\begin{document}`: LXGW Neo ZhiSong as the main font, LXGW Neo XiHei as the sans serif font, Zhuque Fangsong as the monospaced font, and Neo ZhiSong, Neo XiHei, WenKai, and Zhuque Fangsong as the named families. The Neo ZhiSong family takes WenKai as its italic face and ZhenKai Slab as its bold italic, so `\emph` and `\itshape` set the text in Kai. Neo ZhiSong, Neo XiHei, and Zhuque Fangsong have no bold face of their own, so their declarations ask for fake bold, Neo XiHei and Zhuque Fangsong ask for fake slant as well, and the WenKai family takes ZhenKai Slab as its bold face. A document that declares fonts keeps its own, so LXGW does not mix into it; when it does not declare the family `\CJKfamilydefault` names, its text takes the family `\CJKrmdefault` names, or the first declared family when that one is missing too, and a warning says so. A document that declares nothing with the key off is warned about as well.

3.5 Fake bold and slant

A CJK family often has no bold or italic face. The following keys make LuaCJK ask fontspec to embolden or slant the regular face instead. They apply

to families registered afterwards, and are not used when the family itself sets `BoldFont` or `ItalicFont`. They are global: a group does not change what an already registered family uses. A family that names neither `ItalicFont` nor `AutoFakeSlant` has no italic shape: emphasized text keeps its upright face, and fontspec reports the shape as missing.

`AutoFakeBold` = ⟨*true*|**false**|*number*⟩ Embolden the regular face when the bold shape is requested. A number also sets `EmboldenFactor`.

`AutoFakeSlant` = ⟨*true*|**false**|*number*⟩ Slant the regular face when the italic shape is requested. A number also sets `SlantFactor`.

`EmboldenFactor` = ⟨*number*⟩: The amount of emboldening. The initial value is `4`.

`SlantFactor` = ⟨*number*⟩: The amount of slanting, between −0.999 and 0.999. The initial value is `0.167`.

```
\LuaCJKsetup{AutoFakeBold = true}
\setCJKmainfont{LXGWNeoZhiSong.ttf}
\textbf{English 中文字体}\\
English 中文字体
```

English 中文字体
English 中文字体

`\defaultCJKfontfeatures` `\defaultCJKfontfeatures {⟨font features⟩}`

Set the features prepended to every CJK family registered afterwards, and to the font of each of its sub-blocks and fallbacks. The initial value is `Script = CJK`. Families registered before the call keep the features they were given.

`\addCJKfontfeatures` `\addCJKfontfeatures [⟨ignored⟩] {⟨font features⟩}`

Add features to the current CJK family for the rest of the group, for example `\addCJKfontfeatures{Scale = 1.2}`. A leading `*` is accepted; both forms act on the same font.

```
\defaultCJKfontfeatures{Scale = 1.5}
\setCJKfamilyfont{big}
  {LXGWNeoZhiSong.ttf}
\CJKfamily{big}中文字体\\
\CJKfamily{zhsong}中文字体\\
\addCJKfontfeatures{Scale = 1.2}
\CJKfamily{zhsong}中文字体
```

中文字体
中文字体
中文字体

3.6 CJK in mathematics

`CJKmath` = ⟨*true*|**false**⟩: Whether CJK characters may be typed directly in math mode. When it is on, such a character is set in the CJK font of a math family of its own instead of being looked up in the math font, so a character in a superscript or a fraction takes the size of that math style. Enable it with `\LuaCJKsetup{CJKmath = true}` or with the package option `CJKmath = true`. Such a character takes the font of the math version, as `\mathversion` sets it: the bold face of the family in the `bold` version, its regular face when it has no bold one, and the upright shape of the family rather than the shape of the text. Without it a CJK character in math mode is looked up in the math font and usually disappears, so LuaCJK issues a one-time warning in that case.

`\setCJKmathfont` `\setCJKmathfont [⟨options⟩] {⟨font⟩} [⟨options⟩]`

Select the CJK font used for characters in math mode. Without this command the CJK main font is used. This manual enables `CJKmath` and sets this font in its preamble. The selection is local to the current group, like the other font commands: a formula in a group uses the family selected in that group.

Each CJK family that appears in math is registered as a math family of its own, so that TeX can pick the text, script, or scriptscript size of it. The family takes its font from the math version: `\bfdefault` in the `bold` version and `\mddefault` in the others, with the shape `\shapedefault`. `\boldmath` therefore bolds the CJK characters of a formula, within the group that asks for it, and a family without a bold face keeps its regular one without a warning. The family number comes from the range 240–255, the last sixteen of the 256 that LaTeX 2_ε allocates under LuaTeX, which an ordinary document never reaches; every fallback font used in math takes a number of its own, shared by the families that declare it, and a document that needs more than sixteen numbers stops with an error. The script sizes are seven and five tenths of the current text size, the ratio of the standard size tables. That text size is the one of the text around the formula, not the size a `\text` inside it switches to, so a formula that contains a `\text` is as large as one that does not. The math parameters are not taken from the CJK font, which has no `MathConstants` table: they stay those of the document's math font, which is why a fraction of two CJK characters has the rule and the gaps of the surrounding math.

`$ 天地玄黄 $ and 天地玄黄`	$天地玄黄$ and 天地玄黄

4 Text layout

4.1 Spacing

CJKglue = ⟨*glue*⟩: Glue inserted between adjacent CJK characters. The initial value is `0pt plus 0.08\baselineskip`, recomputed at every font selection, so it follows the size; a value given by the document is used as written and does not change with the size, and its stretch order (for example `plus 1filll`) is kept as well.

CJKecglue = ⟨*glue*⟩: Glue inserted between CJK and Latin characters. Beside a punctuation mark the glue has no advance: the mark's own advance already carries the spacing. A half-width ASCII bracket keeps that glue only in front of an opening one; the inside of the pair takes none, so the brackets hug the text they enclose. A formula that sets one opening or closing delimiter alone is the same: the delimiter hugs the text it encloses, and the glue stands on its outside. The initial value is `0.3333em plus 0.1667em minus 0.1111em`, which follows the size of the CJK character it stands beside rather than the size of the Latin text: a run set with `\scriptsize` or `\large` keeps the spacing of the surrounding CJK text on both of its sides.

CJKspace = ⟨*true*|**false**⟩: Whether to keep source spaces between CJK characters. When they are not kept, the space becomes the inter-character glue, so the break opportunity and the stretch survive. A space written with `\ ` or `\nobreakspace` is not a source space, and its advance is that of a space.

xCJKecglue = ⟨*true*|**false**⟩: Whether to replace source spaces between CJK and Latin characters with `CJKecglue`.

`中文 English 中文`	中文 English 中文
`\LuaCJKsetup{CJKspace = true}天 地 玄 黄 宇 宙`	天 地 玄 黄 宇 宙

`\normalspacedchars` `\normalspacedchars {⟨character list⟩}`

The listed characters get no glue on either side, so that a hyphen or a slash inside Latin text does not pull the surrounding text apart. The initial list is / and -. The setting is global. A backslash is not in the list: beside a CJK character it usually begins the name of a printed command, which takes the glue as any other Latin character does, not the none that a separator in a path takes.

`中x中\par` `\normalspacedchars{x}中x中`	中 x 中 中x中

Characters that only control the text and have no advance of their own come in two kinds. U+200B, U+2060, U+180E, and U+FEFF carry nothing a font can use, so they are removed at the input: they neither take glue nor separate two CJK characters. The joiners (U+200C, U+200D) and the variation selectors are left to the font: each follows the font of the character in front of it, so an emoji sequence is not split into two pictures and a variation sequence selects its glyph; what the font leaves of them is removed before the line is broken, so they take no advance and add no `CJKecglue` between two CJK characters. A character requested with `\char` is not removed, because `\char` asks for the glyph itself. A box is read by the advance it occupies in the same way: one that occupies none does not separate the text around it, so what `\rlap`, `\llap` and `\clap` set over the text stays in one run. The source spaces on either side of such a box belong to that one boundary as well: the space next to the character behind the box becomes its glue and the space in front of the box gives up its advance, so two spaces around it are one space, as they are when no box stands between them. A box otherwise takes the class of the text nearest its edge: a kern, a spacing glue, or a rule inside it does not hide that text, so the glue is added by the class of the text and not by the skip the document wrote inside the box. Only a filling glue, such as the `\hfil` of `\numberline`, leaves the edge without text, which keeps the titles of a table of contents aligned. A vertical box is read at its edge by the same rule, so the rule under the text of an `\underline` does not hide that text and the underlined run takes the glue on both sides. A box that the edge reaches is read in turn, and one that has no class of its own does not hide the text behind it: the frame xcolor draws over a box as a second, classless box does not hide the content it encloses. The background a colour box draws is part of that box: the glue stands in front of the background, so the highlight covers the box and its padding and both inter-character glues stay outside it. A paragraph line begins with the marker TEX writes for its paragraph settings, which is passed over in the same way, so the first character of a `\parbox` reaches the left edge, while `\parfillskip` ends the line and leaves the right edge without text. A box that holds no text and has ink is Latin material, so a rule box takes the glue on both sides.

`CJKfilltwosides` (*env.*) Set the lines in a `minipage` whose lines are justified to ⟨*width*⟩, separated by `\\`. Its ⟨*position*⟩ argument places the box as `\parbox` does: `t` puts the baseline of the first line on the surrounding baseline, `c` centres the box on the text axis (the default), and `b` puts the last line there. The environment leaves vertical mode first, as `\parbox` does, so the text after it joins the box on the same line instead of beginning a paragraph, and a space after the environment is ignored as xeCJK ignores it. The settings of the body are those of a `\parbox` as well: the paragraph indent and the paragraph skip are zero, so every paragraph after a paragraph

break starts at the left edge of the box and is filled like the first, and the leading is the document's own rather than whatever an enclosing alignment left behind. A `\footnote` in the body is set at the bottom of the box, below a rule, and numbered with the box's own counter, as it is in a `minipage`. A width of zero or less keeps the surrounding line width. The starred form behaves the same; it does not compare against the natural width of the content.

```
\begin{CJKfilltwosides}[⟨position⟩]{⟨width⟩}
⟨lines⟩
\end{CJKfilltwosides}
```

```
\begin{CJKfilltwosides}{6em}
  瞻彼闋者，\\虛室生白
\end{CJKfilltwosides}
```

瞻彼闋者，
虛室生白

```
\begin{CJKfilltwosides*}{0pt}
  瞻彼闋者，\\虛室生白
\end{CJKfilltwosides*}
```

瞻 彼 闋 者 ，
虛 室 生 白

4.2 Punctuation

PunctStyle = ⟨*style*⟩: Select a punctuation style. The implemented styles are `quanjiao` (the default), `plain`, `banjiao`, `kaiming`, `CCT`, and `hangmobanjiao`. An unknown name issues a warning and falls back to `plain`. A style with a fixed width never cuts into the ink: a mark keeps its ink and the smaller of its two blanks, so a mark whose ink nearly fills its advance, U+FF05 for instance, keeps the full width.

PunctWidth = ⟨*length*|*false*⟩: Give every punctuation mark the same advance, whatever the style says. `0pt` is an advance too, so the marks take none at all; `false` (the initial value) returns them to the width the style gives. A width set with `\LuaCJKsetwidth` still wins for that mark.

PunctBoundWidth = ⟨*length*|*false*⟩: How much of a mark's advance stays inside the line when the mark ends or starts the line; the rest of the ink stands in the margin, or in the indentation at the start of a paragraph or of a list item. The default leaves the mark's own blank out, so its ink stands on the measure, and line breaking counts that hang, so a mark can end the line the natural width would refuse. A centered or a flush-right paragraph does not begin its lines at the edge of the measure, so its start and a line after `\\` do not hang there, where the rule would move the whole line; the end of a line still hangs, and so does the last line of the paragraph,

which puts the ink of the mark on the alignment edge and makes every line of it stop at the same place. A mark inside a box hangs only when nothing inked follows it in that box: the frame of `\fbox` is the edge the line ends at, so the mark it holds does not hang, while the `\mbox` that ends at the mark hangs its blank as plain text does. `0pt` hangs the whole advance; `false` returns to that default.

`LatinPunct` = ⟨*true*|**false**⟩: Treat the quotation marks (U+2018, U+2019, U+201C, U+201D), the middle dot (U+00B7), the two dot leader (U+2025), the ellipses (U+2026, U+22EF), and the hyphenation point (U+2027) as Latin punctuation: they stay in the current Latin font and are not compressed. This keeps an apostrophe inside an English word from turning into a full-width form. The em dash is not affected.

`PunctFamily` = ⟨*family*|*false*⟩: Use a different CJK family for the punctuation marks. The family must be declared with `\setCJKfamilyfont`, and may be given as `\CJKsfdefault` and friends; `false` (the initial value) makes the marks follow the text again. An undeclared family leaves the marks in the text font and warns once. The `Baseline` of that family moves the marks it sets as well.

`AllowBreakBetweenPuncts` = ⟨*true*|**false**⟩: Whether two adjacent punctuation marks may be separated by a line break. A dash or an ellipsis is not affected: a run of them is governed by `NoBreakLongPunct`. The prohibitions that keep a closing, middle, dash, or ellipsis mark out of the line start, and an opening mark out of the line end, are not lifted, so a closing quotation mark is never set at the start of a line. The marks are still compressed.

`PunctKinsoku` = ⟨**true**|*false*⟩: Whether to apply line-breaking prohibitions for punctuation. A closing, middle, dash, or ellipsis punctuation cannot start a line, an opening punctuation cannot end a line, and the same holds for a half-width ASCII bracket: an opening one cannot end a line and a closing one cannot start one; two adjacent marks are not separated, and a run of long punctuation is kept together. A word space between two marks separates them, so the break there stays, and `Try again. (Why?)` may break after the full stop. A mark at the edge of a box takes part as well. The prohibitions are independent of `PunctStyle`, so they stay active with `PunctStyle = plain`.

```
不亦说乎，不亦乐乎\\
\LuaCJKsetup{PunctFamily = zhkai}
不亦说乎，不亦乐乎
```

不亦说乎，不亦乐乎
不亦说乎，不亦乐乎

```
\LuaCJKsetup{LatinPunct = false}%
a’a\quad 不亦说乎！\\
\LuaCJKsetup{LatinPunct, PunctWidth = .8em}
a’a\quad 不亦说乎！
```

a’a 不亦说乎！
a’a 不亦说乎！

```
\fbox{甲，乙}\\
\LuaCJKsetup{PunctWidth = 0pt}%
\fbox{甲，乙}
```

甲，乙
甲乙

```
列位看官。\\
\LuaCJKsetup{PunctBoundWidth = .5em}
列位看官。\\
（列位看官
```

列位看官。
列位看官。
（列位看官

The character sets that the punctuation code works with come next. Each of their keys has three forms: the plain one replaces the set, the form with + adds characters, and the form with - removes them. Ranges use the same spelling as `\LuaCJKDeclareCharClass`. The sets are global: a group does not bring the previous members back.

`KaiMingPunct<+->` = ⟨*char list*⟩: The marks that keep a full width in the `kaiming` style. Initially 。、．、？、！.

`MiddlePunct<+->` = ⟨*char list*⟩: The marks centered between the two halves of their box. A middle mark cannot start a line.

`LongPunct<+->` = ⟨*char list*⟩: The marks that are never compressed against each other, such as dashes and ellipses.

`NoBreakLongPunct<+->` = ⟨*char list*⟩: The long marks that must not be separated by a line break. Initially the same as `LongPunct`.

\LuaCJKDeclareCharClass `\LuaCJKDeclareCharClass {⟨class⟩} {⟨range list⟩}`

Override the class of a code-point range. Ranges are written in hexadecimal and the leading " may be omitted; `"START -> "END` gives a range, and the list is comma separated.

The classes are: `CJK` and `Default` make the range behave like CJK or Latin text; `FullLeft` and `HalfLeft` treat it as an opening mark, `FullRight` and `HalfRight` as a closing mark, so the line-breaking prohibitions apply; the first of each pair is a full-width mark and is compressed and hung as well, while the second is an ASCII bracket: it keeps its own advance, does not hang, hugs the text between the two of a pair, and takes the CJK/Latin glue on the outside of the pair, in front of an opening bracket and behind a closing one. `NormalSpace` takes the range out of the CJK/Latin glue, like `\normalspacedchars`; `CJStarter`, `CM`, and `Boundary` are treated as `CJK`, because LuaTEX has no model of combining marks or run boundaries.

\LuaCJKDeclarePunctClass `\LuaCJKDeclarePunctClass {⟨group⟩} {⟨range list⟩}`

Override the punctuation group of a range. The groups are `open`, `close`, `middle`, `dash`, `ellipsis`, `half-open` and `half-close` for the half-width brackets, and `latin-close` for the ASCII closing marks: that group takes the zero-width boundary glue beside CJK, and its advance belongs to the Latin font, so no compression style, no hang, and `PunctWidth` rewrites it. `none` removes the punctuation treatment, so the characters are no longer compressed or hung.

\LuaCJKResetCharClass Drop every class and punctuation override declared so far, and put the normal-spaced set back to the characters of `\normalspacedchars`.

```
中ABC中\par
\LuaCJKDeclareCharClass{CJK}{0041 -> 005A}
中ABC中\par
\LuaCJKDeclarePunctClass{none}{FF0C}%
字，，字\par
\LuaCJKResetCharClass 中ABC中
```

中 ABC 中
中ABC中
字，，字
中 ABC 中

\LuaCJKEditPunctStyle `\LuaCJKEditPunctStyle {⟨style⟩} {⟨key-value list⟩}`

Change the parameters of an existing style. The keys are `compress-adjacent`, `adjacent-ratio`, `keep-open-close`; a key that is not given keeps its value.

\LuaCJKsetwidth `\LuaCJKsetwidth {⟨punctuation list⟩} {⟨length⟩}`

Give the listed punctuation marks a fixed advance, overriding the selected `PunctStyle` and `PunctWidth`. The setting is global: a group does not restore the previous width, and a later call adds to the table rather than replacing it. A leading `*` is accepted; the width it would set at the start or end of a line is not implemented, and the star warns once.

\LuaCJKsetkern `\LuaCJKsetkern {⟨before⟩} {⟨after⟩} {⟨length⟩}`

Set the blank between two adjacent punctuation marks, overriding the style. The length is the blank between the two inks rather than a kern measured from the advance, so `0pt` makes the inks meet instead of widening the gap; the blank is read from the outline of the glyph, and the length is used as a kern when the file cannot be read. Only the first character of each argument is used. Like `\LuaCJKsetwidth`, the setting is global. Unlike xeCJK, it may also be used in the document body: the blank is read when a line is built, so the lines after the call take the new gap.

```
\LuaCJKsetwidth{「}{1.5em}字「字\\
\LuaCJKsetkern{「}{」}{.3em}字「」字
```

字 「字
字 「」字

\LuaCJKDeclarePunctStyle `\LuaCJKDeclarePunctStyle {⟨name⟩} {⟨key-value list⟩}`

Define a punctuation style.

`compress-adjacent` = ⟨**true**|*false*⟩: whether two adjacent punctuation marks are compressed.

`adjacent-ratio` = ⟨*number*⟩: the ratio of the compression, relative to the width of the mark that is compressed. A compression takes at most the blank on the right of that mark, so it never reaches into the ink.

`keep-open-close` = ⟨**true**|*false*⟩: whether an opening mark next to a closing one stays uncompressed.

```
\LuaCJKDeclarePunctStyle{mine}
  {adjacent-ratio = .25}
字，，字\par
\LuaCJKsetup{PunctStyle = mine}字，，字\par
\LuaCJKEditPunctStyle{mine}
  {adjacent-ratio = .75}
字，，字
```

字，，字
字，，字
字，，字

4.3 Line breaking

A run of dashes or ellipses (U+2013, U+2014, U+2015, U+FF0D, U+2026, and U+22EF) is one unit: the marks are joined without glue, so justification cannot pull a two-em dash apart, and no break is offered between them. A source space between two marks of a run adds no glue either, so a newline in the source cannot space the run apart. No break is offered in front of the run either, so a line does not begin with a dash or an ellipsis. The key `LongPunct` replaces the set.

An explicit `\nobreak` between two characters stays a prohibition, and the glue between them is still inserted: the glue follows the penalty, so the break it offers is still forbidden. A source space in front of the penalty becomes that glue as well, and the penalty moves in front of it.

A box whose baseline has been moved belongs to the character in front of it: `\textsuperscript` and `\textsubscript` set their content in one. No glue is inserted beside such a box, and it offers no break before it, so its content never starts a line of its own. A framed box is not such a mark: `\fbox`, `\framebox` and `\fcolorbox` lower a vertical frame around their content, so the box takes the glue of the text it holds on both sides, and offers the break that glue gives.

`CheckSingle` = ⟨*true*|**false**⟩ When on, the last two break points of a paragraph whose final character is CJK are made unbreakable, so a single character cannot end up alone on the last line. The initial value is ⟨*false*⟩. A box at the end of the paragraph does not hide its text: the character the box ends with is the final character, so `\underline{乙}` is protected too. What a box holds is already unbreakable, so its characters count as the one unit they travel as, and three of them insert no penalty.

`WidowPenalty` = ⟨*penalty*⟩: The penalty used by `CheckSingle`. The initial value is `10000`; a lower value allows the break when it is worth the cost.

`CJLineBreak` = ⟨*normal*|*strict*⟩: In the `strict` mode a character whose Unicode `Line_Break` property is `CJ` (the small kana and the like) may not start a line. The mode only adds prohibitions; it does not remove any. The prohibition belongs to the character that may not start the line, so the setting of the group that character is set in decides. The initial value is `normal`, which treats those characters like other CJK characters. The set is read from `LineBreak.txt` of the unicode-data bundle, so the package needs that bundle for the strict mode.

```
\parbox{3em}{天地玄黄宇宙洪荒\par}\quad
\LuaCJKsetup{CheckSingle = true}%
\parbox{3em}{天地玄黄宇宙洪荒\par}
```

天地玄　天地玄
黄宇宙　黄　宇
洪荒　　宙洪荒

\LuaCJKnobreak `\LuaCJKnobreak {⟨text⟩}`

Keep ⟨*text*⟩ on one line. The nodes of the argument are marked and the line-breaking pass forbids a break between two of them, including a glue the argument writes with `\quad` or `\hspace`; the boundary between the argument and the text around it belongs to that text, so it keeps the break the text offers, and two adjacent arguments are two units for the same reason. The marking ends with the current group.

```
\parbox{3em}{天地玄黄}\quad
\parbox{3em}{\LuaCJKnobreak{天地玄黄}}
```

天地玄黄 天地玄黄

4.4 Suspending and hooks

\LuaCJKdisable
\LuaCJKenable

`\LuaCJKdisable` stops the typographic adjustments: no `CJKglue` or `CJKecglue` is inserted, punctuation is not compressed, no line-breaking prohibitions are added, punctuation is not hung, and CJK characters in math mode are left alone. CJK characters are still typeset in the CJK font, so text stays readable. The switch is local to the current group; `\LuaCJKenable` turns the adjustments back on.

This is the hook verbatim-like environments use, and it is also useful when measuring raw node lists.

```
精致甲版 红楼梦\\
\LuaCJKdisable 精致甲版 红楼梦
```

精致甲版红楼梦
精致甲版 红楼梦

\LuaCJKShipoutHook `\LuaCJKShipoutHook {⟨code⟩}`

Run ⟨*code*⟩ before every page is shipped out. The code is added to the `shipout/before` hook, so it obeys the usual hook ordering.

4.5 Verbatim and listings

\LuaCJKVerbAddon
\LuaCJKOffVerbAddon

Make one CJK character as wide as two spaces of the current monospaced font, which keeps columns aligned when a code listing contains CJK text. The gap is added as `CJKglue`, so the CJK font itself is not scaled; a negative gap only raises a warning and adds nothing. Breaks between CJK characters are forbidden until the group ends, as in Latin verbatim text.

The command does nothing at the outermost group level, so it is meant to be used inside the environment that needs it, for example in the `formatcom` option of fancyvrb. It turns the typographic adjustments back on inside its own group, because the gap it computes reaches the text only through them, so it works in a verbatim-like environment whose `Verb` setting is `true` or `env` as well. `\LuaCJKOffVerbAddon` cancels it again inside the same group and restores the suspension that was in force before the addon turned it off.

```
\ttfamily abc汉字abc\\
\ttfamily\LuaCJKVerbAddon
abc汉字abc
```

```
abc 汉字 abc
abc汉字abc
```

`Verb` = ⟨**true** | *false* | *env* | *env+*⟩ What to do inside `\verb` and the verbatim-like environments (`verbatim`, `Verbatim`, `lstlisting`, and their relatives). `true` suspends the adjustments in both; `env` keeps them inside the environments but not in `\verb`, which preserves column alignment when the code contains CJK text; `env+` keeps them everywhere; `false` leaves verbatim material entirely alone. Since `env+` and `false` both leave the adjustments on, they behave the same in LuaCJK. Suspension leaves the material itself alone: its characters are not compressed, hung, or joined by inter-character glue, while the boundary between the material and the text around it takes the glue its two classes give, as the same characters in a text font would. A `\verb` beside a CJK character is therefore spaced as `\texttt` is, and a punctuation mark next to suspended material keeps the glue and the break opportunity it has beside ordinary text. Whatever the setting is, `\LuaCJKVerbAddon` turns the adjustments back on inside the group it is used in. The inline `\Verb` and `\UseVerb` of fancyvrb are treated as `\verb` is, and its block commands `\VerbatimInput` and `\UseVerbatim` as the environments are.

listings processes one character at a time and typesets a character that is outside its character table at the moment it is read. CJK characters are outside that table, so they end up in front of the surrounding Latin text. The companion package LuaCJK-listings fixes this: it adds the CJK characters to the character table of

listings, so they stay in place, counts each of them as two columns, and turns the spacing adjustments off inside the listing. A run of CJK characters is one word for the `emph` and `keywords` options, so a CJK variable name can be highlighted; a Latin character, a digit, or a CJK mark ends the word. With `breaklines` on, a run is set one character at a time, so a line breaks between two CJK characters; the run is still one word, so `emph` and `keywords` still highlight it as a whole. LuaCJK loads it automatically when listings is used, so no extra command is needed.

`\lstinline` reads its argument with changed category codes; the patch would break that scanner, so inline code is left alone and takes the normal handling.

4.6 Decorations

The companion package LuaCJKfntef provides underlining, emphasis dots, and strike-through. It builds on lua-ul, so the decorated text may contain arbitrary commands, CJK and Latin text, and may still be hyphenated and broken between lines.

`\LuaCJKfntefsetup` `\LuaCJKfntefsetup {⟨keyval list⟩}`

Set the keys of the package as a whole. The key `commute` takes `true|false`and is `true` initially: with it every decoration puts its ink where its own keys say, so the order in which two decorations are nested does not change the result; without it the inks of one group of nested decorations are packed into the band their own positions make, outermost first, the rest of the band shared as equal gaps between two neighbours. The two writings then have the same extent and differ only in order, the order xeCJK gives; a `depth` deeper than the group's shallowest ink is overridden, because the whole group follows the shallowest one. In a group the key covers the decorations started inside that group alone. The wave and the slash are drawings whose ink the node list does not place, so they take no part in the packing and keep their position. The same holds for a decoration written inside a box: the box is typeset on its own, so the decoration around it is not in the same list and the ink inside the box does not move; put the box around the text instead, and both decorations stay in one list.

`\CJKunderline` `\CJKunderline [⟨options⟩] {⟨text⟩}`

Underline ⟨*text*⟩. The initial geometry is `depth = 0.2em` and `thickness = 0.4pt`. A leading `*` or `-` is accepted and ignored.

Every decoration takes the keys below; a key that a command does not use is accepted and has no effect. A bare color name with no key means `format =`

⟨*color*⟩. The keys `top` and `bottom` apply to `\CJKunderline` alone and override the computed geometry, whether or not `format` colors the rule. A drawn pattern never changes the height or depth of the line it decorates, so the leading does not grow. The underline, the double rule, the wave, the slash, and a symbol each get one pattern per font size, so a pattern grows with the size of the text.

`thickness` = ⟨*length*⟩: the thickness of the rule, initially `0.4pt`.

`depth` = ⟨*length*⟩: the distance from the baseline to the top of the rule, initially `0.2em`; for the emphasis dots it is their depth below the bottom of the em box.

`textformat` = ⟨*code*⟩: code applied to the text, for example `textformat = \bfseries`.

`format` = ⟨*color name*⟩: the color of the rule, for example `format = red`; it affects the rule only, not the text.

`color` = ⟨*color name*⟩: another name for `format`.

`subtract` = ⟨*length*⟩: taken off the rule at both ends, off by default; `true` means half an em, so two neighbouring commands do not run together.

`skip` = ⟨*true*|**false**⟩: leaves the punctuation out of the rule.

`hidden` = ⟨*true*|**false**⟩: hides the text and keeps the rule, with the width, height, and depth of the hidden text; the text still breaks between lines, and no glyph of it reaches the PDF, so it cannot be read or selected.

`gap` = ⟨*length*⟩: the distance between the two rules of the double line, initially `1.1pt`.

`height` = ⟨*length*⟩: the distance from the baseline to the center of the strike-out rule, initially `0.3em`.

`symbol` = ⟨*code*⟩: a symbol replacing the wave or the dots.

`selectable` = ⟨*true*|**false**⟩: whether the decoration symbol stays as text, initially `false`; with `true` a PDF reader can copy and select it.

Without `format` the rule takes the color of the text it decorates, for example the color set by `\textcolor`.

`sep` is accepted for compatibility and has no effect.

```
\CJKunderline[thickness = 1pt]{有涯}\\
\CJKunderline[format = red]{有涯}\\
\CJKunderline[textformat = \bfseries]
  {有涯}\\
\CJKunderline[hidden]{有涯}
```

有涯
有涯
有涯

```
今天是 \CJKunderline{\hspace*{3em}} 年\\
第 \CJKunderline{\hspace*{2em}} 题
```

今天是 _____ 年
第 ____ 题

```
\CJKunderline{有涯}\CJKunderline{有涯}\\
\CJKunderline[subtract]{吾生也有涯}\\
\CJKunderline[skip]{有涯，无涯}
```

有涯有涯
吾生也有涯
有涯，无涯

\CJKunderdblline `\CJKunderdblline [⟨options⟩] {⟨text⟩}`

Put two parallel rules under ⟨*text*⟩, with the keys listed above. Both rules use `thickness`, the top rule starts at `depth`, and `gap` is the distance between them. A leading `*` or `-` is accepted and ignored.

\CJKunderwave `\CJKunderwave [⟨options⟩] {⟨text⟩}`

Put a wave under ⟨*text*⟩. The wave is drawn with l3draw, and its periods stay joined: one period is `1em` wide and reaches `.25em` above and below. A span of whole characters is an exact number of periods, so the wave runs from the first character's edge to the last one's; when justification stretches the span, the span is divided into whole periods at its final width, so the period follows the stretch of the line. Like the other drawn patterns it leaves the height and depth of the line alone. `symbol` replaces the wave with another symbol: one symbol per em, centered, its top half an em below the baseline.

```
\CJKunderwave{吾生也有涯}\\
\CJKunderwave[symbol = \textbullet]
  {吾生也有涯}\\
\CJKunderdot{吾生也有涯}\\
\CJKunderdot[symbol = \textbullet]
  {吾生也有涯}
```

吾生也有涯
吾生也有涯
• • • • •
吾生也有涯
吾生也有涯
• • • • •

\CJKxout `\CJKxout [⟨options⟩] {⟨text⟩}`

Strike ⟨*text*⟩ out with a repeated slash, like ulem's `\xout`. The slash runs from `-0.15em` to `0.8em` relative to the baseline, so it crosses the whole body of a CJK glyph.

\LuaCJKemdescent The depth of the em box of the current font, usable as a length. A CJK em box is one em square, so an object of height `1em` placed at this depth covers the area a CJK character occupies; a font without OpenType metrics falls back to `.12em`.

\CJKunderanyline `\CJKunderanyline {⟨depth⟩} {⟨object⟩} {⟨text⟩}`

Put ⟨*object*⟩ under ⟨*text*⟩, repeated by `\leaders`. The bottom of ⟨*object*⟩ is placed ⟨*depth*⟩ below the baseline, so for a `\rule` (which has no depth of its own) the rule occupies ⟨*depth*⟩ below the baseline upwards. To cover a CJK character, use the em box of the font: ⟨*depth*⟩ is `\LuaCJKemdescent` and the object is `1em` high, as the example below does. The object fills the em box, which is the area a PDF reader selects: the advance across, the em box down, so a font whose ink is smaller shows its side bearing all round. The type is cached, so the same arguments reuse one underline type.

\CJKunderanysymbol `\CJKunderanysymbol[⟨options⟩]{⟨depth⟩}{⟨symbol⟩}{⟨text⟩}`

The same, with one ⟨*symbol*⟩ per em: the symbol is centered in a cell as wide as a CJK character, so a row of symbols lines up with the characters above them. ⟨*depth*⟩ is the distance from the top of the symbol to the *bottom of the em box*, so `0pt` leaves the same gap below the character as the side bearings leave beside it; a font without OpenType metrics is computed at `.12em`. The option `sep` is accepted for compatibility but has no effect, because the cell is fixed at one em.

```
\CJKunderanyline{\LuaCJKemdescent}
  {\color{yellow}\rule{1pt}{1em}}{有涯}\\
\CJKunderanysymbol{0pt}{\tiny$\triangle$}{有涯}
```

有涯
有涯
△ △

\CJKunderdot `\CJKunderdot [⟨options⟩] {⟨text⟩}`

Put emphasis dots under ⟨*text*⟩. Each CJK character gets one dot, centred in the one-em cell of the character, its top `0.2em` below the bottom of the em box; Latin letters, digits, and spaces get none, each line of a broken span carries its own dots, and the dots follow the characters when the inter-character glue stretches. With `symbol = ⟨symbol⟩` a symbol takes the place of the dots and is placed like the symbol of `\CJKunderanysymbol`: one per em, centered, its top `0.3em` under the bottom of the em box.

```
\CJKunderline{虚室生白，\CJKunderdot{吉祥}止止，
  虚室生白}
```

虚室生白，吉祥止止，
虚室生白

\CJKsout `\CJKsout [⟨options⟩] {⟨text⟩}`

Strike ⟨*text*⟩ through, with the keys listed above. The initial height passes through the middle of a CJK glyph, which sits higher than the middle of the Latin x-height; use `height = 0.5ex` for text that is entirely Latin.

```
\CJKsout[thickness = 2pt, format = red]
  {吾生也有涯}\\
\CJKsout[height = .5ex]{吾生也有涯}
```

吾生也有涯
吾生也有涯

```
\CJKunderline   {吾生也有涯}\\
\CJKunderdblline{吾生也有涯}\\
\CJKunderwave   {吾生也有涯}\\
\CJKunderdot    {吾生也有涯}\\
\CJKsout        {吾生也有涯}\\
\CJKxout        {吾生也有涯}
```

吾生也有涯
吾生也有涯
吾生也有涯
吾生也有涯
吾生也有涯
吾生也有涯

5 Comparison with xeCJK

LuaCJK follows the xeCJK interface where the two engines allow it, so a document that already uses xeCJK mostly needs no change. Only the differences are listed.

5.1 Different behavior

- `Verb`: `env+` behaves like `false`, because both leave the adjustments on.
- Punctuation geometry. A style gives a mark half the size, half with the sentence-ending marks kept full, or 0.7 of it, and the advance and the place of the ink come from the mark's outline: the style takes the outer blank and no more, so a mark whose ink nearly fills its advance keeps the full width. The gap between two adjacent marks, and the length `\LuaCJKsetkern` gives, are the blank between the two inks.
- `\LuaCJKDeclarePunctStyle` takes the three keys `compress-adjacent`, `adjacent-ratio`, and `keep-open-close`; the thirty geometry keys of xeCJK describe its side-bearing model and have no counterpart here.
- `CJKfilltwosides*` does not compare with the natural width.
- The CJK font is applied to the characters before the text is shaped, so the OpenType features of the CJK font take effect: a combining mark attaches to its base, a variation sequence selects its glyph, and the font's own ligatures and `locl` forms appear, two dashes joined into one two-em dash for instance. The spacing, the punctuation advance, and the prohibitions still come from LuaCJK itself.

- In LuaCJKfntef, `format` takes a color name rather than code, and the decoration is drawn by lua-ul but configured through the keys of the package, so the `ulem` option list does not exist.

5.2 Not available

- `NewLineCS`, `EnvCS`, `InlineEnv`, and `PlainEquation`: they exist so that xeCJK can recognize TEX constructs while it scans tokens, and a finished node list needs no hint.
- `PoZheHaoLigature`: the font's own two-em dash ligature applies when it has one, so there is no option that switches it on.
- `RubberPunctSkip`: the advance of a mark is a kern, which is neither a break opportunity nor part of justification, and that is what keeps a fixed-width style stable.
- `CheckFullRight`: it needs the control sequence after the mark, which a node list does not show; use `\LuaCJKnobreak`.

A Implementation

1 ⟨*package⟩

2 ⟨@@=LuaCJK⟩

A.1 Loading and conflicts

LuaCJK only works with LuaTEX. The engine test runs before any other package is loaded, so that a wrong engine stops with a clear message.

```
3 \sys_if_engine_luatex:F
4   {
5     \msg_new:nnn { LuaCJK } { engine }
6       { LuaCJK ~ requires ~ LuaTeX. }
7     \msg_critical:nn { LuaCJK } { engine }
8     \tex_endinput:D
9   }
10 \RequirePackage { fontspec }
```

\LuaCJKversion The version of the package. The version string, taken from the package metadata.

```
11 \tl_const:Nx \c__LuaCJK_version_tl { \ExplFileVersion }
12 \cs_new:Npn \LuaCJKversion { \tl_use:N \c__LuaCJK_version_tl }
```

(*End of definition for* `\LuaCJKversion` *and* `\c__LuaCJK_version_tl`*. This function is documented on page 25.*)

luatexja installs its own CJK font and spacing layer and works on the same callbacks as LuaCJK, so the two are rejected whatever the loading order. xeCJK needs no check here: it stops by itself on any engine other than XETEX.

```
13 \msg_new:nnn { LuaCJK } { luatexja-conflict }
14   {
15     LuaCJK ~ and ~ LuaTeX-ja ~ must ~ not ~ be ~ loaded ~ together.\\
16     Load ~ only ~ one ~ of ~ them.
17   }
18 \cs_if_exist:cT { ver@luatexja.sty }
19   { \msg_critical:nn { LuaCJK } { luatexja-conflict } }
20 \hook_gput_code:nnn { package / luatexja / before } { LuaCJK }
21   { \msg_critical:nn { LuaCJK } { luatexja-conflict } }
```

A.2 Registers, styles, and options

The runtime state lives in attributes and registers, which the options below fill in.

\g__LuaCJK_font_attr
\c__LuaCJK_font_attr_int
\g__LuaCJK_cjk_space_attr
\c__LuaCJK_cjk_space_attr_int
\g__LuaCJK_xecglue_attr
\c__LuaCJK_xecglue_attr_int
\g__LuaCJK_inserted_attr
\c__LuaCJK_inserted_attr_int
\g__LuaCJK_math_font_attr
\c__LuaCJK_math_font_attr_int
\g__LuaCJK_math_bracket_attr
\c__LuaCJK_math_bracket_attr_int
\g__LuaCJK_disabled_attr
\c__LuaCJK_disabled_attr_int
\g__LuaCJK_punct_font_attr
\c__LuaCJK_punct_font_attr_int
\g__LuaCJK_settings_attr
\c__LuaCJK_settings_attr_int
\g__LuaCJK_nobreak_attr
\c__LuaCJK_nobreak_attr_int
\g__LuaCJK_attached_attr
\c__LuaCJK_attached_attr_int
\g__LuaCJK_explicit_space_attr
\c__LuaCJK_explicit_space_attr_int
\g__LuaCJK_shift_attr
\c__LuaCJK_shift_attr_int
\g__LuaCJK_punct_shift_attr
\c__LuaCJK_punct_shift_attr_int
\g__LuaCJK_shifted_attr
\c__LuaCJK_shifted_attr_int

The attributes record, per node, the CJK font, the space and glue policies, whether the node was inserted by LuaCJK, the CJK math font, whether the adjustments are suspended, whether a formula sets one opening or closing delimiter, whether the character is one that attaches to the character in front of it, whether a space is one the document wrote with `\ ` or `\nobreakspace`, the `Baseline` offset of the family that sets the character in scaled points, and whether that offset has been added to the glyph yet. The integer constants hold the allocated attribute numbers for the Lua side.

```
\clist_map_inline:nn
  {
    font,     cjk_space,  xecglue,   inserted,   math_font,
    disabled, punct_font, settings,  nobreak,    attached,
    math_bracket,
    explicit_space,
    shift,    punct_shift,
    shifted
  }
  {
    \exp_args:Nc \newattribute { g__LuaCJK_#1_attr }
    \int_const:cn { c__LuaCJK_#1_attr_int } { \int_use:N \allocationnumber }
  }
\g__LuaCJK_cjk_space_attr   = 0
\g__LuaCJK_xecglue_attr     = 0
\g__LuaCJK_math_font_attr   = 0
\g__LuaCJK_punct_font_attr  = 0
\g__LuaCJK_settings_attr    = 0
\g__LuaCJK_shift_attr       = 0
\g__LuaCJK_punct_shift_attr = 0
```

(*End of definition for* `\g__LuaCJK_font_attr` *and others.*)

\l__LuaCJK_ccglue_skip
\l__LuaCJK_ecglue_skip

The glue used between CJK characters and between CJK and Latin text.

```
\skip_new:N \l__LuaCJK_ccglue_skip
\skip_new:N \l__LuaCJK_ecglue_skip
\bool_new:N \l__LuaCJK_ccglue_touched_bool
\bool_new:N \l__LuaCJK_ecglue_touched_bool
\skip_set:Nn \l__LuaCJK_ccglue_skip { 0pt plus 0.08 \tex_baselineskip:D }
\skip_set:Nn \l__LuaCJK_ecglue_skip
  { 0.3333em plus 0.1667em minus 0.1111em }
```

(*End of definition for* `\l__LuaCJK_ccglue_skip` *and* `\l__LuaCJK_ecglue_skip`.)

`\__LuaCJK_set_ccglue:n`
`\__LuaCJK_set_ecglue:n`

The two glues keep their default relative to the current size, so they are recomputed on every font selection unless the document set them. A value given by the document is taken as it is.

```
\cs_new_protected:Npn \__LuaCJK_set_ccglue:n #1
  {
    \skip_set:Nn \l__LuaCJK_ccglue_skip {#1}
    \bool_set_true:N \l__LuaCJK_ccglue_touched_bool
  }
\cs_new_protected:Npn \__LuaCJK_set_ecglue:n #1
  {
    \skip_set:Nn \l__LuaCJK_ecglue_skip {#1}
    \bool_set_true:N \l__LuaCJK_ecglue_touched_bool
  }
\cs_new_protected:Npn \__LuaCJK_refresh_glues:
  {
    \bool_if:NF \l__LuaCJK_ccglue_touched_bool
      {
        \skip_set:Nn \l__LuaCJK_ccglue_skip
          { 0pt plus 0.08 \tex_baselineskip:D }
      }
    \bool_if:NF \l__LuaCJK_ecglue_touched_bool
      {
        \skip_set:Nn \l__LuaCJK_ecglue_skip
          { 0.3333em plus 0.1667em minus 0.1111em }
      }
  }
```

(*End of definition for* `\__LuaCJK_set_ccglue:n` *and* `\__LuaCJK_set_ecglue:n`.)

`\__LuaCJK_explicit_space:`
`\__LuaCJK_explicit_nobreak_space:`
`\__LuaCJK_nobreak_space:`

The space the document asked for is marked. The node level cannot tell the two apart: a source space and `\ ` reach the spacing pass as the same `spaceskip` glue with the same dimensions, so the mark is the only thing that separates them. `\ ` and `\nobreakspace` become a group around the space token, and `~` reaches the second through `\nobreakspace`. Both are protected, so an `\edef` or an `\write` carries them as one token: the body of an `\edef` is still the space the document wrote, at the price of a message printed with `\ ` showing the control space instead of a space. `\space` stays a source space, because it is the form that appears in messages and has to expand. The definition of `\nobreakspace` is copied before it is replaced, because a body that named `\nobreakspace` again would call the wrapper once the wrapper is in its place. `\scan_stop` ends the value of the attribute: a space token that came straight after it would be taken for the blank

that ends the number. The group that sets the catcode of the space is what lets the two be assigned at all: expl3 ignores a space, so a control space cannot be written in its syntax.

```
\cs_new_eq:NN \__LuaCJK_nobreak_space: \nobreakspace
\cs_new_protected:Npn \__LuaCJK_explicit_space:
  {
    \group_begin:
      \g__LuaCJK_explicit_space_attr = 1 \scan_stop:
      \char_generate:nn { 32 } { 10 }
    \group_end:
  }
\cs_new_protected:Npn \__LuaCJK_explicit_nobreak_space:
  {
    \group_begin:
      \g__LuaCJK_explicit_space_attr = 1 \scan_stop:
      \__LuaCJK_nobreak_space:
    \group_end:
  }
\group_begin:
  \char_set_catcode_space:n { 32 }
  \cs_gset_eq:NN \ \__LuaCJK_explicit_space:
  \cs_gset_eq:NN \nobreakspace \__LuaCJK_explicit_nobreak_space:
\group_end:
```

(*End of definition for* `\__LuaCJK_explicit_space:`, `\__LuaCJK_explicit_nobreak_space:`, *and* `\__LuaCJK_nobreak_space:`.)

A punctuation style is a property list, read through the settings of each node.

`g__LuaCJK_punct_style_prop` The names of the styles LuaCJK knows about, including those defined by `\LuaCJKDeclarePunctStyle`.

```
\prop_new:N \g__LuaCJK_punct_style_prop
\prop_gput:Nnn \g__LuaCJK_punct_style_prop { quanjiao      } { }
\prop_gput:Nnn \g__LuaCJK_punct_style_prop { plain         } { }
\prop_gput:Nnn \g__LuaCJK_punct_style_prop { banjiao       } { }
\prop_gput:Nnn \g__LuaCJK_punct_style_prop { kaiming       } { }
\prop_gput:Nnn \g__LuaCJK_punct_style_prop { CCT           } { }
\prop_gput:Nnn \g__LuaCJK_punct_style_prop { hangmobanjiao } { }
```

(*End of definition for* `\g__LuaCJK_punct_style_prop`.)

`__LuaCJK_set_punct_style:n` Store the selected style, or warn and fall back to `plain` when the name is unknown. The style name read by the Lua side.

`\l__LuaCJK_punct_style_tl`

```
\tl_new:N \l__LuaCJK_punct_style_tl
\tl_set:Nn \l__LuaCJK_punct_style_tl { quanjiao }
```

```
\msg_new:nnn { LuaCJK } { punct-style-unknown }
  {
    PunctStyle ~ `#1' ~ is ~ not ~ implemented.\\
    Using ~ `plain' ~ instead.
  }
\cs_new_protected:Npn \__LuaCJK_set_punct_style:n #1
  {
    \prop_if_in:NnTF \g__LuaCJK_punct_style_prop {#1}
      { \tl_set:Nn \l__LuaCJK_punct_style_tl {#1} }
      {
        \msg_warning:nnn { LuaCJK } { punct-style-unknown } {#1}
        \tl_set:Nn \l__LuaCJK_punct_style_tl { plain }
      }
  }
```

(*End of definition for* `\__LuaCJK_set_punct_style:n` *and* `\l__LuaCJK_punct_style_tl`.)

`\LuaCJKEditPunctStyle` Change the parameters of an existing style. A key that is not given keeps its current value, which is why the three "given" switches are tracked here instead of starting from the defaults. Whether the corresponding key was part of the edit.

```
\bool_new:N \l__LuaCJK_punct_compress_given_bool
\bool_new:N \l__LuaCJK_punct_ratio_given_bool
\bool_new:N \l__LuaCJK_punct_keep_given_bool
\tl_new:N \l__LuaCJK_punct_compress_value_tl
\tl_new:N \l__LuaCJK_punct_keep_value_tl

\keys_define:nn { LuaCJK / punct-edit }
  {
    compress-adjacent .code:n =
      {
        \bool_set_true:N \l__LuaCJK_punct_compress_given_bool
        \tl_set:Nn \l__LuaCJK_punct_compress_value_tl {#1}
      },
    adjacent-ratio    .code:n =
      {
        \bool_set_true:N \l__LuaCJK_punct_ratio_given_bool
        \fp_set:Nn \l__LuaCJK_punct_ratio_fp {#1}
      },
    keep-open-close   .code:n =
      {
        \bool_set_true:N \l__LuaCJK_punct_keep_given_bool
        \tl_set:Nn \l__LuaCJK_punct_keep_value_tl {#1}
      },
  }
```

```
\NewDocumentCommand \LuaCJKEditPunctStyle { m m }
  {
    \bool_set_false:N \l__LuaCJK_punct_compress_given_bool
    \bool_set_false:N \l__LuaCJK_punct_ratio_given_bool
    \bool_set_false:N \l__LuaCJK_punct_keep_given_bool
    \keys_set:nn { LuaCJK / punct-edit } {#2}
    \lua_now:e
      {
        LuaCJK.edit_punct_style
          ( "\lua_escape:e {#1}" ,
            \bool_if:NTF \l__LuaCJK_punct_compress_given_bool
              { \l__LuaCJK_punct_compress_value_tl  } { nil },
            \bool_if:NTF \l__LuaCJK_punct_ratio_given_bool
              { \fp_use:N \l__LuaCJK_punct_ratio_fp } { nil },
            \bool_if:NTF \l__LuaCJK_punct_keep_given_bool
              { \l__LuaCJK_punct_keep_value_tl      } { nil } )
      }
  }
```

(*End of definition for* `\LuaCJKEditPunctStyle` *and others. This function is documented on page 38.*)

`\LuaCJKsetwidth` Give specific punctuation marks a fixed advance, overriding the selected style. The star form of xeCJK, which sets the width used at the start or end of a line, has no counterpart yet and only warns. Guards the one-time warning for the star form. `\l__LuaCJK_bound_width_warned_bool`

```
\bool_new:N \l__LuaCJK_bound_width_warned_bool
\msg_new:nnn { LuaCJK } { bound-width-unsupported }
  {
    The ~ star ~ form ~ of ~ \token_to_str:N \LuaCJKsetwidth, ~ which ~
    sets ~ the ~ width ~ at ~ the ~ start ~ or ~ end ~ of ~ a ~ line, ~ is ~
    not ~ implemented.\\
    The ~ width ~ is ~ applied ~ to ~ every ~ position ~ instead.
  }

\NewDocumentCommand \LuaCJKsetwidth { s m m }
  {
    \IfBooleanT {#1}
      {
        \bool_if:NF \l__LuaCJK_bound_width_warned_bool
          {
            \bool_set_true:N \l__LuaCJK_bound_width_warned_bool
            \msg_warning:nn { LuaCJK } { bound-width-unsupported }
          }
      }
    \lua_now:e
```

```
      {
        LuaCJK.set_punct_width
          ( "\lua_escape:e {#2}" , \dim_to_decimal_in_sp:n {#3} )
      }
  }
```

(*End of definition for* `\LuaCJKsetwidth` *and* `\l__LuaCJK_bound_width_warned_bool`. *This function is documented on page 39.*)

`\LuaCJKsetkern` Set the gap between two adjacent punctuation marks, overriding the selected style. Only the first character of each argument is used.

```
\NewDocumentCommand \LuaCJKsetkern { m m m }
  {
    \lua_now:e
      {
        LuaCJK.set_punct_kern
          ( "\lua_escape:e {#1}" , "\lua_escape:e {#2}"
          , \dim_to_decimal_in_sp:n {#3} )
      }
  }
```

(*End of definition for* `\LuaCJKsetkern`. *This function is documented on page 39.*)

A single CJK font rarely covers every block of Unicode. A sub-block gives a range of characters its own font inside a family, named by an option of the font commands.

`\LuaCJKDeclareSubCJKBlock` `\g__LuaCJK_block_prop` `g__LuaCJK_block_names_clist` The ranges of each block, and the declaration order that decides which block wins when the ranges overlap.

```
\prop_new:N \g__LuaCJK_block_prop
\clist_new:N \g__LuaCJK_block_names_clist
\NewDocumentCommand \LuaCJKDeclareSubCJKBlock { m m }
  {
    \__LuaCJK_build_ranges:n {#2}
    \prop_gput:NnV \g__LuaCJK_block_prop {#1} \l__LuaCJK_ranges_tl
    \clist_if_in:NnF \g__LuaCJK_block_names_clist {#1}
      { \clist_gput_right:Nn \g__LuaCJK_block_names_clist {#1} }
    \lua_now:e
      {
        LuaCJK.declare_block
          ( "\lua_escape:e {#1}" , { \l__LuaCJK_ranges_tl } )
      }
  }
```

(*End of definition for* `\LuaCJKDeclareSubCJKBlock`, `\g__LuaCJK_block_prop`, *and* `\g__LuaCJK_block_names_clist`. *This function is documented on page 28.*)

\LuaCJKCancelSubCJKBlock
\LuaCJKRestoreSubCJKBlock

Turn a block off or on again. The list of canceled blocks is part of the settings of the text, so the change ends with the group; the fonts of the current family are refreshed so it takes effect at once.

```
\clist_new:N \l__LuaCJK_blocks_off_clist
\cs_new_protected:Npn \__LuaCJK_set_block_enabled:nn #1#2
  {
    \clist_map_inline:nn {#1}
      {
        \str_if_eq:nnTF {#2} { false }
          {
            \clist_if_in:NnF \l__LuaCJK_blocks_off_clist {##1}
              { \clist_put_right:Nn \l__LuaCJK_blocks_off_clist {##1} }
          }
          { \clist_remove_all:Nn \l__LuaCJK_blocks_off_clist {##1} }
      }
    \__LuaCJK_settings_push:
    \__LuaCJK_update_font:
  }
\NewDocumentCommand \LuaCJKCancelSubCJKBlock { m }
  { \__LuaCJK_set_block_enabled:nn {#1} { false } }
\NewDocumentCommand \LuaCJKRestoreSubCJKBlock { m }
  { \__LuaCJK_set_block_enabled:nn {#1} { true } }
```

(*End of definition for* `\LuaCJKCancelSubCJKBlock` *and* `\LuaCJKRestoreSubCJKBlock`. *These functions are documented on page* 29.)

__LuaCJK_extract_font_keys:nn
__LuaCJK_parse_font_options:n

Pull the keys that LuaCJK keeps for itself out of a font option list: the block names, which name a font of their own, and `Baseline`. The remaining options go into `\l__LuaCJK_features_tl`, which is what fontspec is given, and `Baseline` goes into `\l__LuaCJK_shift_tl`. A list that names neither is handed to fontspec exactly as it was written, because the parse would only respell it. The block font of each family and block, the blocks found in the option list being processed, the filtered options, the vertical offset `Baseline` asks for, and scratch variables for splitting one option at the equals sign.

```
\prop_new:N \g__LuaCJK_family_block_prop
\prop_new:N \g__LuaCJK_pending_block_prop
\tl_new:N \l__LuaCJK_features_tl
\tl_new:N \l__LuaCJK_block_key_tl
\tl_new:N \l__LuaCJK_option_key_tl
\seq_new:N \l__LuaCJK_option_seq
\bool_new:N \l__LuaCJK_is_block_bool
\tl_new:N \l__LuaCJK_shift_tl
\cs_new_protected:Npn \__LuaCJK_put_feature:n #1
  {
```

```
    \tl_if_empty:NTF \l__LuaCJK_features_tl
      { \tl_set:Nn \l__LuaCJK_features_tl {#1} }
      { \tl_put_right:Nn \l__LuaCJK_features_tl { , ~ #1 } }
  }
\cs_new_protected:Npn \__LuaCJK_font_option:nn #1#2
  {
    \tl_set:Nn \l__LuaCJK_option_key_tl {#1}
    \tl_trim_spaces:N \l__LuaCJK_option_key_tl
    \str_if_eq:eeTF { \l__LuaCJK_option_key_tl } { Baseline }
      { \tl_set:Ne \l__LuaCJK_shift_tl {#2} }
      {
        \bool_set_false:N \l__LuaCJK_is_block_bool
        \clist_map_inline:Nn \g__LuaCJK_block_names_clist
          {
            \str_if_eq:eeT {##1} { \l__LuaCJK_option_key_tl }
              { \bool_set_true:N \l__LuaCJK_is_block_bool }
          }
        \bool_if:NTF \l__LuaCJK_is_block_bool
          { \prop_gput:NVn \g__LuaCJK_pending_block_prop
              \l__LuaCJK_option_key_tl {#2} }
          { \__LuaCJK_put_feature:n { #1 = {#2} } }
      }
  }
\cs_new_protected:Npn \__LuaCJK_font_option:n #1
  { \__LuaCJK_put_feature:n {#1} }
\cs_new_protected:Npn \__LuaCJK_parse_font_options:n #1
  {
    \tl_clear:N \l__LuaCJK_features_tl
    \tl_clear:N \l__LuaCJK_shift_tl
    \keyval_parse:NNn \__LuaCJK_font_option:n \__LuaCJK_font_option:nn {#1}
  }
\cs_new_protected:Npn \__LuaCJK_extract_font_keys:nn #1#2
  {
    \prop_gclear:N \g__LuaCJK_pending_block_prop
    \__LuaCJK_parse_font_options:n {#2}
    \prop_gput:NnV \g__LuaCJK_shift_decl_prop {#1} \l__LuaCJK_shift_tl
    \prop_if_empty:NTF \g__LuaCJK_pending_block_prop
      {
        \tl_if_empty:NT \l__LuaCJK_shift_tl
          { \tl_set:Nn \l__LuaCJK_features_tl {#2} }
      }
      { \__LuaCJK_register_block_fonts:n {#1} }
  }
\cs_new_protected:Npn \__LuaCJK_register_block_fonts:n #1
  {
```

```
    \prop_map_inline:Nn \g__LuaCJK_pending_block_prop
      {
        \group_begin:
          \__LuaCJK_build_features:n { }
          \fontspec_gset_family:NVn \l__LuaCJK_tmpa_tl
            \l__LuaCJK_feature_tl {##2}
          \tl_set:Ne \l__LuaCJK_block_key_tl { #1 | ##1 }
          \prop_gput:NVV \g__LuaCJK_family_block_prop
            \l__LuaCJK_block_key_tl \l__LuaCJK_tmpa_tl
        \group_end:
      }
  }
```

(*End of definition for* `\__LuaCJK_extract_font_keys:nn` *and others.*)

`\LuaCJKDeclareCharClass` Override the class of a code-point range. The range list uses the xeCJK spelling: hexadecimal numbers with a leading `"`, and `"START -> "END` for a range. The Lua table being built, the two halves of one range, and the current hexadecimal number and its value.

```
\tl_new:N \l__LuaCJK_ranges_tl
\seq_new:N \l__LuaCJK_range_seq
\tl_new:N \l__LuaCJK_range_pair_tl
\tl_new:N \l__LuaCJK_hex_tl
\tl_new:N \l__LuaCJK_hex_int_tl

\cs_new_protected:Npn \__LuaCJK_hex_store:n #1
  {
    \tl_set:Nn \l__LuaCJK_hex_tl {#1}
    \tl_replace_all:Nnn \l__LuaCJK_hex_tl { " } { }
    \tl_trim_spaces:N \l__LuaCJK_hex_tl
    \tl_set:Ne \l__LuaCJK_hex_int_tl
      { \exp_args:NV \int_from_hex:n \l__LuaCJK_hex_tl }
  }
\cs_new_protected:Npn \__LuaCJK_range_entry:n #1
  {
    \seq_set_split:Nnn \l__LuaCJK_range_seq { -> } {#1}
    \tl_clear:N \l__LuaCJK_range_pair_tl
    \seq_map_inline:Nn \l__LuaCJK_range_seq
      {
        \__LuaCJK_hex_store:n {##1}
        \tl_put_right:Nx \l__LuaCJK_range_pair_tl
          { \exp_not:V \l__LuaCJK_hex_int_tl , }
      }
    \tl_put_right:Nx \l__LuaCJK_ranges_tl
      { { \exp_not:V \l__LuaCJK_range_pair_tl } , }
```

```
  }
\cs_new_protected:Npn \__LuaCJK_build_ranges:n #1
  {
    \tl_clear:N \l__LuaCJK_ranges_tl
    \clist_map_inline:nn {#1} { \__LuaCJK_range_entry:n {##1} }
  }
\msg_new:nnn { LuaCJK } { unknown-char-class }
  { Unknown ~ character   ~ class ~ `#1' ~ is ~ being ~ ignored. }
\msg_new:nnn { LuaCJK } { unknown-punct-class }
  { Unknown ~ punctuation ~ class ~ `#1' ~ is ~ being ~ ignored. }
\NewDocumentCommand \LuaCJKDeclareCharClass { m m }
  {
    \str_case:nnF {#1}
      {
        { CJK         } { \__LuaCJK_push_class:nn { true    } {#2} }
        { Default     } { \__LuaCJK_push_class:nn { false   } {#2} }
        { FullLeft    } { \__LuaCJK_push_punct:nn { "open"  } {#2} }
        { HalfLeft    } { \__LuaCJK_push_punct:nn { "half-
open" } {#2} }
        { FullRight   } { \__LuaCJK_push_punct:nn { "close" } {#2} }
        { HalfRight   } { \__LuaCJK_push_punct:nn { "half-
close" } {#2} }
        { NormalSpace } { \__LuaCJK_push_normal_space:n       {#2} }
        { CJStarter   } { \__LuaCJK_push_class:nn { true    } {#2} }
        { CM          } { \__LuaCJK_push_class:nn { true    } {#2} }
        { Boundary    } { \__LuaCJK_push_class:nn { true    } {#2} }
      } { \msg_warning:nnn { LuaCJK } { unknown-char-class } {#1} }
  }
\cs_new_protected:Npn \__LuaCJK_push_normal_space:n #1
  {
    \__LuaCJK_build_ranges:n {#1}
    \lua_now:e
      { LuaCJK.add_normal_spaced ( { \l__LuaCJK_ranges_tl } ) }
  }
\cs_new_protected:Npn \__LuaCJK_push_class:nn #1#2
  {
    \__LuaCJK_build_ranges:n {#2}
    \lua_now:e
      { LuaCJK.declare_class ( { \l__LuaCJK_ranges_tl } , #1 ) }
  }
```

(*End of definition for* `\LuaCJKDeclareCharClass` *and others. This function is documented on page 38.*)

`\LuaCJKDeclarePunctClass` Override the punctuation group of a range, or with `none` remove the punctuation

treatment from it.

```
\NewDocumentCommand \LuaCJKDeclarePunctClass { m m }
  {
    \str_case:nnF {#1}
      {
        { open     } { \__LuaCJK_push_punct:nn { "open" } {#2}     }
        { close    } { \__LuaCJK_push_punct:nn { "close" } {#2}    }
        { middle   } { \__LuaCJK_push_punct:nn { "middle" } {#2}   }
        { dash     } { \__LuaCJK_push_punct:nn { "dash" } {#2}     }
        { ellipsis } { \__LuaCJK_push_punct:nn { "ellipsis" } {#2} }
        { half-open   } { \__LuaCJK_push_punct:nn { "half-
open" } {#2} }
        { half-close  } { \__LuaCJK_push_punct:nn { "half-
close" } {#2} }
        { latin-close } { \__LuaCJK_push_punct:nn { "latin-
close" } {#2} }
        { none     } { \__LuaCJK_push_punct:nn { false } {#2}      }
      } { \msg_warning:nnn { LuaCJK } { unknown-punct-class } {#1} }
  }
\cs_new_protected:Npn \__LuaCJK_push_punct:nn #1#2
  {
    \__LuaCJK_build_ranges:n {#2}
    \lua_now:e
      { LuaCJK.declare_punct ( { \l__LuaCJK_ranges_tl } , #1 ) }
  }
```

(*End of definition for* `\LuaCJKDeclarePunctClass`*. This function is documented on page 38.*)

`\__LuaCJK_set_charset:nnn` Replace, extend, or reduce one of the punctuation character sets. The ranges are converted the same way as for `\LuaCJKDeclareCharClass`.

```
\cs_new_protected:Npn \__LuaCJK_set_charset:nnn #1#2#3
  {
    \__LuaCJK_build_ranges:n {#3}
    \lua_now:e
      {
        LuaCJK.set_charset
          ( "#1" , "#2" , { \l__LuaCJK_ranges_tl } )
      }
  }
```

(*End of definition for* `\__LuaCJK_set_charset:nnn`.)

`\LuaCJKResetCharClass` Drop every class and punctuation override declared so far, and put the normal-spaced set back to the characters of `\normalspacedchars`.

```
\NewDocumentCommand \LuaCJKResetCharClass { }
```

```
  {
    \lua_now:n { LuaCJK.reset_classes() }
    \__LuaCJK_set_normal_spaced_chars:
  }
```

(*End of definition for* `\LuaCJKResetCharClass`*. This function is documented on page 38.*)

`\LuaCJKDeclarePunctStyle` Define a style from a small set of keys and hand the result to Lua. The values collected from the keys, with the defaults used when a key is not given.

```
\bool_new:N \l__LuaCJK_punct_compress_bool
\fp_new:N \l__LuaCJK_punct_ratio_fp
\bool_new:N \l__LuaCJK_punct_keep_bool

\keys_define:nn { LuaCJK / punct-style }
  {
    compress-adjacent .bool_set:N = \l__LuaCJK_punct_compress_bool,
    adjacent-ratio    .fp_set:N   = \l__LuaCJK_punct_ratio_fp,
    keep-open-close   .bool_set:N = \l__LuaCJK_punct_keep_bool,
  }
\NewDocumentCommand \LuaCJKDeclarePunctStyle { m m }
  {
    \bool_set_true:N \l__LuaCJK_punct_compress_bool
    \fp_set:Nn \l__LuaCJK_punct_ratio_fp { 0.5 }
    \bool_set_true:N \l__LuaCJK_punct_keep_bool
    \keys_set:nn { LuaCJK / punct-style } {#2}
    \prop_gput:Nnn \g__LuaCJK_punct_style_prop {#1} { }
    \lua_now:e
      {
        LuaCJK.define_punct_style(
          "\lua_escape:e {#1}",
          \bool_if:NTF \l__LuaCJK_punct_compress_bool { true } { false },
          \fp_use:N \l__LuaCJK_punct_ratio_fp,
          \bool_if:NTF \l__LuaCJK_punct_keep_bool { true } { false } )
      }
  }
```

(*End of definition for* `\LuaCJKDeclarePunctStyle` *and others. This function is documented on page 39.*)

`\__LuaCJK_set_autofake:nn` Turn fake bold or slant on or off. A number both enables the feature and sets the factor, as in xeCJK.

```
\cs_new_protected:Npn \__LuaCJK_set_autofake:nn #1#2
  {
    \str_case:nnF {#2}
      {
        { true  }
```

```
      {
        \str_if_eq:nnTF {#1} { bold }
          {
            \tl_gset_eq:NN \g__LuaCJK_autofake_bold_tl
              \g__LuaCJK_embolden_factor_tl
          }
          {
            \tl_gset_eq:NN \g__LuaCJK_autofake_slant_tl
              \g__LuaCJK_slant_factor_tl
          }
      }
      { false }
      {
        \str_if_eq:nnTF {#1} { bold }
          { \tl_gclear:N \g__LuaCJK_autofake_bold_tl  }
          { \tl_gclear:N \g__LuaCJK_autofake_slant_tl }
      }
    }
    {
      \str_if_eq:nnTF {#1} { bold }
        {
          \tl_gset:Nn \g__LuaCJK_embolden_factor_tl {#2}
          \tl_gset:Nn \g__LuaCJK_autofake_bold_tl   {#2}
        }
        {
          \tl_gset:Nn \g__LuaCJK_slant_factor_tl    {#2}
          \tl_gset:Nn \g__LuaCJK_autofake_slant_tl  {#2}
        }
    }
}
```

(*End of definition for* `\__LuaCJK_set_autofake:nn`.)

`\LuaCJKsetup` The same key path serves `\LuaCJKsetup` and the package options. The keys are defined here, but the package options are read at the end of the package, once every helper a key code calls has been defined and the Lua module has been loaded, which `CJLineBreak` needs: a key whose code calls a helper of the font section would otherwise act through `\LuaCJKsetup` alone and leave an undefined control sequence behind when it was given as a package option. The switches read by the Lua side, the `Verb` setting, whether a block of verbatim material is being set, the configuration file name, the fake bold and slant parameters, and the fallback fonts.

```
\tl_new:N  \l__LuaCJK_punct_kinsoku_tl
\tl_set:Nn \l__LuaCJK_punct_kinsoku_tl { 1 }
```


\l__LuaCJK_cj_strict_tl
\l__LuaCJK_verb_mode_tl
\l__LuaCJK_local_config_tl
\l__LuaCJK_verbatim_block_bool
\l__LuaCJK_check_single_tl
\l__LuaCJK_widow_penalty_int
\l__LuaCJK_punct_width_dim
\l__LuaCJK_punct_bound_width_dim
\l__LuaCJK_punct_width_given_bool
\l__LuaCJK_punct_bound_width_given_bool
\l__LuaCJK_break_between_puncts_bool
\g__LuaCJK_autofake_bold_tl

```
\tl_new:N  \l__LuaCJK_cj_strict_tl
\tl_set:Nn \l__LuaCJK_cj_strict_tl { 0 }
\tl_new:N  \l__LuaCJK_cjk_math_tl
\tl_set:Nn \l__LuaCJK_cjk_math_tl { 0 }
\tl_new:N  \l__LuaCJK_latin_punct_tl
\tl_set:Nn \l__LuaCJK_latin_punct_tl { 0 }
\tl_new:N  \l__LuaCJK_verb_mode_tl
\tl_set:Nn \l__LuaCJK_verb_mode_tl { true }
\bool_new:N \l__LuaCJK_verbatim_block_bool
\bool_new:N \l__LuaCJK_break_between_puncts_bool
\dim_new:N  \l__LuaCJK_punct_width_dim
\dim_set:Nn \l__LuaCJK_punct_width_dim { 0pt }
\bool_new:N \l__LuaCJK_punct_width_given_bool
\dim_new:N  \l__LuaCJK_punct_bound_width_dim
\dim_set:Nn \l__LuaCJK_punct_bound_width_dim { 0pt }
\bool_new:N \l__LuaCJK_punct_bound_width_given_bool
\tl_new:N \l__LuaCJK_check_single_tl
\tl_set:Nn \l__LuaCJK_check_single_tl { 0 }
\int_new:N \l__LuaCJK_widow_penalty_int
\int_set:Nn \l__LuaCJK_widow_penalty_int { 10000 }
\tl_new:N  \l__LuaCJK_local_config_tl
\tl_new:N  \g__LuaCJK_autofake_bold_tl
\tl_new:N  \g__LuaCJK_autofake_slant_tl
\tl_new:N  \g__LuaCJK_embolden_factor_tl
\tl_gset:Nn \g__LuaCJK_embolden_factor_tl { 4 }
\tl_new:N  \g__LuaCJK_slant_factor_tl
\tl_gset:Nn \g__LuaCJK_slant_factor_tl { 0.167 }
\prop_new:N \g__LuaCJK_fallback_prop
\bool_new:N \l__LuaCJK_auto_fallback_bool
\bool_new:N \g__LuaCJK_load_lxgw_bool
\tl_new:N \l__LuaCJK_fallback_family_tl
\tl_new:N \l__LuaCJK_fallback_key_tl
\prop_new:N \g__LuaCJK_shift_decl_prop
\prop_new:N \g__LuaCJK_font_shift_prop
\prop_new:N \g__LuaCJK_shift_pushed_prop
\tl_new:N \l__LuaCJK_shift_dim_tl
\tl_new:N \l__LuaCJK_shift_fid_tl
\tl_new:N \l__LuaCJK_shift_key_tl
\tl_new:N \g__LuaCJK_shift_value_tl
\tl_new:N \l__LuaCJK_shift_old_tl
\cs_generate_variant:Nn \prop_get:NnNF { NVNF }
\cs_generate_variant:Nn \prop_gput:Nnn { NVV , NnV }
\tl_new:N \l__LuaCJK_fallback_ids_tl
\tl_new:N \l__LuaCJK_fontid_tl
```

```
\keys_define:nn { LuaCJK / options }
  {
    CJKglue             .code:n      = { \__LuaCJK_set_ccglue:n {#1} },
    CJKecglue           .code:n      = { \__LuaCJK_set_ecglue:n {#1} },
    CJKspace            .choice:,
    CJKspace / true     .code:n      = { \g__LuaCJK_cjk_space_attr = 1 },
    CJKspace / false    .code:n      = { \g__LuaCJK_cjk_space_attr = 0 },
    CJKspace            .default:n   = { true  },
    CJKspace            .initial:n   = { false },
    xCJKecglue          .choice:,
    xCJKecglue / true   .code:n      = { \g__LuaCJK_xecglue_attr = 1 },
    xCJKecglue / false  .code:n      = { \g__LuaCJK_xecglue_attr = 0 },
    xCJKecglue          .default:n   = { true  },
    xCJKecglue          .initial:n   = { false },
    PunctStyle          .code:n      = { \__LuaCJK_set_punct_style:n {#1} },
    PunctStyle          .default:n   = { quanjiao },
    PunctKinsoku        .choice:,
    PunctKinsoku / true .code:n      =
      { \tl_set:Nn \l__LuaCJK_punct_kinsoku_tl { 1 } },
    PunctKinsoku / false.code:n      =
      { \tl_set:Nn \l__LuaCJK_punct_kinsoku_tl { 0 } },
    PunctKinsoku        .default:n   = { true  },
    CJLineBreak         .choice:,
    CJLineBreak / normal.code:n      =
      { \tl_set:Nn \l__LuaCJK_cj_strict_tl { 0 } },
    CJLineBreak / strict.code:n      =
      {
        \tl_set:Nn \l__LuaCJK_cj_strict_tl { 1 }
        \__LuaCJK_load_cj:
      },
    CJLineBreak         .default:n   = { strict },
    CJKmath             .choice:,
    CJKmath / true      .code:n      =
      { \tl_set:Nn \l__LuaCJK_cjk_math_tl { 1 } \__LuaCJK_update_font: },
    CJKmath / false     .code:n      =
      { \tl_set:Nn \l__LuaCJK_cjk_math_tl { 0 } \__LuaCJK_update_font: },
    CJKmath             .default:n   = { true  },
    Verb                .choice:,
    Verb / true         .code:n      =
      { \tl_set:Nn \l__LuaCJK_verb_mode_tl { true  } },
    Verb / env          .code:n      =
      { \tl_set:Nn \l__LuaCJK_verb_mode_tl { env   } },
    Verb / env+         .code:n      =
      { \tl_set:Nn \l__LuaCJK_verb_mode_tl { env+  } },
    Verb / false        .code:n      =
```

```
538       { \tl_set:Nn \l__LuaCJK_verb_mode_tl { false } },
539     Verb                .default:n   = { true },
540     KaiMingPunct        .code:n      =
541       { \__LuaCJK_set_charset:nnn { kaiming } { set    } {#1} },
542     KaiMingPunct+       .code:n      =
543       { \__LuaCJK_set_charset:nnn { kaiming } { add    } {#1} },
544     KaiMingPunct-       .code:n      =
545       { \__LuaCJK_set_charset:nnn { kaiming } { remove } {#1} },
546     MiddlePunct         .code:n      =
547       { \__LuaCJK_set_charset:nnn { middle  } { set    } {#1} },
548     MiddlePunct+        .code:n      =
549       { \__LuaCJK_set_charset:nnn { middle  } { add    } {#1} },
550     MiddlePunct-        .code:n      =
551       { \__LuaCJK_set_charset:nnn { middle  } { remove } {#1} },
552     LongPunct           .code:n      =
553       { \__LuaCJK_set_charset:nnn { long    } { set    } {#1} },
554     LongPunct+          .code:n      =
555       { \__LuaCJK_set_charset:nnn { long    } { add    } {#1} },
556     LongPunct-          .code:n      =
557       { \__LuaCJK_set_charset:nnn { long    } { remove } {#1} },
558     NoBreakLongPunct    .code:n      =
559       { \__LuaCJK_set_charset:nnn { nobreak } { set    } {#1} },
560     NoBreakLongPunct+   .code:n      =
561       { \__LuaCJK_set_charset:nnn { nobreak } { add    } {#1} },
562     NoBreakLongPunct-   .code:n      =
563       { \__LuaCJK_set_charset:nnn { nobreak } { remove } {#1} },
564     PunctWidth          .code:n      =
565       {
566         \str_if_eq:nnTF {#1} { false }
567           {
568             \bool_set_false:N \l__LuaCJK_punct_width_given_bool
569             \dim_zero:N \l__LuaCJK_punct_width_dim
570           }
571           {
572             \bool_set_true:N \l__LuaCJK_punct_width_given_bool
573             \dim_set:Nn \l__LuaCJK_punct_width_dim {#1}
574           }
575       },
576     PunctWidth          .default:n   = { false },
577     PunctFamily         .code:n      =
578       {
579         \str_if_eq:nnTF {#1} { false }
580           { \tl_clear:N \l__LuaCJK_punct_family_tl }
581           { \tl_set:Ne \l__LuaCJK_punct_family_tl {#1} }
582         \__LuaCJK_update_font:
```

```
  },
PunctFamily         .default:n   = { false },
PunctBoundWidth     .code:n      =
  {
    \str_if_eq:nnTF {#1} { false }
      {
        \bool_set_false:N \l__LuaCJK_punct_bound_width_given_bool
        \dim_zero:N \l__LuaCJK_punct_bound_width_dim
      }
      {
        \bool_set_true:N \l__LuaCJK_punct_bound_width_given_bool
        \dim_set:Nn \l__LuaCJK_punct_bound_width_dim {#1}
      }
  },
PunctBoundWidth     .default:n   = { false },
LatinPunct          .choice:,
LatinPunct / true   .code:n      =
  { \tl_set:Nn \l__LuaCJK_latin_punct_tl { 1 } },
LatinPunct / false  .code:n      =
  { \tl_set:Nn \l__LuaCJK_latin_punct_tl { 0 } },
LatinPunct          .default:n   = { true  },
LatinPunct          .initial:n   = { false },
AllowBreakBetweenPuncts
                    .choice:,
AllowBreakBetweenPuncts / true
                    .code:n      =
  { \bool_set_true:N \l__LuaCJK_break_between_puncts_bool },
AllowBreakBetweenPuncts / false
                    .code:n      =
  { \bool_set_false:N \l__LuaCJK_break_between_puncts_bool },
AllowBreakBetweenPuncts
                    .default:n   = { true  },
CheckSingle         .choice:,
CheckSingle / true  .code:n      =
  { \tl_set:Nn \l__LuaCJK_check_single_tl { 1 } },
CheckSingle / false .code:n      =
  { \tl_set:Nn \l__LuaCJK_check_single_tl { 0 } },
CheckSingle         .default:n   = { true },
WidowPenalty        .code:n      =
  { \int_set:Nn \l__LuaCJK_widow_penalty_int {#1} },
LocalConfig         .code:n      =
  {
    \str_case:nnF {#1}
      {
        { true  } { \tl_set:Nn \l__LuaCJK_local_config_tl { luacjk.cfg } }
```

```
            { false } { \tl_clear:N \l__LuaCJK_local_config_tl          }
          } { \tl_set:Nn \l__LuaCJK_local_config_tl { luacjk-
#1.cfg } }
      },
    LocalConfig         .default:n   = { true  },
    AutoFakeBold        .code:n      = { \__LuaCJK_set_autofake:nn { bold  } {#1} },
    AutoFakeBold        .default:n   = { true  },
    AutoFakeBold        .initial:n   = { false },
    AutoFakeSlant       .code:n      = { \__LuaCJK_set_autofake:nn { slant } {#1} },
    AutoFakeSlant       .default:n   = { true  },
    AutoFakeSlant       .initial:n   = { false },
    EmboldenFactor      .code:n      =
      {
        \tl_gset:Nn \g__LuaCJK_embolden_factor_tl {#1}
        \tl_if_empty:NF \g__LuaCJK_autofake_bold_tl
          { \tl_gset:Nn \g__LuaCJK_autofake_bold_tl {#1} }
      },
    AutoFallBack        .choice:,
    AutoFallBack / true .code:n      =
      {
        \bool_set_true:N \l__LuaCJK_auto_fallback_bool
        \__LuaCJK_refresh_fallbacks:
      },
    AutoFallBack / false.code:n      =
      { \bool_set_false:N \l__LuaCJK_auto_fallback_bool },
    AutoFallBack        .default:n   = { true  },
    LoadLXGW            .choice:,
    LoadLXGW / true     .code:n      =
      { \bool_gset_true:N \g__LuaCJK_load_lxgw_bool },
    LoadLXGW / false    .code:n      =
      { \bool_gset_false:N \g__LuaCJK_load_lxgw_bool },
    LoadLXGW            .default:n   = { true  },
    LoadLXGW            .initial:n   = { true  },
    SlantFactor         .code:n      =
      {
        \tl_gset:Nn \g__LuaCJK_slant_factor_tl {#1}
        \tl_if_empty:NF \g__LuaCJK_autofake_slant_tl
          { \tl_gset:Nn \g__LuaCJK_autofake_slant_tl {#1} }
      },
    unknown             .code:n      =
      {
        \msg_warning:nne { LuaCJK } { unknown-option }
          { \l_keys_key_str }
      },
  }
```

```
\msg_new:nnn { LuaCJK } { unknown-option }
  { Unknown ~ option ~ `#1' ~ is ~ being ~ ignored. }

\NewDocumentCommand \LuaCJKsetup { m }
  {
    \keys_set:nn { LuaCJK / options } {#1}
    \__LuaCJK_settings_push:
  }
```

(*End of definition for* `\LuaCJKsetup` *and others. This function is documented on page 25.*)

`\__LuaCJK_settings_push:` Hand the current options to Lua, which stores them once and selects the stored copy with an attribute. Called after `\LuaCJKsetup` and after every font selection, because the default glues and the size-dependent distances have to be recomputed for the new size. The attribute is what makes a group that ends before the paragraph still decide how its own text is set.

```
\cs_new:Npn \__LuaCJK_bool_js:N #1
  { \str_if_eq:VnTF #1 { 1 } { true } { false } }
\cs_new:Npn \__LuaCJK_width_js:Nn #1#2
  { \bool_if:NTF #1 { \dim_to_decimal_in_sp:n {#2} } { false } }
\cs_new_protected:Npn \__LuaCJK_settings_push:
  {
    \bool_if:NT \g__LuaCJK_lua_loaded_bool
      {
        \lua_now:e
          {
            LuaCJK.new_settings
              (
                "\lua_escape:e { \l__LuaCJK_punct_style_tl }" ,
                \__LuaCJK_bool_js:N \l__LuaCJK_punct_kinsoku_tl ,
                \__LuaCJK_bool_js:N \l__LuaCJK_cj_strict_tl ,
                \__LuaCJK_bool_js:N \l__LuaCJK_latin_punct_tl ,
                \__LuaCJK_bool_js:N \l__LuaCJK_cjk_math_tl ,
                \__LuaCJK_width_js:Nn
                  \l__LuaCJK_punct_width_given_bool \l__LuaCJK_punct_width_dim ,
                \__LuaCJK_width_js:Nn
                  \l__LuaCJK_punct_bound_width_given_bool
                  \l__LuaCJK_punct_bound_width_dim ,
                \__LuaCJK_bool_js:N \l__LuaCJK_check_single_tl ,
                \int_use:N \l__LuaCJK_widow_penalty_int ,
                \bool_if:NTF \l__LuaCJK_break_between_puncts_bool
                  { true } { false } ,
                \bool_if:NTF \l__LuaCJK_auto_fallback_bool
                  { true } { false } ,
                "\lua_escape:e { \l__LuaCJK_blocks_off_clist }"
```

```
                )
            }
        }
    }
```

(*End of definition for* `\__LuaCJK_settings_push:`.)

`\LuaCJKdisable` `\LuaCJKenable` Set the node attribute that makes the Lua callbacks skip the adjustments. The attribute follows the TEX grouping. The value ends with `\scan_stop`: TEX reads a number until a token cannot continue it, and the body of the command is followed by whatever the document writes next, so a digit there continues the value instead of being set. listings calls `\LuaCJKdisable` from its `OutputBox` hook, where a word block follows the command at once, so with `columns = fullflexible` the first number of a line joined the value and disappeared from the page.

```
\NewDocumentCommand \LuaCJKdisable { }
  { \g__LuaCJK_disabled_attr = 1 \scan_stop: }
\NewDocumentCommand \LuaCJKenable { }
  { \g__LuaCJK_disabled_attr = 0 \scan_stop: }
```

(*End of definition for* `\LuaCJKdisable` *and* `\LuaCJKenable`. *These functions are documented on page* 41.)

`\LuaCJKShipoutHook` Run code just before every page is shipped out. This is the place for work that must see the finished page; xeCJK calls a macro at the same moment, so the name is kept, but the code goes into the `shipout/before` hook and therefore follows its rules.

```
\NewDocumentCommand \LuaCJKShipoutHook { m }
  { \hook_gput_code:nnn { shipout / before } { LuaCJK } {#1} }
```

(*End of definition for* `\LuaCJKShipoutHook`. *This function is documented on page* 41.)

`\g__LuaCJK_nobreak_id_int` The identifier of the next break-prohibited span. A span takes one when its `\LuaCJKnobreak` starts outside every other span, so a call inside one joins the span already in force. The value 1 marks the span of `\LuaCJKVerbAddon`, which an argument inside it joins as well, so the first argument of a document carries 2.

```
\int_new:N \g__LuaCJK_nobreak_id_int
\int_gset:Nn \g__LuaCJK_nobreak_id_int { 1 }
```

(*End of definition for* `\g__LuaCJK_nobreak_id_int`.)

`\LuaCJKnobreak` Keep the argument on one line. The nodes of the argument carry the identifier of the span that encloses them, and a break is forbidden only where the two identifiers agree, so a break inside the argument is forbidden, and so is a break at a

glue the document writes there, whether it is a \quad, an \hspace, or an \hfill. The boundary between the argument and the text around it carries no mark or carries another span, so it keeps the break the text offers; two arguments written one after the other are two spans for the same reason. A call inside another one joins the span already in force, which is what keeps the whole content of an outer argument on one line. The identifier ends with \scan_stop for the reason \LuaCJKdisable gives: the argument follows the assignment at once, so a digit at the start of it would continue the value instead of being set.

```
\NewDocumentCommand \LuaCJKnobreak { +m }
  {
    \group_begin:
      \int_compare:nNnT \g__LuaCJK_nobreak_attr < { 1 }
        {
          \int_gincr:N \g__LuaCJK_nobreak_id_int
          \g__LuaCJK_nobreak_attr = \int_use:N \g__LuaCJK_nobreak_id_int \scan_stop:
        }
      #1
    \group_end:
  }
```

(*End of definition for* \LuaCJKnobreak. *This function is documented on page 41.*)

\LuaCJKVerbAddon
\LuaCJKOffVerbAddon

Align CJK text in a monospaced context: one CJK character plus the glue that follows it should take the width of two spaces. The gap therefore is twice the space of the current font minus the width of one CJK character. The width of the character is measured after switching to the CJK family in a group, because the current font is the monospaced one. A proportional font has no fixed space to align to, so the command adds nothing there. The glue is inserted by the spacing pass, and that pass leaves a suspended node alone, so the addon turns the adjustments back on inside its group. Without that it computes a gap that nothing inserts, which is what used to happen in a verbatim-like environment, whose begin hook suspends the material before the formatcom option of fancyvrb runs. The suspension the addon found is remembered, and \LuaCJKOffVerbAddon puts it back, so the material is left alone again once the addon is cancelled. The width of two spaces, the width of one CJK character, the gap between them, the saved glues, whether the addon is on, the suspension in force when it was switched on, and the box used to measure the character.

```
\dim_new:N \l__LuaCJK_verb_space_dim
\dim_new:N \l__LuaCJK_verb_cjk_dim
\dim_new:N \l__LuaCJK_verb_gap_dim
\skip_new:N \l__LuaCJK_verb_saved_ccglue_skip
\skip_new:N \l__LuaCJK_verb_saved_ecglue_skip
```

\l__LuaCJK_verb_space_dim
\l__LuaCJK_verb_cjk_dim
\l__LuaCJK_verb_gap_dim
\l__LuaCJK_verb_saved_ccglue_skip
\l__LuaCJK_verb_saved_ecglue_skip
\l__LuaCJK_verb_saved_ccglue_touched_bool
\l__LuaCJK_verb_saved_ecglue_touched_bool
\l__LuaCJK_verb_addon_bool
\l__LuaCJK_verb_saved_disabled_int
\l__LuaCJK_verb_box

```
\bool_new:N \l__LuaCJK_verb_saved_ccglue_touched_bool
\bool_new:N \l__LuaCJK_verb_saved_ecglue_touched_bool
\bool_new:N \l__LuaCJK_verb_addon_bool
\int_new:N \l__LuaCJK_verb_saved_disabled_int
\box_new:N \l__LuaCJK_verb_box

\msg_new:nnn { LuaCJK } { verb-addon-narrow }
  {
    The ~ monospaced ~ font ~ is ~ too ~ narrow ~ for ~ one ~ CJK ~
    character.\\
    `\token_to_str:N \LuaCJKVerbAddon' ~ adds ~ no ~ space.
  }
\cs_new_protected:Npn \__LuaCJK_verb_addon_compute:
  {
    \dim_set:Nn \l__LuaCJK_verb_space_dim
      { 2 \tex_fontdimen:D 2 \tex_font:D }
    \dim_compare:nNnTF
      { \tex_fontdimen:D 2 \tex_font:D }
      = { \tex_fontcharwd:D \tex_font:D 77 }
      {
        \dim_set_eq:NN \l__LuaCJK_verb_cjk_dim \l__LuaCJK_verb_space_dim
        \group_begin:
          \prop_get:NVNT \g__LuaCJK_family_prop \l__LuaCJK_family_tl \l__LuaCJK_tmpa_t
            {
              \exp_args:NV \__LuaCJK_family_font:n \l__LuaCJK_tmpa_tl
              \selectfont
              \hbox_set:Nn \l__LuaCJK_verb_box { 一 }
              \dim_gset:Nn \l__LuaCJK_verb_cjk_dim
                { \box_wd:N \l__LuaCJK_verb_box }
            }
        \group_end:
        \dim_set:Nn \l__LuaCJK_verb_gap_dim
          { \l__LuaCJK_verb_space_dim - \l__LuaCJK_verb_cjk_dim }
        \dim_compare:nNnTF { \l__LuaCJK_verb_gap_dim } < { 0pt }
          {
            \msg_warning:nn { LuaCJK } { verb-addon-narrow }
            \LuaCJKsetup { CJKglue = 0pt, CJKecglue = 0pt }
          }
          {
            \LuaCJKsetup
              {
                CJKglue   = \dim_use:N \l__LuaCJK_verb_gap_dim ,
                CJKecglue = \dim_eval:n { \l__LuaCJK_verb_gap_dim / 2 }
              }
          }
```

```
      }
      { \LuaCJKsetup { CJKglue = 0pt, CJKecglue = 0pt } }
  }
\NewDocumentCommand \LuaCJKVerbAddon { }
  {
    \int_compare:nNnF \tex_currentgrouplevel:D = \c_zero_int
      {
        \skip_set_eq:NN \l__LuaCJK_verb_saved_ccglue_skip
          \l__LuaCJK_ccglue_skip
        \skip_set_eq:NN \l__LuaCJK_verb_saved_ecglue_skip
          \l__LuaCJK_ecglue_skip
        \bool_set_eq:NN \l__LuaCJK_verb_saved_ccglue_touched_bool
          \l__LuaCJK_ccglue_touched_bool
        \bool_set_eq:NN \l__LuaCJK_verb_saved_ecglue_touched_bool
          \l__LuaCJK_ecglue_touched_bool
        \int_set:Nn \l__LuaCJK_verb_saved_disabled_int
          { \g__LuaCJK_disabled_attr }
        \LuaCJKenable
        \__LuaCJK_verb_addon_compute:
        \g__LuaCJK_nobreak_attr = 1
        \bool_set_true:N \l__LuaCJK_verb_addon_bool
      }
  }
\NewDocumentCommand \LuaCJKOffVerbAddon { }
  {
    \bool_if:NT \l__LuaCJK_verb_addon_bool
      {
        \skip_set_eq:NN \l__LuaCJK_ccglue_skip
          \l__LuaCJK_verb_saved_ccglue_skip
        \skip_set_eq:NN \l__LuaCJK_ecglue_skip
          \l__LuaCJK_verb_saved_ecglue_skip
        \bool_set_eq:NN \l__LuaCJK_ccglue_touched_bool
          \l__LuaCJK_verb_saved_ccglue_touched_bool
        \bool_set_eq:NN \l__LuaCJK_ecglue_touched_bool
          \l__LuaCJK_verb_saved_ecglue_touched_bool
        \__LuaCJK_settings_push:
        \g__LuaCJK_nobreak_attr = 0
        \int_compare:nNnTF { \l__LuaCJK_verb_saved_disabled_int } > { 0 }
          { \LuaCJKdisable }
          { \LuaCJKenable }
        \bool_set_false:N \l__LuaCJK_verb_addon_bool
      }
  }
```

(*End of definition for* `\LuaCJKVerbAddon` *and others. These functions are documented on page* 42.)

__LuaCJK_verbatim@font_hook:

Verbatim-like material must usually be left alone. `\verb` opens its own group and selects `\verbatim@font` inside it, so extending that command covers `\verb` on its own. The hook consults the current `Verb` setting, so that `\LuaCJKsetup` can change the behavior after the package is loaded.

```
\cs_new_protected:Npn \__LuaCJK_verbatim@font_hook:
  {
    \tl_if_eq:VnTF \l__LuaCJK_verb_mode_tl { true }
      { \LuaCJKdisable }
      {
        \tl_if_eq:VnT \l__LuaCJK_verb_mode_tl { env }
          { \LuaCJKdisable }
      }
  }
\cs_if_exist:NT \verbatim@font
  {
    \cs_gset_protected:Npx \verbatim@font
      {
        \exp_not:o { \verbatim@font }
        \exp_not:N \__LuaCJK_verbatim@font_hook:
      }
  }
```

(*End of definition for* `\__LuaCJK_verbatim@font_hook:`.)

__LuaCJK_verbatim_block_disable:
__LuaCJK_fv_prep_hook:
__LuaCJK_fv_patch:

fancyvrb never selects `\verbatim@font`, and its formatting hook serves the inline commands and every block at once, so the hook waits for the package to be loaded and then applies the rule of the call it serves: the rule of `\verb` while no block is open, and the rule of the environments, which only `true` suspends, when one is. A block is an environment of the list below, or a command that sets saved verbatim material, and both mark themselves in their own group. The inline commands cannot mark themselves: a token placed after `\Verb` is scanned as part of its text, because the scanner reads to the end of the line, and `\FVC@Verb` closes its group from inside the box it sets, so neither can be wrapped. The bool is therefore false by default and set by the block side.

```
\cs_new_protected:Npn \__LuaCJK_verbatim_block_disable:
  {
    \tl_if_eq:VnT \l__LuaCJK_verb_mode_tl { true }
      { \LuaCJKdisable }
  }
\cs_new_protected:Npn \__LuaCJK_fv_prep_hook:
  {
    \bool_if:NTF \l__LuaCJK_verbatim_block_bool
      { \__LuaCJK_verbatim_block_disable: }
```

```
      { \__LuaCJK_verbatim@font_hook: }
  }
\cs_new_protected:Npn \__LuaCJK_fv_patch:
  {
    \cs_if_exist:NT \FV@FormattingPrep
      {
        \cs_gset_protected:Npx \FV@FormattingPrep
          {
            \exp_not:N \__LuaCJK_fv_prep_hook:
            \exp_not:o { \FV@FormattingPrep }
          }
      }
    \cs_if_exist:NT \FV@UseVerbatim
      {
        \cs_gset_eq:NN \__LuaCJK_fv_use_verbatim: \FV@UseVerbatim
        \cs_gset_protected:Npn \FV@UseVerbatim ##1
          {
            \group_begin:
              \bool_set_true:N \l__LuaCJK_verbatim_block_bool
              \__LuaCJK_fv_use_verbatim: {##1}
            \group_end:
          }
      }
  }
\hook_gput_code:nnn { package / fancyvrb / after } { LuaCJK }
  { \__LuaCJK_fv_patch: }
\__LuaCJK_fv_patch:
```

(*End of definition for* `\__LuaCJK_verbatim_block_disable:`, `\__LuaCJK_fv_prep_hook:`, *and* `\__LuaCJK_fv_patch:`.)

`\__LuaCJK_verbatim_env_begin:`
`\__LuaCJK_verbatim_env_end:`

Environments that do not go through `\verbatim@font` need their own hooks. Only the `true` setting suspends the adjustments here: `env` and `env+` keep them, which is what keeps code columns aligned when CJK text is used inside the code. The begin hook marks the block for `\__LuaCJK_fv_prep_hook:` as well, so that the formatting hook of fancyvrb applies the rule of the environment rather than the rule of `\verb` inside it.

```
\cs_new_protected:Npn \__LuaCJK_verbatim_env_begin:
  {
    \bool_set_true:N \l__LuaCJK_verbatim_block_bool
    \__LuaCJK_verbatim_block_disable:
  }
\cs_new_protected:Npn \__LuaCJK_verbatim_env_end:
  {
    \tl_if_eq:VnT \l__LuaCJK_verb_mode_tl { true }
```

```
      { \LuaCJKenable }
  }
\clist_map_inline:nn
  {
    verbatim,  Verbatim,  BVerbatim,
    verbatim*, Verbatim*, LVerbatim, lstlisting
  }
  {
    \hook_gput_code:nnn { env / #1 / begin } { LuaCJK }
      { \__LuaCJK_verbatim_env_begin: }
    \hook_gput_code:nnn { env / #1 / end   } { LuaCJK }
      { \__LuaCJK_verbatim_env_end:   }
  }
```

(*End of definition for* `\__LuaCJK_verbatim_env_begin:` *and* `\__LuaCJK_verbatim_env_end:`.)

A.3 Fonts

`\__LuaCJK_family_font:n`

Choose the font of a CJK family without selecting it. fontspec declares a family under the Unicode encoding, so a lookup made in another encoding finds no such shape, warns that the shape is undefined, and falls back to the Latin font: with fontenc's `T1` option the CJK text would be set in the T1 text font, and inside pifont's `U` encoding in the symbol font. Every place that resolves the font identifier of a family goes through this command, and `\selectfont` stays with the caller, which also sets the series, the shape, and the size where it needs them.

```
\cs_new_protected:Npn \__LuaCJK_family_font:n #1
  { \fontencoding { TU } \fontfamily {#1} }
```

(*End of definition for* `\__LuaCJK_family_font:n`.)

`\__LuaCJK_set_family:nnn`

Define a family through fontspec and remember the resulting family name. The family is only registered; the switch happens in `\__LuaCJK_switch_family:n`.

`\g__LuaCJK_family_prop`
`\l__LuaCJK_family_tl`

The registered families, and the family name currently selected.

```
\prop_new:N \g__LuaCJK_family_prop
\tl_new:N \l__LuaCJK_family_tl
\tl_set:Nn \l__LuaCJK_family_tl { zhsong }
```

`\l__LuaCJK_fontspec_tl`
`\l__LuaCJK_family_name_tl`
`\l__LuaCJK_new_family_tl`
`\l__LuaCJK_tmpa_tl`

Scratch variables holding the generated fontspec family name, the family name looked up in the property list, a temporary value, and the name of the one-off family `\CJKfontspec` is creating. That name is a variable of its own because it is held across the registration of the family, which runs `\selectfont` and therefore `\__LuaCJK_update_font:`: a scratch variable would be spent by the time the command selects the family.

```
\tl_new:N \l__LuaCJK_fontspec_tl
```

```
\tl_new:N \l__LuaCJK_family_name_tl
\tl_new:N \l__LuaCJK_new_family_tl
\tl_new:N \l__LuaCJK_tmpa_tl
\tl_new:N \l__LuaCJK_tmpb_tl
\int_new:N \l__LuaCJK_tmpa_int
\tl_new:N \l__LuaCJK_registered_name_tl
```

The font name and the explicit options of each family, kept so that `\addCJKfontfeatures` can register the family again, and the features prepended to every family. Whether one family's options already mention `AutoFakeBold`, `AutoFakeSlant`, `BoldFont`, or `ItalicFont`.

\g__LuaCJK_font_name_prop
\g__LuaCJK_font_feature_prop
\l__LuaCJK_default_feature_tl

```
\bool_new:N \l__LuaCJK_autofake_bold_given_bool
\bool_new:N \l__LuaCJK_autofake_slant_given_bool
\bool_new:N \l__LuaCJK_bold_font_given_bool
\bool_new:N \l__LuaCJK_italic_font_given_bool
\prop_new:N \g__LuaCJK_font_name_prop
\prop_new:N \g__LuaCJK_font_feature_prop
\tl_new:N  \l__LuaCJK_default_feature_tl
\tl_set:Nn \l__LuaCJK_default_feature_tl { Script = CJK }
\tl_new:N  \l__LuaCJK_feature_tl
\cs_generate_variant:Nn \fontspec_gset_family:Nnn { NVV , NVn }
```

\l__LuaCJK_autofake_bold_given_bool
\l__LuaCJK_autofake_slant_given_bool
\l__LuaCJK_bold_font_given_bool
\l__LuaCJK_italic_font_given_bool

```
\cs_new_protected:Npn \__LuaCJK_set_family:nnn #1#2#3
  {
    \__LuaCJK_build_features:n {#2}
    \prop_gput:NnV \g__LuaCJK_font_feature_prop {#1} \l__LuaCJK_feature_tl
    \prop_gput:Nnn \g__LuaCJK_font_name_prop {#1} {#3}
    \__LuaCJK_register_family:n {#1}
  }
```

Combine the default features, the options given for this family, and the automatic fake bold and slant, unless the family sets them itself.

__LuaCJK_build_features:n

```
\cs_new_protected:Npn \__LuaCJK_note_feature_key:nn #1#2
  { \__LuaCJK_note_feature_key:n {#1} }
\cs_new_protected:Npn \__LuaCJK_note_feature_key:n #1
  {
    \str_case:en { \tl_trim_spaces:n {#1} }
      {
        { AutoFakeBold  } { \bool_set_true:N \l__LuaCJK_autofake_bold_given_bool  }
        { AutoFakeSlant } { \bool_set_true:N \l__LuaCJK_autofake_slant_given_bool }
        { BoldFont      } { \bool_set_true:N \l__LuaCJK_bold_font_given_bool      }
        { ItalicFont    } { \bool_set_true:N \l__LuaCJK_italic_font_given_bool    }
      }
  }
```

```
\cs_new_protected:Npn \__LuaCJK_build_features:n #1
  {
    \clist_map_inline:nn
      { autofake_bold, autofake_slant, bold_font, italic_font }
      { \exp_args:Nc \bool_set_false:N { l__LuaCJK_##1_given_bool } }
    \keyval_parse:NNn
      \__LuaCJK_note_feature_key:n \__LuaCJK_note_feature_key:nn {#1}
    \tl_set_eq:NN \l__LuaCJK_feature_tl \l__LuaCJK_default_feature_tl
    \tl_if_blank:nF {#1}
      { \tl_put_right:Nn \l__LuaCJK_feature_tl { , ~ #1 } }
    \tl_if_empty:NF \g__LuaCJK_autofake_bold_tl
      {
        \bool_lazy_or:nnF
          { \l__LuaCJK_autofake_bold_given_bool }
          { \l__LuaCJK_bold_font_given_bool     }
          {
            \tl_put_right:Nx \l__LuaCJK_feature_tl
              { , ~ AutoFakeBold = \exp_not:V \g__LuaCJK_autofake_bold_tl }
          }
      }
    \tl_if_empty:NF \g__LuaCJK_autofake_slant_tl
      {
        \bool_lazy_or:nnF
          { \l__LuaCJK_autofake_slant_given_bool }
          { \l__LuaCJK_italic_font_given_bool    }
          {
            \tl_put_right:Nx \l__LuaCJK_feature_tl
              { , ~ AutoFakeSlant = \exp_not:V \g__LuaCJK_autofake_slant_tl }
          }
      }
  }
```

Build the fontspec family from the stored font name and features, and re-select it if it is the current one.

`__LuaCJK_register_family:n`

```
\cs_new_protected:Npn \__LuaCJK_register_family:n #1
  {
    \prop_get:NnNF \g__LuaCJK_font_name_prop {#1} \l__LuaCJK_registered_name_tl
      { \tl_clear:N \l__LuaCJK_registered_name_tl }
    \prop_get:NnNF \g__LuaCJK_font_feature_prop {#1} \l__LuaCJK_feature_tl
      { \tl_clear:N \l__LuaCJK_feature_tl }
    \prop_get:NnNF \g__LuaCJK_shift_decl_prop {#1} \l__LuaCJK_shift_dim_tl
      { \tl_clear:N \l__LuaCJK_shift_dim_tl }
    \group_begin:
      \fontspec_gset_family:NVV \l__LuaCJK_fontspec_tl
        \l__LuaCJK_feature_tl \l__LuaCJK_registered_name_tl
```

```
      \prop_gput:NnV \g__LuaCJK_family_prop {#1} \l__LuaCJK_fontspec_tl
      \tl_if_empty:NF \l__LuaCJK_shift_dim_tl
        {
          \tl_set:Ne \l__LuaCJK_shift_key_tl { #1 | \l__LuaCJK_fontspec_tl }
          \prop_gput:NVV \g__LuaCJK_font_shift_prop
            \l__LuaCJK_shift_key_tl \l__LuaCJK_shift_dim_tl
        }
    \group_end:
    \prop_if_in:NVT \g__LuaCJK_family_prop \l__LuaCJK_family_tl
      { \selectfont }
  }
```

__LuaCJK_push_fallback:
__LuaCJK_refresh_fallbacks:

Resolve the fallback families of the current CJK family to font identifiers at the current size, series, and shape, and hand the list to Lua together with the identifier of the family itself. A font identifier belongs to one instance of a font, so the list has to be resolved again whenever a font is selected: the identifier of the bold or enlarged main font is not the one the upright size was resolved at. The CJK family whose list was last handed to Lua for each font identifier, which makes a selection that resolves to a known identifier cost one property lookup, and whether the Lua module has been loaded. The guard of `\__LuaCJK_update_font:` is raised around the resolution, so the temporary selections it makes do not change the identifier they are resolved for. The math chain is refreshed in the same pass, so a fallback declared after the document started, or an `AutoFallBack` that is switched on, reaches the next formula even when no font is selected before it.

```
\prop_new:N \g__LuaCJK_fallback_pushed_prop
\bool_new:N \g__LuaCJK_lua_loaded_bool
\cs_new_protected:Npn \__LuaCJK_push_fallback:
  {
    \int_compare:nNnT { \g__LuaCJK_fontid_int } > { 0 }
      {
        \tl_set:Ne \l__LuaCJK_fontid_tl { \int_use:N \g__LuaCJK_fontid_int }
        \prop_get:NVNT \g__LuaCJK_fallback_prop \l__LuaCJK_family_tl \l__LuaCJK_tmpa_t
          {
            \prop_get:NVNF \g__LuaCJK_fallback_pushed_prop
              \l__LuaCJK_fontid_tl \l__LuaCJK_tmpb_tl
              { \tl_clear:N \l__LuaCJK_tmpb_tl }
            \tl_if_eq:NNF \l__LuaCJK_tmpb_tl \l__LuaCJK_family_tl
              { \exp_args:NV \__LuaCJK_push_fallback_aux:n \l__LuaCJK_tmpa_tl }
          }
      }
  }
\cs_new_protected:Npn \__LuaCJK_push_fallback_aux:n #1
```

```
  {
    \group_begin:
      \tl_clear:N \l__LuaCJK_fallback_ids_tl
      \clist_map_inline:nn {#1}
        {
          \__LuaCJK_family_font:n {##1} \selectfont
          \tl_set:Ne \l__LuaCJK_tmpb_tl
            { \int_eval:n { \tex_fontid:D \tex_font:D } }
          \tl_put_right:NV \l__LuaCJK_fallback_ids_tl \l__LuaCJK_tmpb_tl
          \tl_put_right:Nn \l__LuaCJK_fallback_ids_tl { , }
          \__LuaCJK_push_shift:nnn { \l__LuaCJK_tmpb_tl } { \l__LuaCJK_family_tl } {##
        }
      \bool_if:NT \g__LuaCJK_lua_loaded_bool
        {
          \tl_if_empty:NF \l__LuaCJK_fallback_ids_tl
            {
              \lua_now:e
                {
                  LuaCJK.set_fallback
                    ( \int_use:N \g__LuaCJK_fontid_int ,
                      { \l__LuaCJK_fallback_ids_tl } )
                }
              \prop_gput:NVV \g__LuaCJK_fallback_pushed_prop
                \l__LuaCJK_fontid_tl \l__LuaCJK_family_tl
            }
        }
    \group_end:
  }
\cs_new_protected:Npn \__LuaCJK_refresh_fallbacks:
  {
    \prop_gclear:N \g__LuaCJK_fallback_pushed_prop
    \bool_set_true:N \l__LuaCJK_selectfont_bool
    \__LuaCJK_push_fallback:
    \__LuaCJK_math_font_update:
    \bool_set_false:N \l__LuaCJK_selectfont_bool
  }
```

__LuaCJK_shift_value:nn
__LuaCJK_push_shift:nnn

Compute the vertical offset `Baseline` asks for, and hand the offset of a fallback font to Lua. `\__LuaCJK_shift_value:nn` reads the value of one declaration, keyed by the declaring family and the family fontspec returned for it, and evaluates it at the size in force, so a value in `em` follows the text. It stores scaled points in `\g__LuaCJK_shift_value_tl`, which is global because the value is computed inside the group that selects the font and read after that group: the text and the punctuation marks take their offset from an attribute, and an attribute assign-

ment inside the group would be taken back with it. The attribute is what keeps the offset with the characters a family sets: a font identifier is shared by every declaration that supplies the same font, so a family selected inside a group would otherwise move the whole paragraph. `\__LuaCJK_push_shift:nnn` hands the offset of a fallback font to Lua, keyed by its font identifier, because a fallback is chosen when the text is shaped and only its identifier says which one applies. One font instance carries one offset, so a value is handed over again only when it differs from the one Lua holds, which is what lets two fallback lists that name the same font take turns. A change also drops the record of which fallback list was handed over for a font identifier, because a list that shares the instance has to be resolved again for its own value. The `Baseline` value of each declared font, keyed by the declaring family and the family fontspec returned for it; the value Lua holds for each font identifier; and scratch variables for the identifier, the offset, and the two values that decide whether it has to be handed over, the offset being evaluated at the size in force so that a value in `em` follows the text.

```
\cs_new_protected:Npn \__LuaCJK_shift_value:nn #1#2
  {
    \tl_set:Ne \l__LuaCJK_shift_key_tl { #1 | #2 }
    \prop_get:NVNF \g__LuaCJK_font_shift_prop \l__LuaCJK_shift_key_tl
      \l__LuaCJK_shift_dim_tl
      { \tl_clear:N \l__LuaCJK_shift_dim_tl }
    \tl_if_empty:NTF \l__LuaCJK_shift_dim_tl
      { \tl_gset:Nn \g__LuaCJK_shift_value_tl { 0 } }
      {
        \tl_gset:Ne \g__LuaCJK_shift_value_tl
          {
            \dim_to_decimal_in_sp:n
              { \dim_eval:n { \l__LuaCJK_shift_dim_tl } }
          }
      }
  }
\cs_new_protected:Npn \__LuaCJK_push_shift:nnn #1#2#3
  {
    \tl_set:Ne \l__LuaCJK_shift_fid_tl {#1}
    \__LuaCJK_shift_value:nn {#2} {#3}
    \prop_get:NVNF \g__LuaCJK_shift_pushed_prop \l__LuaCJK_shift_fid_tl
      \l__LuaCJK_shift_old_tl
      { \tl_set:Nn \l__LuaCJK_shift_old_tl { 0 } }
    \tl_if_eq:NNF \g__LuaCJK_shift_value_tl \l__LuaCJK_shift_old_tl
      {
        \prop_gput:NVV \g__LuaCJK_shift_pushed_prop
          \l__LuaCJK_shift_fid_tl \g__LuaCJK_shift_value_tl
        \prop_gclear:N \g__LuaCJK_fallback_pushed_prop
```

```
        \lua_now:e
          {
            LuaCJK.set_yoffset
              ( \l__LuaCJK_shift_fid_tl , \g__LuaCJK_shift_value_tl )
          }
      }
  }
```

(*End of definition for* `\__LuaCJK_set_family:nnn` *and others.*)

`\__LuaCJK_switch_family:n` Select a registered family and warn once about an unknown name.

```
\cs_new_protected:Npn \__LuaCJK_switch_family:n #1
  {
    \prop_if_in:NnTF \g__LuaCJK_family_prop {#1}
      {
        \tl_set:Nn \l__LuaCJK_family_tl {#1}
        \selectfont
      }
      {
        \seq_if_in:NnF \g__LuaCJK_unknown_family_seq {#1}
          {
            \seq_gput_right:Nn \g__LuaCJK_unknown_family_seq {#1}
            \msg_warning:nnn { LuaCJK } { family-unknown } {#1}
          }
      }
  }
\seq_new:N \g__LuaCJK_unknown_family_seq
\msg_new:nnn { LuaCJK } { family-unknown }
  { Unknown ~ CJK ~ family ~ `#1' ~ is ~ being ~ ignored. }
```

(*End of definition for* `\__LuaCJK_switch_family:n`.)

`\__LuaCJK_reset_family:` Reset the CJK family to `\CJKfamilydefault`. The name is expanded first, because the default is written as `\CJKrmdefault` until the document chooses a family of its own. A name that no family carries is passed over in silence: `\normalfont` also runs when the document starts, and the hook at `\begin{document}` runs the same reset once the default has been settled, so a document that declared no font at all has nothing to switch to.

```
\cs_new_protected:Npn \__LuaCJK_reset_family:
  {
    \tl_set:Ne \l__LuaCJK_tmpa_tl { \CJKfamilydefault }
    \prop_if_in:NVT \g__LuaCJK_family_prop \l__LuaCJK_tmpa_tl
      { \exp_args:NV \__LuaCJK_switch_family:n \l__LuaCJK_tmpa_tl }
  }
```

(*End of definition for* `\__LuaCJK_reset_family:`.)

`\__LuaCJK_ensure_family_default:` Check that the family `\CJKfamilydefault` names was declared. A document that declares only some of its families sets its main text in no CJK font at all, and says nothing about it, so the family is replaced by `\CJKrmdefault` when that one is declared and by the first declared family otherwise. A document that declared nothing gets a warning of its own, because `LoadLXGW` is off and there is no family to fall back to.

```
\cs_new_protected:Npn \__LuaCJK_ensure_family_default:
  {
    \tl_set:Ne \l__LuaCJK_tmpa_tl { \CJKfamilydefault }
    \prop_if_in:NVF \g__LuaCJK_family_prop \l__LuaCJK_tmpa_tl
      {
        \prop_if_empty:NTF \g__LuaCJK_family_prop
          { \msg_warning:nn { LuaCJK } { no-cjk-font } }
          {
            \tl_set:Ne \l__LuaCJK_tmpb_tl { \CJKrmdefault }
            \prop_if_in:NVF \g__LuaCJK_family_prop \l__LuaCJK_tmpb_tl
              {
                \prop_map_inline:Nn \g__LuaCJK_family_prop
                  { \tl_set:Nn \l__LuaCJK_tmpb_tl {##1} \prop_map_break: }
              }
            \msg_warning:nnee { LuaCJK } { family-default-undefined }
              { \l__LuaCJK_tmpa_tl } { \l__LuaCJK_tmpb_tl }
            \tl_gset_eq:NN \CJKfamilydefault \l__LuaCJK_tmpb_tl
            \exp_args:NV \__LuaCJK_switch_family:n \l__LuaCJK_tmpb_tl
          }
      }
  }
\msg_new:nnn { LuaCJK } { no-cjk-font }
  {
    No ~ CJK ~ font ~ is ~ set, ~ and ~ LoadLXGW ~ is ~ off.\\
    Declare ~ one ~ with ~ \token_to_str:N \setCJKmainfont.
  }
\msg_new:nnn { LuaCJK } { family-default-undefined }
  {
    The ~ default ~ CJK ~ family ~ `#1' ~ is ~ not ~ declared;\\
    using ~ `#2' ~ instead.
  }
```

(*End of definition for* `\__LuaCJK_ensure_family_default:`.)

`__LuaCJK_math_font_update:` Register the CJK family used by the current settings as a math family and hand the fallback chain of that family to Lua, or clear the math font attribute when

CJKmath is off or the family is not known. __LuaCJK_update_font: runs it at every font selection, and __LuaCJK_math_version_refresh: runs it again when the math version changes, which is why it stands on its own.

```
\cs_new_protected:Npn \__LuaCJK_math_font_update:
  {
    \tl_if_eq:NnTF \l__LuaCJK_cjk_math_tl { 1 }
      {
        \tl_if_empty:NTF \l__LuaCJK_math_family_tl
          { \tl_set_eq:NN \l__LuaCJK_tmpa_tl \l__LuaCJK_family_tl }
          { \tl_set_eq:NN \l__LuaCJK_tmpa_tl \l__LuaCJK_math_family_tl }
        \prop_get:NVNTF \g__LuaCJK_family_prop \l__LuaCJK_tmpa_tl \l__LuaCJK_tmpb_tl
          {
            \exp_args:NV \__LuaCJK_register_math_family:n \l__LuaCJK_tmpb_tl
            \exp_args:NV \__LuaCJK_math_fallbacks_push:n \l__LuaCJK_tmpa_tl
          }
          { \g__LuaCJK_math_font_attr = \c_zero_int }
      }
      { \g__LuaCJK_math_font_attr = \c_zero_int }
  }
```

(*End of definition for* __LuaCJK_math_font_update:.)

__LuaCJK_update_font:

On every \selectfont, resolve the current family, its fallback families, the math family, and the PunctFamily family to font identifiers and store them in the attributes, with the Baseline offset of the current family and of the punctuation family; each fallback font's offset goes to Lua with its identifier. The punctuation family takes a selection of its own, because nothing reaches it through \CJKfamily; without the offset computed here its marks would stand on the baseline of a font they are not set in. The offset is computed inside the group that selects the font, so that em follows the family's own size, and assigned to the attribute after the group, which would otherwise take the assignment back with it. A guard against re-entering the update while \selectfont runs, and the font identifier of the current CJK family.

\l__LuaCJK_selectfont_bool
\g__LuaCJK_fontid_int

```
\bool_new:N \l__LuaCJK_selectfont_bool
\int_new:N \g__LuaCJK_fontid_int
```

The math family registered by \setCJKmathfont, the family chosen by PunctFamily, and the font identifier resolved for the latter.

\l__LuaCJK_math_family_tl
\l__LuaCJK_punct_family_tl
\g__LuaCJK_punct_fontid_int

```
\tl_new:N \l__LuaCJK_math_family_tl
\tl_new:N \l__LuaCJK_punct_family_tl
\int_new:N \g__LuaCJK_punct_fontid_int
```

The last math family number handed out, the map from a CJK family name to its number, and the upper bound. A CJK character in a math list needs a math

\g__LuaCJK_math_family_int
\g__LuaCJK_math_family_prop
\c__LuaCJK_math_family_max_int

family of its own, because the family is what selects the font *and* its size for the three math styles. LaTeX 2_ε allocates the 256 families that LuaTeX allows from zero upwards, so the numbers start at 240; a document that needs more than sixteen numbers is out of luck and gets an error. A fallback font used in math takes a number of its own, and the families that declare it share that number.

```
\int_new:N \g__LuaCJK_math_family_int
\int_gset:Nn \g__LuaCJK_math_family_int { 239 }
\int_const:Nn \c__LuaCJK_math_family_max_int { 255 }
\prop_new:N \g__LuaCJK_math_family_prop
\cs_new_protected:Npn \__LuaCJK_update_font:
  {
    \bool_if:NF \l__LuaCJK_selectfont_bool
      {
        \bool_set_true:N \l__LuaCJK_selectfont_bool
        \prop_get:NVNTF \g__LuaCJK_family_prop \l__LuaCJK_family_tl \l__LuaCJK_family_
          {
            \group_begin:
              \exp_args:NV \__LuaCJK_family_font:n \l__LuaCJK_family_name_tl
              \selectfont
              \int_gset:Nn \g__LuaCJK_fontid_int
                { \tex_fontid:D \tex_font:D }
              \__LuaCJK_shift_value:nn
                { \l__LuaCJK_family_tl } { \l__LuaCJK_family_name_tl }
            \group_end:
            \g__LuaCJK_font_attr = \g__LuaCJK_fontid_int
            \g__LuaCJK_shift_attr = \g__LuaCJK_shift_value_tl
          }
          {
            \g__LuaCJK_font_attr = \c_zero_int
            \g__LuaCJK_shift_attr = \c_zero_int
            \tl_clear:N \l__LuaCJK_family_name_tl
          }
        \__LuaCJK_math_font_update:
        \__LuaCJK_push_fallback:
        \__LuaCJK_push_block_fonts:
        \tl_if_empty:NTF \l__LuaCJK_punct_family_tl
          {
            \g__LuaCJK_punct_font_attr = \c_zero_int
            \g__LuaCJK_punct_shift_attr = \c_zero_int
          }
          {
            \prop_get:NVNTF \g__LuaCJK_family_prop \l__LuaCJK_punct_family_tl
              \l__LuaCJK_tmpa_tl
              {
```

```
                \group_begin:
                  \exp_args:NV \__LuaCJK_family_font:n \l__LuaCJK_tmpa_tl
                  \selectfont
                  \int_gset:Nn \g__LuaCJK_punct_fontid_int
                    { \tex_fontid:D \tex_font:D }
                  \__LuaCJK_shift_value:nn
                    { \l__LuaCJK_punct_family_tl } { \l__LuaCJK_tmpa_tl }
                \group_end:
                \g__LuaCJK_punct_font_attr = \g__LuaCJK_punct_fontid_int
                \g__LuaCJK_punct_shift_attr = \g__LuaCJK_shift_value_tl
              }
              {
                \g__LuaCJK_punct_font_attr = \c_zero_int
                \g__LuaCJK_punct_shift_attr = \c_zero_int
                \__LuaCJK_warn_punct_family:
              }
          }
        \bool_set_false:N \l__LuaCJK_selectfont_bool
      }
  }
```

(*End of definition for* `\__LuaCJK_update_font:` *and others.*)

`\__LuaCJK_warn_punct_family:`

Report the family `PunctFamily` names when no declared family carries that name, so the fallback to the text font is not silent. Whether the document has started, and the names already reported. The warning waits for the start of the document, because the preamble may select a font before the family exists — `LoadLXGW` declares the LXGW faces in the hook at `\begin{document}` — and the hook that sets the flag runs after `\AtBeginDocument` material, so a family declared there counts as declared; it tests the family once more, because a document that selects no font after that would never reach the check in `\__LuaCJK_update_font:`. It is said once per name, because the update runs at every font selection.

```
\bool_new:N \g__LuaCJK_document_started_bool
\seq_new:N \g__LuaCJK_punct_family_unknown_seq
\hook_gput_code:nnn { begindocument / end } { LuaCJK }
  {
    \bool_gset_true:N \g__LuaCJK_document_started_bool
    \__LuaCJK_warn_punct_family:
  }

\msg_new:nnn { LuaCJK } { punct-family-unknown }
  {
    The ~ CJK ~ family ~ `#1' ~ that ~ the ~ PunctFamily ~ key ~ names ~
```

```
    is ~ not ~ declared;\\
    the ~ punctuation ~ follows ~ the ~ text ~ font.
  }
\cs_new_protected:Npn \__LuaCJK_warn_punct_family:
  {
    \bool_if:NT \g__LuaCJK_document_started_bool
      {
        \tl_if_empty:NF \l__LuaCJK_punct_family_tl
          {
            \prop_if_in:NVF \g__LuaCJK_family_prop \l__LuaCJK_punct_family_tl
              {
                \exp_args:NNV \seq_if_in:NnF
                  \g__LuaCJK_punct_family_unknown_seq \l__LuaCJK_punct_family_tl
                  {
                    \seq_gput_right:NV \g__LuaCJK_punct_family_unknown_seq
                      \l__LuaCJK_punct_family_tl
                    \msg_warning:nne { LuaCJK }
                      { punct-family-unknown } { \l__LuaCJK_punct_family_tl }
                  }
              }
          }
      }
  }
```

(*End of definition for* `\__LuaCJK_warn_punct_family:`, `\g__LuaCJK_document_started_bool`, *and* `\g__LuaCJK_punct_family_unknown_seq`.)

`\__LuaCJK_math_family:n` Give a CJK family or one of its fallback fonts a math family number. A number is handed out once per family name, so two CJK families that declare the same fallback font share its number.

```
\msg_new:nnn { LuaCJK } { math-family-exhausted }
  {
    Too ~ many ~ CJK ~ and ~ fallback ~ families ~ for ~ the ~ math ~
    families ~ 240--255.
  }
\cs_new_protected:Npn \__LuaCJK_math_family:n #1
  {
    \prop_if_in:NnF \g__LuaCJK_math_family_prop {#1}
      {
        \int_gincr:N \g__LuaCJK_math_family_int
        \int_compare:nNnT \g__LuaCJK_math_family_int > \c__LuaCJK_math_family_max_int
          { \msg_error:nn { LuaCJK } { math-family-exhausted } }
        \prop_gput:NnV \g__LuaCJK_math_family_prop {#1}
          \g__LuaCJK_math_family_int
      }
```

```
  }
```

(*End of definition for* `\__LuaCJK_math_family:n`.)

`\__LuaCJK_math_size_set:` `\l__LuaCJK_math_size_tl`

The text size a math family is built at: the size LaTeX 2_ε has installed the math fonts for, which is `\glb@currsize`, and `\f@size` before the document starts. The two differ inside amsmath's `\text`, whose `\mathchoice` sets `\f@size` to the script and scriptscript size for the branches it discards, and a math family built there would keep that size for every character after the `\text`.

```
\tl_new:N \l__LuaCJK_math_size_tl
\cs_new_protected:Npn \__LuaCJK_math_size_set:
  {
    \tl_set_eq:NN \l__LuaCJK_math_size_tl \f@size
    \cs_if_exist:NT \glb@currsize
      {
        \tl_if_empty:NF \glb@currsize
          { \tl_set_eq:NN \l__LuaCJK_math_size_tl \glb@currsize }
      }
  }
```

(*End of definition for* `\__LuaCJK_math_size_set:` *and* `\l__LuaCJK_math_size_tl`.)

`\__LuaCJK_math_family_fonts:nn`

Fill the three style fonts of one math family number from a CJK family, at the text size of the formula and at seven and five tenths of it, which is the ratio of the standard LaTeX size tables. The family is selected inside a group, so the caller's font is not disturbed, and the assignments are global because a math font belongs to a family, not to a group. The series and the shape come from the math version, not from the text around the formula: the family is selected at `\mddefault` and `\shapedefault`, and, in the `bold` math version, at `\bfdefault` as well when the family declares that shape. The regular face is what loads the shape list the test reads, and a family without a bold face keeps its regular one in silence. A formula therefore follows `\boldmath`, and a `\bfseries` that happens to precede it no longer leaks into it. The font identifier of the text size is recorded in `\g__LuaCJK_math_textfont_int`, which is the font a fallback chain tests for the character, because coverage does not depend on the size.

```
\int_new:N \g__LuaCJK_math_textfont_int
\cs_new_protected:Npn \__LuaCJK_math_family_fonts:nn #1#2
  {
    \group_begin:
      \__LuaCJK_family_font:n {#1}
      \fontseries { \mddefault }
      \fontshape { \shapedefault }
      \fontsize { \l__LuaCJK_math_size_tl pt } { \baselineskip }
```

```
      \selectfont
      \bool_lazy_and:nnT
        { \str_if_eq_p:ee { \math@version } { bold } }
        {
          \cs_if_exist_p:c
            { \use:c { f@encoding } / #1 / \bfdefault / \shapedefault }
        }
        { \fontseries { \bfdefault } \selectfont }
      \global \textfont #2 = \font
      \int_gset:Nn \g__LuaCJK_math_textfont_int
        { \tex_fontid:D \tex_font:D }
      \fontsize
        { \dim_eval:n { \l__LuaCJK_math_size_tl pt * 7 / 10 } }
        { \baselineskip }
      \selectfont
      \global \scriptfont #2 = \font
      \fontsize
        { \dim_eval:n { \l__LuaCJK_math_size_tl pt * 5 / 10 } }
        { \baselineskip }
      \selectfont
      \global \scriptscriptfont #2 = \font
    \group_end:
  }
```

(*End of definition for* `\__LuaCJK_math_family_fonts:nn`.)

`\__LuaCJK_register_math_family:n` Register a CJK family as a math family and point the math font attribute at its number. A CJK character in a math list needs a family rather than a box, because the family is what selects the font *and* its size for text, script, and scriptscript style; a box is built once and keeps the size it was built at.

```
\cs_new_protected:Npn \__LuaCJK_register_math_family:n #1
  {
    \__LuaCJK_math_family:n {#1}
    \int_set:Nn \l__LuaCJK_tmpa_int
      { \prop_item:Nn \g__LuaCJK_math_family_prop {#1} }
    \__LuaCJK_math_size_set:
    \__LuaCJK_math_family_fonts:nn {#1} \l__LuaCJK_tmpa_int
    \g__LuaCJK_math_font_attr = \l__LuaCJK_tmpa_int
  }
```

(*End of definition for* `\__LuaCJK_register_math_family:n`.)

`\__LuaCJK_math_fallbacks_push:n` Register the fallback fonts of a CJK family as math families and hand the chain to Lua, which sets the character in the first font that has it. The chain holds the text font of the family itself first, so that Lua can test it too, then each fallback in the

order it was declared, as pairs of a font identifier and a math family number. The family is given by the key it was declared under, which is the key of the fallback list, so a formula falls back like the text of the same family.

```
\tl_new:N \l__LuaCJK_math_fallback_ids_tl
\cs_new_protected:Npn \__LuaCJK_math_fallbacks_push:n #1
  {
    \bool_if:NT \g__LuaCJK_lua_loaded_bool
      {
        \__LuaCJK_math_size_set:
        \tl_clear:N \l__LuaCJK_math_fallback_ids_tl
        \tl_put_right:Nx \l__LuaCJK_math_fallback_ids_tl
          {
            \int_use:N \g__LuaCJK_math_textfont_int ,
            \int_use:N \g__LuaCJK_math_font_attr ,
          }
        \prop_get:NnNT \g__LuaCJK_fallback_prop {#1} \l__LuaCJK_tmpa_tl
          {
            \exp_args:NV \clist_map_inline:nn \l__LuaCJK_tmpa_tl
              {
                \__LuaCJK_math_family:n {##1}
                \int_set:Nn \l__LuaCJK_tmpa_int
                  { \prop_item:Nn \g__LuaCJK_math_family_prop {##1} }
                \__LuaCJK_math_family_fonts:nn {##1} \l__LuaCJK_tmpa_int
                \tl_put_right:Nx \l__LuaCJK_math_fallback_ids_tl
                  {
                    \int_use:N \g__LuaCJK_math_textfont_int ,
                    \int_use:N \l__LuaCJK_tmpa_int ,
                  }
              }
          }
        \lua_now:e
          {
            LuaCJK.set_math_fallback
              ( \int_use:N \g__LuaCJK_math_font_attr ,
                { \l__LuaCJK_math_fallback_ids_tl } )
          }
      }
  }
```

(*End of definition for* `\__LuaCJK_math_fallbacks_push:n`.)

`\__LuaCJK_math_version_refresh:` Register the math family again when the math version changes. LaTeX 2_ε knows the numbers of its own math families only, so `\mathversion` cannot reach the fonts of a CJK math family: the version is applied by hand. `\glb@settings` is where a version change and a text-size change are applied, and it runs

\every@math@size afterwards, so that register is where the refresh belongs. It runs there also when the group that changed the version ends and the version in force before the group is applied again, which is what keeps {\boldmath $中$} local. The guard stops the font selections of the registration from running the whole update again.

```
\cs_new_protected:Npn \__LuaCJK_math_version_refresh:
  {
    \bool_if:NF \l__LuaCJK_selectfont_bool
      {
        \bool_set_true:N \l__LuaCJK_selectfont_bool
        \__LuaCJK_math_font_update:
        \bool_set_false:N \l__LuaCJK_selectfont_bool
      }
  }
```

(*End of definition for* `\__LuaCJK_math_version_refresh:`.)

`__LuaCJK_toks_put_right:Nn` Append tokens to a token register such as \every@math@size. The register holds tokens rather than a token list variable, and \tl_put_right:Nn would put the name of the register into its own value, so the assignment is written out with \the.

```
\cs_new_protected:Npn \__LuaCJK_toks_put_right:Nn #1#2
  { #1 \expandafter { \tex_the:D #1 #2 } }
\exp_args:Nc \__LuaCJK_toks_put_right:Nn { every@math@size }
  { \__LuaCJK_math_version_refresh: }
```

(*End of definition for* `\__LuaCJK_toks_put_right:Nn`.)

`__LuaCJK_push_block_fonts:` Resolve the block fonts of the current family at the current size and hand them to Lua, keyed by the font identifier of the family. Font identifiers depend on the size, so this runs again whenever a font is selected.

```
\int_new:N \g__LuaCJK_block_fontid_int
\cs_new_protected:Npn \__LuaCJK_push_block_fonts:
  {
    \int_compare:nNnT { \g__LuaCJK_fontid_int } > { 0 }
      {
        \clist_map_inline:Nn \g__LuaCJK_block_names_clist
          {
            \tl_set:Ne \l__LuaCJK_block_key_tl { \l__LuaCJK_family_tl | ##1 }
            \int_gzero:N \g__LuaCJK_block_fontid_int
            \prop_get:NVNT \g__LuaCJK_family_block_prop
              \l__LuaCJK_block_key_tl \l__LuaCJK_tmpa_tl
              {
                \group_begin:
```

```
                  \exp_args:NV \__LuaCJK_family_font:n \l__LuaCJK_tmpa_tl
                  \selectfont
                  \int_gset:Nn \g__LuaCJK_block_fontid_int
                    { \tex_fontid:D \tex_font:D }
                \group_end:
              }
            \lua_now:e
              {
                LuaCJK.set_block_font
                  ( \int_use:N \g__LuaCJK_fontid_int ,
                    "\lua_escape:e {##1}" ,
                    \int_use:N \g__LuaCJK_block_fontid_int )
              }
          }
      }
  }
```

(*End of definition for* `\__LuaCJK_push_block_fonts:`.)

The glues and the settings belong to the size that is being selected, but `\baselineskip` has not changed yet when the `selectfont` hook runs, so they are refreshed after the font selection. The guard keeps the temporary font selections inside `\__LuaCJK_update_font:` from doing the same twice.

```
\hook_gput_code:nnn { selectfont } { LuaCJK } { \__LuaCJK_update_font: }
\hook_gput_code:nnn { cmd / selectfont / after } { LuaCJK }
  {
    \bool_if:NF \l__LuaCJK_selectfont_bool
      {
        \__LuaCJK_refresh_glues:
        \__LuaCJK_settings_push:
      }
  }
```

`\CJKrmdefault` `\CJKsfdefault` `\CJKttdefault` `\CJKfamilydefault`

The CJK families used by the standard font commands. The family hooks below re-select the CJK family whenever `\rmfamily` and its siblings run, and `\normalfont` selects `\CJKfamilydefault`, which follows `\CJKrmdefault` until the document gives it a value of its own. The value `\CJKfamilydefault` starts with, kept so that the hook at `\begin{document}` can tell an untouched default from a chosen one.

```
\clist_map_inline:nn { { rm } { zhsong } , { sf } { zhhei } , { tt } { zhfs } }
  {          \l__LuaCJK_family_default_init_tl
    \exp_args:Nc \tl_new:N { CJK \use_i:nn #1 default }
    \exp_args:Nc \tl_set:Nx { CJK \use_i:nn #1 default }
      { \use_ii:nn #1 }
```

```
  }
\tl_new:N \CJKfamilydefault
\tl_set:Nn \CJKfamilydefault { zhsong }
\tl_new:N \l__LuaCJK_family_default_init_tl
\tl_set_eq:NN \l__LuaCJK_family_default_init_tl \CJKfamilydefault
```

(*End of definition for* `\CJKrmdefault` *and others.*)

`\defaultCJKfontfeatures` Set the features prepended to every CJK family, replacing the initial `Script = CJK`. Families registered earlier keep the value they were given; call `\setCJKmainfont` again to apply a new default.

```
\NewDocumentCommand \defaultCJKfontfeatures { m }
  { \tl_set:Nn \l__LuaCJK_default_feature_tl {#1} }
```

(*End of definition for* `\defaultCJKfontfeatures`. *This function is documented on page* 31.)

`\addCJKfontfeatures` Add features to the current family for the rest of the group. The star form is accepted for compatibility; LuaCJK has no concept of blocks, so both forms act on the same font.

```
\NewDocumentCommand \addCJKfontfeatures { s o m }
  {
    \exp_args:NV \__LuaCJK_add_family_features:n
      \l__LuaCJK_family_tl {#3}
  }
```

Append features to the stored ones of a family and register it again.

`\__LuaCJK_add_family_features:n`

```
\cs_new_protected:Npn \__LuaCJK_add_family_features:n #1#2
  {
    \prop_get:NnNTF \g__LuaCJK_font_feature_prop {#1} \l__LuaCJK_feature_tl
      {
        \tl_set:Ne \l__LuaCJK_feature_tl
          {
            \exp_not:V \l__LuaCJK_feature_tl
            \tl_if_blank:nF {#2} { , \exp_not:n {#2} }
          }
        \prop_get:NnNF \g__LuaCJK_font_name_prop {#1} \l__LuaCJK_registered_name_tl
          { \tl_clear:N \l__LuaCJK_registered_name_tl }
        \prop_get:NnNF \g__LuaCJK_shift_decl_prop {#1} \l__LuaCJK_shift_dim_tl
          { \tl_clear:N \l__LuaCJK_shift_dim_tl }
        \group_begin:
          \fontspec_gset_family:NVV \l__LuaCJK_fontspec_tl
            \l__LuaCJK_feature_tl \l__LuaCJK_registered_name_tl
          \tl_if_empty:NF \l__LuaCJK_shift_dim_tl
            {
              \tl_set:Ne \l__LuaCJK_shift_key_tl
```

```
                { #1 | \l__LuaCJK_fontspec_tl }
              \prop_gput:NVV \g__LuaCJK_font_shift_prop
                \l__LuaCJK_shift_key_tl \l__LuaCJK_shift_dim_tl
            }
        \group_end:
        \prop_put:NnV \g__LuaCJK_family_prop       {#1}
          \l__LuaCJK_fontspec_tl
        \prop_put:NnV \g__LuaCJK_font_feature_prop {#1}
          \l__LuaCJK_feature_tl
        \selectfont
      }
      { \msg_warning:nnn { LuaCJK } { family-unknown } {#1} }
  }
```

(*End of definition for* `\addCJKfontfeatures` *and* `\__LuaCJK_add_family_features:n`*. This function is documented on page 31.*)

`\setCJKmainfont`
`\setCJKsansfont`
`\setCJKmonofont`
`\setCJKfamilyfont`

The public font commands, all built on `\__LuaCJK_set_family_with_options:nnnn`. The family of `\setCJKmainfont` and its siblings is already the value of `\CJKrmdefault` and friends; `\setCJKfamilyfont` expands its argument so that the same name reaches either command.

```
\NewDocumentCommand \setCJKmainfont { o m o }
  {
    \exp_args:NV \__LuaCJK_set_family_with_options:nnnn
      \CJKrmdefault {#1} {#2} {#3}
  }
\NewDocumentCommand \setCJKsansfont { o m o }
  {
    \exp_args:NV \__LuaCJK_set_family_with_options:nnnn
      \CJKsfdefault {#1} {#2} {#3}
  }
\NewDocumentCommand \setCJKmonofont { o m o }
  {
    \exp_args:NV \__LuaCJK_set_family_with_options:nnnn
      \CJKttdefault {#1} {#2} {#3}
  }
\NewDocumentCommand \setCJKfamilyfont { m o m o }
  {
    \tl_set:Ne \l__LuaCJK_tmpa_tl {#1}
    \exp_args:NV \__LuaCJK_set_family_with_options:nnnn
      \l__LuaCJK_tmpa_tl {#2} {#3} {#4}
  }
\NewDocumentCommand \setCJKfallbackfamilyfont { m o m o }
  {
    \tl_set:Ne \l__LuaCJK_fallback_key_tl {#1}
```

```
    \tl_if_novalue:nTF {#2}
      {
        \tl_if_novalue:nTF {#4}
          { \__LuaCJK_add_fallback:nn {  } {#3} }
          { \__LuaCJK_add_fallback:nn {#4} {#3} }
      }
      { \__LuaCJK_add_fallback:nn {#2} {#3} }
  }
```

Accept the options before or after the font name.

```
\cs_new_protected:Npn \__LuaCJK_set_family_with_options:nnnn #1#2#3#4
  {
    \tl_if_novalue:nTF {#2}
```

__LuaCJK_set_family_with_options:nnnn

```
      {
        \tl_if_novalue:nTF {#4}
          { \__LuaCJK_set_family_with_options_aux:nnn {#1} {  } {#3} }
          { \__LuaCJK_set_family_with_options_aux:nnn {#1} {#4} {#3} }
      }
      { \__LuaCJK_set_family_with_options_aux:nnn {#1} {#2} {#3} }
  }
\cs_generate_variant:Nn \__LuaCJK_set_family:nnn { nVn }
\cs_new_protected:Npn \__LuaCJK_set_family_with_options_aux:nnn #1#2#3
  {
    \__LuaCJK_extract_font_keys:nn {#1} {#2}
    \__LuaCJK_set_family:nVn {#1} \l__LuaCJK_features_tl {#3}
  }
```

Register one fallback font for a CJK family, keeping the order in which the fonts were declared.

```
\cs_new_protected:Npn \__LuaCJK_add_fallback:nn #1#2
  {
```

__LuaCJK_add_fallback:nn

```
    \__LuaCJK_parse_font_options:n {#1}
    \tl_if_empty:NT \l__LuaCJK_shift_tl
      { \tl_set:Nn \l__LuaCJK_features_tl {#1} }
    \group_begin:
      \exp_args:NV \__LuaCJK_build_features:n \l__LuaCJK_features_tl
      \fontspec_gset_family:NVn
        \l__LuaCJK_fallback_family_tl \l__LuaCJK_feature_tl {#2}
      \tl_if_empty:NF \l__LuaCJK_shift_tl
        {
          \tl_set:Ne \l__LuaCJK_shift_key_tl
            { \l__LuaCJK_fallback_key_tl | \l__LuaCJK_fallback_family_tl }
          \prop_gput:NVV \g__LuaCJK_font_shift_prop
            \l__LuaCJK_shift_key_tl \l__LuaCJK_shift_tl
        }
```

```
      \prop_get:NVNF \g__LuaCJK_fallback_prop
        \l__LuaCJK_fallback_key_tl \l__LuaCJK_tmpa_tl
        { \tl_clear:N \l__LuaCJK_tmpa_tl }
      \clist_put_right:NV \l__LuaCJK_tmpa_tl \l__LuaCJK_fallback_family_tl
      \prop_gput:NVV \g__LuaCJK_fallback_prop
        \l__LuaCJK_fallback_key_tl \l__LuaCJK_tmpa_tl
    \group_end:
    \__LuaCJK_refresh_fallbacks:
  }
```

(*End of definition for* `\setCJKmainfont` *and others. These functions are documented on page 27.*)

`\CJKfamily` Switch to a registered family and gobble the following spaces. The name is expanded first, so that a family is named by the macro that holds it: `\CJKfamily {⟨\CJKsfdefault⟩}` selects the family that `\sffamily` selects, and the name an undeclared family is reported with is the expanded one.

```
\NewDocumentCommand \CJKfamily { m }
  {
    \tl_set:Ne \l__LuaCJK_tmpa_tl {#1}
    \exp_args:NV \__LuaCJK_switch_family:n \l__LuaCJK_tmpa_tl
    \tex_ignorespaces:D
  }
```

(*End of definition for* `\CJKfamily`*. This function is documented on page 27.*)

`\newCJKfontfamily` Define a control sequence that switches to a new family, and register the family under the same name.

```
\NewDocumentCommand \newCJKfontfamily { o m o m o }
  {
    \tl_set:Ne \l__LuaCJK_tmpa_tl
      { \IfNoValueTF {#1} { \cs_to_str:N #2 } {#1} }
    \cs_set_protected:Npx #2
      { \exp_not:N \__LuaCJK_switch_family:n { \exp_not:V \l__LuaCJK_tmpa_tl } }
    \exp_args:NV \__LuaCJK_set_family_with_options:nnnn
      \l__LuaCJK_tmpa_tl {#3} {#4} {#5}
  }
```

(*End of definition for* `\newCJKfontfamily`*. This function is documented on page 27.*)

`\setCJKmathfont` Register a family used for CJK characters in math mode and refresh the attributes at once.

```
\NewDocumentCommand \setCJKmathfont { o m o }
  {
    \__LuaCJK_set_family_with_options:nnnn { LuaCJK / math } {#1} {#2} {#3}
    \tl_set:Nn \l__LuaCJK_math_family_tl { LuaCJK / math }
```

```
    \__LuaCJK_update_font:
  }
```

(*End of definition for* `\setCJKmathfont`*. This function is documented on page 32.*)

`\CJKfontspec` Register a one-off family and switch to it immediately. The name is built once and kept in `\l__LuaCJK_new_family_tl`, because the registration selects a font and therefore runs `\__LuaCJK_update_font:`, which spends the scratch variables on the family `PunctFamily` names. Gives every call its own family name.

```
\int_new:N \g__LuaCJK_fontspec_int
\NewDocumentCommand \CJKfontspec { o m }
  {
    \int_gincr:N \g__LuaCJK_fontspec_int
    \tl_set:Ne \l__LuaCJK_new_family_tl
      { LuaCJK / fontspec / \int_use:N \g__LuaCJK_fontspec_int }
    \exp_args:NV \__LuaCJK_set_family_with_options:nnnn
      \l__LuaCJK_new_family_tl {#1} {#2} { }
    \exp_args:NV \__LuaCJK_switch_family:n \l__LuaCJK_new_family_tl
    \tex_ignorespaces:D
  }
```

(*End of definition for* `\CJKfontspec` *and* `\g__LuaCJK_fontspec_int`*. This function is documented on page 28.*)

`\songti` `\heiti` `\fangsong` `\kaishu` `\lishu` `\youyuan` Shortcuts for the usual families, plus the family hooks of the standard font commands.

```
\cs_new_protected:Npn \songti   { \CJKfamily { zhsong } }
\cs_new_protected:Npn \heiti    { \CJKfamily { zhhei  } }
\cs_new_protected:Npn \fangsong { \CJKfamily { zhfs   } }
\cs_new_protected:Npn \kaishu   { \CJKfamily { zhkai  } }
\cs_new_protected:Npn \lishu    { \CJKfamily { zhli   } }
\cs_new_protected:Npn \youyuan  { \CJKfamily { zhyou  } }
\clist_map_inline:nn { rm, sf, tt }
  {
    \hook_gput_code:nnn { cmd / #1family / after } { LuaCJK }
      { \exp_args:Nv \__LuaCJK_switch_family:n { CJK#1default } }
  }
\hook_gput_code:nnn { cmd / normalfont / after } { LuaCJK }
  { \__LuaCJK_reset_family: }
```

(*End of definition for* `\songti` *and others. These functions are documented on page 28.*)

A.4 Names and configuration

\normalspacedchars

The characters in the list get no glue on either side, which is what one wants for slashes and hyphens inside Latin text. The list is kept on the Lua side as a set of code points, so the setting is global: a group does not restore the previous list. Every character of the list counts, separators included, so the initial list names its two characters without any. The list, initially / and -.

```
\g__LuaCJK_normal_spaced_chars_tl
\tl_new:N \g__LuaCJK_normal_spaced_chars_tl
\tl_gset:Nn \g__LuaCJK_normal_spaced_chars_tl
  {/-}

\cs_new_protected:Npn \__LuaCJK_set_normal_spaced_chars:
  {
    \lua_now:e
      {
        LuaCJK.set_normal_spaced_chars
          ( "\lua_escape:e { \g__LuaCJK_normal_spaced_chars_tl } " )
      }
  }
\NewDocumentCommand \normalspacedchars { m }
  {
    \tl_gset:Nn \g__LuaCJK_normal_spaced_chars_tl {#1}
    \__LuaCJK_set_normal_spaced_chars:
  }
```

(*End of definition for* `\normalspacedchars` *and* `\g__LuaCJK_normal_spaced_chars_tl`*. This function is documented on page 33.*)

__LuaCJK_setup_format_chars:

A character that only controls the text and has no advance of its own needs care: the font loader replaces one that the font does not have by a space glyph and a negative kern, and the spacing pass then sees a Latin atom and surrounds it with `CJKecglue` on both sides. There are two kinds. U+180E, U+200B, U+2060, and U+FEFF carry no information a font can use, so they are given category 9 and removed before the font loader runs. U+200C, U+200D, and the variation selectors are the ones a font shapes with: they are left alone here, and the pass that runs before shaping lets them follow the font of the character in front of them, so that what the font leaves of them can be dropped afterwards. A character produced by `\char` bypasses the category codes and is left alone: `\char` asks for the glyph itself, not for the character's typographic role. Without `pre_shaping_filter` there is no pass to tell the two kinds apart, so every character of the set is removed, which is what the package did before the shaping change.

```
\clist_const:Nn \c__LuaCJK_removed_chars_clist
  { "180E , "200B , "2060 , "FEFF }
```

```
\clist_const:Nn \c__LuaCJK_attached_chars_clist
  { "200C , "200D }
\cs_new_protected:Npn \__LuaCJK_setup_format_chars:
  {
    \clist_map_inline:Nn \c__LuaCJK_removed_chars_clist
      { \char_set_catcode_ignore:n {##1} }
    \bool_if:NF \g__LuaCJK_shaping_bool
      {
        \clist_map_inline:Nn \c__LuaCJK_attached_chars_clist
          { \char_set_catcode_ignore:n {##1} }
        \int_step_inline:nnn { "FE00 } { "FE0F }
          { \char_set_catcode_ignore:n {##1} }
        \int_step_inline:nnn { "E0100 } { "E01EF }
          { \char_set_catcode_ignore:n {##1} }
      }
  }
```

(*End of definition for* `\__LuaCJK_setup_format_chars:`.)

`\__LuaCJK_load_cj:` Read the `CJ` set from `LineBreak.txt` once. The key `CJLineBreak` calls this when it selects `strict`; a second call is cheap because the Lua side remembers that it has read the file.

```
\cs_new_protected:Npn \__LuaCJK_load_cj:
  { \lua_now:e { LuaCJK.load_cj() } }
```

(*End of definition for* `\__LuaCJK_load_cj:`.)

`_LuaCJK_load_local_config:` Load the file chosen by the `LocalConfig` option. The key is read while the package is being loaded, but the file is only input at the end, so that it can use everything LuaCJK defines.

```
\cs_new_protected:Npn \__LuaCJK_load_local_config:
  {
    \tl_if_empty:NF \l__LuaCJK_local_config_tl
      {
        \exp_args:NV \__LuaCJK_load_config_file:n \l__LuaCJK_local_config_tl
      }
  }
\cs_new_protected:Npn \__LuaCJK_load_config_file:n #1
  {
    \file_if_exist:nTF {#1}
      { \file_input:n {#1} }
      { \msg_warning:nnn { LuaCJK } { config-missing } {#1} }
  }
\msg_new:nnn { LuaCJK } { config-missing }
  { The ~ configuration ~ file ~ `#1' ~ was ~ not ~ found. }
```

(*End of definition for* `\__LuaCJK_load_local_config:`.)

`\__LuaCJK_load_lxgw:` When the `LoadLXGW` option is on and the document did not declare any CJK font, fall back to the LXGW faces. This runs at `\begin{document}` so that a family declared anywhere in the preamble still wins, and a document that declares some families but not the default one is reported by `\__LuaCJK_ensure_family_default:` instead. The roman family is declared twice, once as the main family and once under the name `zhsong`, and both declarations carry the same options, because a family name holds one set of features and the later declaration would otherwise drop what the earlier one gave. Neo ZhiSong has no italic face, and emphasis in Chinese is set in Kai, so the italic shape of the roman family is WenKai and its bold italic ZhenKai Slab. Neo ZhiSong, Neo XiHei, and Zhuque Fangsong have no bold face of their own, so their declarations ask fontspec for fake bold, Neo XiHei and Zhuque Fangsong ask for fake slant as well, and the WenKai family takes ZhenKai Slab as its bold face.

```
\cs_new_protected:Npn \__LuaCJK_load_lxgw:
  {
    \setCJKmainfont{LXGWNeoZhiSong.ttf}
      [ AutoFakeBold = 4,
        ItalicFont     = LXGWWenKaiGBLite-Regular.ttf,
        BoldItalicFont = LXGWZhenKaiSlabGB-Regular.ttf ]
    \setCJKsansfont{LXGWNeoXiHei.ttf}
      [ AutoFakeBold = 4, AutoFakeSlant ]
    \setCJKmonofont{LXGWZhuqueFangsong-Regular.ttf}
      [ AutoFakeBold = 4, AutoFakeSlant ]
    \setCJKfamilyfont{zhsong}{LXGWNeoZhiSong.ttf}
      [ AutoFakeBold = 4,
        ItalicFont     = LXGWWenKaiGBLite-Regular.ttf,
        BoldItalicFont = LXGWZhenKaiSlabGB-Regular.ttf ]
    \setCJKfamilyfont{zhhei}{LXGWNeoXiHei.ttf}
      [ AutoFakeBold = 4, AutoFakeSlant ]
    \setCJKfamilyfont{zhkai}{LXGWWenKaiGBLite-Regular.ttf}
      [ AutoFakeSlant,
        BoldFont       = LXGWZhenKaiSlabGB-Regular.ttf,
        ItalicFont     = LXGWWenKaiGBLite-Regular.ttf,
        BoldItalicFont = LXGWZhenKaiSlabGB-Regular.ttf ]
    \setCJKfamilyfont{zhfs}{LXGWZhuqueFangsong-Regular.ttf}
      [ AutoFakeBold = 4, AutoFakeSlant ]
  }
```

(*End of definition for* `\__LuaCJK_load_lxgw:`.)

A.5 The Lua module

The Lua module reads the attribute numbers and installs its callbacks. The current font is resolved once the document starts.

```
\lua_load_module:n { LuaCJK }
\lua_now:e
  {
    LuaCJK.attr_font        =  \int_use:N   \c__LuaCJK_font_attr_int ;
    LuaCJK.attr_cjk_space   =  \int_use:N   \c__LuaCJK_cjk_space_attr_int ;
    LuaCJK.attr_xecglue     =  \int_use:N   \c__LuaCJK_xecglue_attr_int ;
    LuaCJK.attr_inserted    =  \int_use:N   \c__LuaCJK_inserted_attr_int ;
    LuaCJK.attr_math_font   =  \int_use:N   \c__LuaCJK_math_font_attr_int ;
    LuaCJK.attr_math_bracket
                            =  \int_use:N   \c__LuaCJK_math_bracket_attr_int ;
    LuaCJK.attr_punct_font  =  \int_use:N   \c__LuaCJK_punct_font_attr_int ;
    LuaCJK.attr_nobreak     =  \int_use:N   \c__LuaCJK_nobreak_attr_int ;
    LuaCJK.attr_settings    =  \int_use:N   \c__LuaCJK_settings_attr_int ;
    LuaCJK.attr_disabled    =  \int_use:N   \c__LuaCJK_disabled_attr_int ;
    LuaCJK.attr_attached    =  \int_use:N   \c__LuaCJK_attached_attr_int ;
    LuaCJK.attr_explicit_space
                            =  \int_use:N   \c__LuaCJK_explicit_space_attr_int ;
    LuaCJK.attr_shift       =  \int_use:N   \c__LuaCJK_shift_attr_int ;
    LuaCJK.attr_punct_shift =  \int_use:N   \c__LuaCJK_punct_shift_attr_int ;
    LuaCJK.attr_shifted     =  \int_use:N   \c__LuaCJK_shifted_attr_int ;
    LuaCJK.name_ccglue      = "\cs_to_str:N \l__LuaCJK_ccglue_skip" ;
    LuaCJK.name_ecglue      = "\cs_to_str:N \l__LuaCJK_ecglue_skip" ;
  }
\bool_gset_true:N \g__LuaCJK_lua_loaded_bool
```

`\__LuaCJK_detect_shaping:` Ask the module whether luaotfload offers the callback that runs before shaping, and record the answer for the two places that depend on it: the character setup below and the pass that `LuaCJK-core.lua` installs. Without the callback the package keeps its previous behavior in full, including the input removal of every zero-width character, so that a system never applies half of the change.

```
\bool_new:N \g__LuaCJK_shaping_bool
\cs_new_protected:Npn \__LuaCJK_detect_shaping:
  {
    \str_if_eq:eeT
      { \lua_now:e { tex.sprint(tostring(LuaCJK.shaping)) } }
      { true }
      { \bool_gset_true:N \g__LuaCJK_shaping_bool }
    \bool_if:NF \g__LuaCJK_shaping_bool
      { \msg_warning:nn { LuaCJK } { no-pre-shaping } }
  }
```

```
\msg_new:nnn { LuaCJK } { no-pre-shaping }
  {
    This~ luaotfload~ has~ no~ pre_shaping_filter,~ so~ the~ CJK~ font~ is~
    applied~ after~ the~ text~ is~ shaped:~ a~ combining~ mark~ will~ not~
    attach~ to~ its~ base,~ and~ the~ joiners~ and~ variation~ selectors~
    are~ removed~ at~ the~ input.
  }
```

(*End of definition for* `\__LuaCJK_detect_shaping:`.)

Read the package options now that every helper a key code calls exists and the Lua module is loaded, which `CJLineBreak` needs. The hook at `\begin{document}` then settles the family the document starts in: `\CJKfamilydefault` follows `\CJKrmdefault` when the preamble did not choose it, `LoadLXGW` supplies the LXGW faces when the preamble declared no family at all, a chosen default that no family carries is replaced by one that does, and the family that survives is selected, so a preamble that redefines `\CJKfamilydefault` sets the main text in that family from the first line instead of waiting for a `\normalfont`.

```
\ProcessKeyOptions [ LuaCJK / options ]
\__LuaCJK_settings_push:
\__LuaCJK_set_normal_spaced_chars:
\__LuaCJK_detect_shaping:
\__LuaCJK_setup_format_chars:
\hook_gput_code:nnn { begindocument } { LuaCJK }
  {
    \tl_if_eq:NNT \CJKfamilydefault \l__LuaCJK_family_default_init_tl
      {
        \group_begin:
        \tl_gset:Ne \CJKfamilydefault
          {
            \str_case:onF { \familydefault }
              {
                { \rmdefault } { \exp_not:N \CJKrmdefault }
                { \sfdefault } { \exp_not:N \CJKsfdefault }
                { \ttdefault } { \exp_not:N \CJKttdefault }
              }
              { \exp_not:N \CJKrmdefault }
          }
        \group_end:
      }
    \bool_if:NT \g__LuaCJK_load_lxgw_bool
      {
        \prop_if_empty:NT \g__LuaCJK_family_prop
```

```
          { \__LuaCJK_load_lxgw: }
      }
    \__LuaCJK_ensure_family_default:
    \__LuaCJK_reset_family:
    \__LuaCJK_update_font:
    \g__LuaCJK_font_attr = \g__LuaCJK_fontid_int
  }
\__LuaCJK_load_local_config:
\hook_gput_code:nnn { package / listings / after } { LuaCJK }
  { \RequirePackage { LuaCJK-listings } }
\cs_if_exist:cT { ver@listings.sty }
  { \RequirePackage { LuaCJK-listings } }
```

⟨/package⟩

The package ships two Lua modules. `LuaCJK.lua` only loads the core module; all node processing lives in `LuaCJK-core.lua`.

LuaCJK The table shared by the two modules. The core module adds its functions and constants to it.

```
⟨*lua⟩
LuaCJK = LuaCJK or {}

require("LuaCJK-core")
⟨/lua⟩
```

(*End of definition for* `LuaCJK`.)

The core module starts by creating the shared table and caching the node identifiers. The callbacks run on every list, so these lookups happen once.

```
⟨*lua-core⟩
LuaCJK = LuaCJK or {}

```

M Short names for the module table, the node types, and the node functions.

```
local M              = LuaCJK
local GLYPH          = node.id("glyph")
local GLUE           = node.id("glue")
local HLIST          = node.id("hlist")
local VLIST          = node.id("vlist")
local KERN           = node.id("kern")
local RULE           = node.id("rule")
local LOCAL_PAR      = node.id("local_par")
local LIGATURE       = node.id("ligature")
local PENALTY        = node.id("penalty")
```

GLYPH
GLUE
HLIST
VLIST
KERN
LIGATURE
PENALTY
MATH
DISC
WHATSIT
MARK
INS
ADJUST
LOCAL_PAR
SPACESKIP
ITALICCORR
new_node
free_node

```
local MATH            = node.id("math")
local DISC            = node.id("disc")
local WHATSIT         = node.id("whatsit")
local MARK            = node.id("mark")
local INS             = node.id("ins")
local ADJUST          = node.id("adjust")
local LOCAL_PAR       = node.id("local_par")
local SPACESKIP       = node.subtype("spaceskip") or 13
local ITALICCORR      = node.subtype("italiccorrection") or 3
local new_node        = node.new
local free_node       = node.free
local has_attribute   = node.has_attribute
local set_attribute   = node.set_attribute
local unset_attribute = node.unset_attribute

```

(*End of definition for* `M` *and others.*)

`LuaCJK.shaping` Whether luaotfload offers the callback that runs before a list is shaped. The TEX side reads this to decide how the zero-width characters are handled at the input, and the module uses it to choose between the substitution before shaping and the copy after it.

```
M.shaping = luatexbase.callbacktypes ~= nil
  and luatexbase.callbacktypes.pre_shaping_filter ~= nil
```

(*End of definition for* `LuaCJK.shaping`.)

`LuaCJK.attr_font` The attribute numbers. The TEX side fills them in after loading the module.

`LuaCJK.attr_cjk_space`
`LuaCJK.attr_xecglue`
`LuaCJK.attr_inserted`
`LuaCJK.attr_math_font`
`LuaCJK.attr_explicit_space`

```
for _, field in ipairs {
  "font",      "cjk_space",  "xecglue", "inserted",
  "math_font", "punct_font", "nobreak", "settings",
  "attached",  "explicit_space",
} do
  local key = "attr_" .. field
  M[key] = M[key] or nil
end

```

(*End of definition for* `LuaCJK.attr_font` *and others.*)

A.6 Settings and data

Most options describe how text is set, not how the document is configured, so they have to travel with the characters. A snapshot of the current options is registered once and its number is stored in an attribute on the glyphs created

afterwards; a group that ends before the paragraph does therefore still decides how its own text is set.

LuaCJK.settings
aCJK.settings_by_signature
LuaCJK.attr_settings

The snapshots, an index from a signature to a snapshot so that repeated font selections reuse one entry, and the attribute number.

```
local DEFAULT_SETTINGS = {
  ccglue        = { width = 0, stretch = 0, shrink = 0 },
  ecglue        = { width = 0, stretch = 0, shrink = 0 },
  punct_style   = "quanjiao",
  kinsoku       = true,
  cj_strict     = false,
  latin_punct   = false,
  cjk_math      = false,
  punct_width   = false,
  bound_width   = false,
  check_single  = false,
  widow_penalty = 10000,
  allow_break   = false,
  auto_fallback = false,
  blocks_off    = {},
}

M.settings              = { DEFAULT_SETTINGS }
M.settings_by_signature = {}

```

(*End of definition for* `LuaCJK.settings`*,* `LuaCJK.settings_by_signature`*, and* `LuaCJK.attr_settings`*.*)

LuaCJK.new_settings

Read the current glue registers, build a snapshot, and select it. The glue has to be read here rather than passed in, because that is the only place where the registers still hold the values of the moment. The stretch and shrink orders travel with the value, so a register such as `0pt plus 1filll` can still fill a line after the node has been built.

```
local function glue_spec (name)
  local skip = name and tex.skip[name]
  if skip then
    return { width         = skip.width,
             stretch       = skip.stretch,
             shrink        = skip.shrink,
             stretch_order = skip.stretch_order,
             shrink_order  = skip.shrink_order }
  end
  return { width = 0, stretch = 0, shrink = 0 }
```

```
end

function M.new_settings (style, kinsoku, cj_strict, latin_punct, cjk_math,
                         punct_width, bound_width, check_single, widow_penalty,
                         allow_break, auto_fallback, off_list)
  local off = {}
  for name in string.gmatch(off_list or "", "[^,]+") do
    local trimmed = string.match(name, "^%s*(.-)%s*$")
    if trimmed ~= "" then
      off[trimmed] = true
    end
  end
  local entry = {
    ccglue        = glue_spec(M.name_ccglue),
    ecglue        = glue_spec(M.name_ecglue),
    punct_style   = style,
    kinsoku       = kinsoku,
    cj_strict     = cj_strict,
    latin_punct   = latin_punct,
    cjk_math      = cjk_math,
    punct_width   = punct_width,
    bound_width   = bound_width,
    check_single  = check_single,
    widow_penalty = widow_penalty,
    allow_break   = allow_break,
    auto_fallback = auto_fallback,
    blocks_off    = off,
  }
  local signature = table.concat({
    entry.ccglue.width, entry.ccglue.stretch, entry.ccglue.shrink,
    entry.ccglue.stretch_order, entry.ccglue.shrink_order,
    entry.ecglue.width, entry.ecglue.stretch, entry.ecglue.shrink,
    entry.ecglue.stretch_order, entry.ecglue.shrink_order,
    style, tostring(kinsoku), tostring(cj_strict), tostring(latin_punct),
    tostring(cjk_math),
    tostring(punct_width), tostring(bound_width),
    tostring(check_single), widow_penalty,
    tostring(allow_break), tostring(auto_fallback), off_list,
  }, "|")
  local id = M.settings_by_signature[signature]
  if not id then
    id                                = #M.settings + 1
    M.settings[id]                    = entry
    M.settings_by_signature[signature] = id
  end
```

```
  if M.attr_settings then
    tex.setattribute(M.attr_settings, id)
  end
  return id
end

```

(*End of definition for* `LuaCJK.new_settings`.)

`settings_of` The snapshot of a node, or the defaults when it carries none.

```
local function settings_of (n)
  if M.attr_settings and n then
    local id = has_attribute(n, M.attr_settings)
    if id and id > 0 and M.settings[id] then
      return M.settings[id]
    end
  end
  return M.settings[1]
end

M.settings_of = settings_of

```

(*End of definition for* `settings_of`.)

`boundary_settings` The snapshot a boundary reads. A boundary belongs to the CJK character on one side of it, so a CJK–Latin boundary takes that character's snapshot: the glue is the CJK text's, and a Latin run set at another size must not change it on one side only. A CJK–CJK boundary keeps reading the character in front of it, which is a CJK character as well, and a boundary with no CJK side has no glue to size. The CJK side is preferred only when it carries a snapshot; text set before any font selection carries none, and the other side decides then.

```
local function boundary_settings (prev_visible, prev_end, cur, start_class)
  if start_class == "CJK" and prev_end ~= "CJK" then
    local id = M.attr_settings and cur and has_attribute(cur, M.attr_settings)
    if id and id > 0 and M.settings[id] then
      return M.settings[id]
    end
  end
  return settings_of(prev_visible)
end
```

(*End of definition for* `boundary_settings`.)

LuaCJK.cj
LuaCJK.cj_count
LuaCJK.cj_loaded

The characters whose `Line_Break` property is `CJ` (the Japanese small kana and the like), how many there are, and whether the file has been read. The set is read from `LineBreak.txt` of the unicode-data bundle only when `CJLineBreak` asks for `strict`, so a document that keeps the default never pays for the file.

```
M.cj        = {}
M.cj_count  = 0
M.cj_loaded = false

local function read_cj (data)
  local set, count = {}, 0
  for first, last, class in data:gmatch("(%x+)%.%.(%x+)%s*;%s*(%a+)") do
    if class == "CJ" then
      for code = tonumber(first, 16), tonumber(last, 16) do
        set[code] = true
      end
      count = count + 1
    end
  end
  for code, class in data:gmatch("(%x+)%s*;%s*(%a+)") do
    if class == "CJ" then
      set[tonumber(code, 16)] = true
      count = count + 1
    end
  end
  return set, count
end

function M.load_cj ()
  if M.cj_loaded then
    return M.cj_count
  end
  M.cj_loaded = true
  local path = kpse and kpse.find_file and kpse.find_file("LineBreak.txt")
  local file = path and io.open(path)
  if not file then
    texio.write_nl("term and log",
      "Package LuaCJK Warning: LineBreak.txt not found; "
      .. "CJLineBreak = strict has no effect.")
    return 0
  end
  local data = file:read("*a")
  file:close()
  M.cj, M.cj_count = read_cj(data)
  return M.cj_count
```

```
end

```

(*End of definition for* `LuaCJK.cj`, `LuaCJK.cj_count`, *and* `LuaCJK.cj_loaded`.)

The middle dot, the quotation marks, the horizontal bar, the two dot leader, the ellipses, and the hyphenation point are shared with Latin scripts. With `LatinPunct` on, the marks that key lists use the Latin font and are not compressed; the em dash is not in the table, because it stays CJK.

`LuaCJK.is_cjk` Test whether a character is treated as CJK. The table is the set of code points this package sets with the CJK class. It is a policy rather than a Unicode property: the punctuation that CJK text shares with Latin (the middle dot U+00B7, the quotation marks, the horizontal bar U+2015, the two dot leader U+2025, the ellipses U+2026 and U+22EF, and the hyphenation point U+2027) is listed here even though its script is Common, and the combining Bopomofo tone marks U+02EA and U+02EB are left out on purpose, because they attach to the preceding letter and a glue between the two would break the attachment. The table is not the whole class: `\LuaCJKDeclareCharClass` declares a code point CJK or default, and a declared sub-block adds its ranges to the class, which is what lets a block outside the built-in ranges reach the font its family gives it. The explicit declaration is read first, so `Default` takes a code point back out of a block, and a block is read before the shared punctuation of `LatinPunct`, because the block names the font that sets it. The third argument is the font of the glyph, and it decides the code points below U+0100: a font that luaotfload did not load as Unicode holds an encoding rather than characters, so the number that reaches the node is a position in that encoding. Slot 0xB7 of pifont's dingbat font would otherwise be read as the shared middle dot, and a CJK font that lacks it would drop the character; the accent `\r{u}`, slot 0xB7 of a T1 text font, would be read the same way. A code point above U+00FF cannot be a slot of an 8-bit font, so a CJK character typed in a legacy encoding still takes the CJK font, and a Unicode font is read as characters throughout. The answer a block gives for a code point is remembered in a table of its own, because the passes ask `\LuaCJK.is_cjk` once per glyph: the walk over the declared blocks happens once per character, and a declaration empties the table, since it can cover a code point the table has already answered for. The script `scripts/check-charclass.lua` compares the table with `Scripts.txt` and reports a code point whose script is Han, Hiragana, Katakana, Bopomofo, or Hangul but which the table does not list.

```
M.class_override = {}
M.punct_override = {}

```

```
local CJK_RANGES = {
  { 0x00B7,  0x00B7 },  -- middle dot, shared with Latin
  { 0x1100,  0x11FF },  -- Hangul Jamo
  { 0x2014,  0x2014 },  -- em dash, shared with Latin
  { 0x2015,  0x2015 },  -- horizontal bar, shared with Latin
  { 0x2E3A,  0x2E3B },  -- two- and three-em dashes, shared with Latin
  { 0x2018,  0x2019 },  -- single quotation marks
  { 0x201C,  0x201D },  -- double quotation marks
  { 0x2025,  0x2025 },  -- two dot leader, shared with Latin
  { 0x2026,  0x2026 },  -- horizontal ellipsis, shared with Latin
  { 0x2027,  0x2027 },  -- hyphenation point, shared with Latin
  { 0x22EF,  0x22EF },  -- midline ellipsis, shared with Latin
  { 0x2E80,  0x9FFF },  -- radicals, punctuation, kana, ideographs
  { 0xA960,  0xA97C },  -- Hangul Jamo Extended-A
  { 0xAC00,  0xD7C6 },  -- Hangul syllables and Jamo Extended-B
  { 0xD7CB,  0xD7FB },  -- Hangul Jamo Extended-B
  { 0xF900,  0xFAFF },  -- compatibility ideographs
  { 0xFE30,  0xFE4F },  -- compatibility forms
  { 0xFF00,  0xFFEF },  -- fullwidth forms
  { 0x16FE2, 0x16FE3 }, -- Old Chinese hook and iteration marks
  { 0x16FF0, 0x16FF6 }, -- Han, Vietnamese alternate reading marks
  { 0x1AFF0, 0x1AFF3 }, -- Kana Extended-B
  { 0x1AFF5, 0x1AFFB },
  { 0x1AFFD, 0x1AFFE },
  { 0x1B000, 0x1B128 }, -- Kana Supplement and Kana Extended-A
  { 0x1B132, 0x1B132 },
  { 0x1B150, 0x1B152 },
  { 0x1B155, 0x1B155 },
  { 0x1B164, 0x1B168 },
  { 0x1F200, 0x1F200 }, -- square hiragana hoka
  { 0x20000, 0x3FFFF }, -- ideographic planes
}

local function in_ranges (c, ranges)
  for i = 1, #ranges do
    if c >= ranges[i][1] and c <= ranges[i][2] then
      return true
    end
  end
  return false
end

local LATIN_PUNCT = {
  [0x00B7] = true, [0x2018] = true, [0x2019] = true,
  [0x201C] = true, [0x201D] = true, [0x2025] = true,
```

```
  [0x2026] = true, [0x2027] = true, [0x22EF] = true,
}

local SLOT_FONT = { }

local BLOCK_CHARS = { }

local function is_slot (c, f)
  if c > 0xFF or not f then
    return false
  end
  local known = SLOT_FONT[f]
  if known == nil then
    local data = font.getfont(f)
    known = not (data and data.encodingbytes == 2)
    SLOT_FONT[f] = known
  end
  return known
end

local function is_cjk (c, s, f)
  local override = M.class_override[c]
  if override ~= nil then
    return override
  end
  local block = BLOCK_CHARS[c]
  if block == nil then
    block = false
    for _, name in ipairs(M.block_order) do
      for _, range in ipairs(M.blocks[name].ranges) do
        if c >= range[1] and c <= (range[2] or range[1]) then
          block = true
          break
        end
      end
      if block then
        break
      end
    end
    BLOCK_CHARS[c] = block
  end
  if block then
    return true
  end
  if s and s.latin_punct and LATIN_PUNCT[c] then
```

```
    return false
  end
  return in_ranges(c, CJK_RANGES) and not is_slot(c, f)
end

M.is_cjk = is_cjk

```

(*End of definition for* `LuaCJK.is_cjk`.)

`LuaCJK.is_attached` Test whether a character attaches to the character in front of it instead of standing on its own: the two joiners and the variation selectors. A font shapes with these, and their own advance is nothing, so they follow the font of the character they attach to and take no part in the classes. The combining marks are not in this set: a CJK one is in the CJK ranges and is shaped with its base, which is where its attachment comes from.

```
local ATTACHED = {
  [0x200C] = true, [0x200D] = true,
}

local function is_attached (c)
  return ATTACHED[c]
    or (c >= 0xFE00  and c <= 0xFE0F)
    or (c >= 0xE0100 and c <= 0xE01EF)
end

M.is_attached = is_attached

```

(*End of definition for* `LuaCJK.is_attached`.)

`PUNCT_CLASS` The punctuation LuaCJK knows about, grouped as opening, closing, middle, dash, ellipsis, and the Latin closing marks. The group decides compression, line-breaking prohibitions, and hanging. The groups are the lists of xeCJK: its `OP` and `PR` are opening, its `CL`, `PO`, `EX`, and `IS` are closing, its hyphens are dashes, and its `NS` is read as closing or middle. A mark is listed here only when the CJK table holds it as well, so the small-form and vertical variants outside that table keep no class. The ASCII brackets are the exception, because they are set in the Latin font and cannot be compressed or hung: `half-open` and `half-close` are their own groups, which only the prohibitions and the spacing read. The Latin closing marks are the other exception, for the same reason: `latin-close` holds the marks that end an English phrase even inside CJK text, so they take the zero-

width boundary glue of a mark and the prohibition of a closing one, and nothing an advance of the CJK font brings.

```
local PUNCT_CLASS = {
  [0x0021] = "latin-close", [0x0022] = "latin-close",
  [0x0025] = "latin-close", [0x0027] = "latin-close",
  [0x002C] = "latin-close", [0x002E] = "latin-close",
  [0x003A] = "latin-close", [0x003B] = "latin-close",
  [0x003F] = "latin-close",
  [0x0028] = "half-open",  [0x005B] = "half-open",
  [0x007B] = "half-open",
  [0x0029] = "half-close", [0x005D] = "half-close",
  [0x007D] = "half-close",
  [0x2018] = "open",     [0x201C] = "open",
  [0x3008] = "open",     [0x300A] = "open",
  [0x300C] = "open",     [0x300E] = "open",
  [0x3010] = "open",     [0x3014] = "open",
  [0x3016] = "open",     [0x3018] = "open",
  [0x301A] = "open",     [0x301D] = "open",
  [0xFE35] = "open",     [0xFE37] = "open",
  [0xFE39] = "open",     [0xFE3B] = "open",
  [0xFE3D] = "open",     [0xFE3F] = "open",
  [0xFE41] = "open",     [0xFE43] = "open",
  [0xFE47] = "open",     [0xFF04] = "open",
  [0xFF08] = "open",     [0xFF3B] = "open",
  [0xFF5B] = "open",     [0xFF5F] = "open",
  [0xFF62] = "open",     [0xFFE1] = "open",
  [0xFFE5] = "open",     [0xFFE6] = "open",
  [0x2019] = "close",    [0x201D] = "close",
  [0x3001] = "close",    [0x3002] = "close",
  [0x3009] = "close",    [0x300B] = "close",
  [0x300D] = "close",    [0x300F] = "close",
  [0x3011] = "close",    [0x3015] = "close",
  [0x3017] = "close",    [0x3019] = "close",
  [0x301B] = "close",    [0x301E] = "close",
  [0x301F] = "close",    [0xFE36] = "close",
  [0xFE38] = "close",    [0xFE3A] = "close",
  [0xFE3C] = "close",    [0xFE3E] = "close",
  [0xFE40] = "close",    [0xFE42] = "close",
  [0xFE44] = "close",    [0xFE48] = "close",
  [0xFF01] = "close",    [0xFF05] = "close",
  [0xFF09] = "close",    [0xFF0C] = "close",
  [0xFF0E] = "close",    [0xFF1A] = "close",
  [0xFF1B] = "close",    [0xFF1F] = "close",
  [0xFF3D] = "close",    [0xFF5D] = "close",
```

```
  [0xFF60] = "close",    [0xFF61] = "close",
  [0xFF63] = "close",    [0xFF64] = "close",
  [0xFFE0] = "close",
  [0x00B7] = "middle",   [0x2027] = "middle",
  [0x30FB] = "middle",   [0xFF65] = "middle",
  [0x2013] = "dash",     [0x2014] = "dash",
  [0x2015] = "dash",     [0x2E3A] = "dash",
  [0x2E3B] = "dash",     [0x301C] = "dash",
  [0x30A0] = "dash",     [0xFF0D] = "dash",
  [0xFF5E] = "dash",
  [0x2025] = "ellipsis", [0x2026] = "ellipsis", [0x22EF] = "ellipsis",
}

```

(*End of definition for* `PUNCT_CLASS`.)

`half_mark` Whether a group belongs to the half-width brackets.

```
local function half_mark (mark)
  return mark == "half-open" or mark == "half-close"
end

```

(*End of definition for* `half_mark`.)

`inert_mark` Whether a group is a mark whose advance belongs to the font alone, so that nothing a CJK advance would bring reaches it: the half-width brackets and the Latin closing marks.

```
local function inert_mark (mark)
  return half_mark(mark) or mark == "latin-close"
end

```

(*End of definition for* `inert_mark`.)

`punct_class` Look up the group of a character. The font is passed on to `\LuaCJK.is_cjk`'s rule, because a slot of a legacy font is a punctuation mark no more than it is a CJK character: the middle dot of pifont's slot 0xB7 takes no compression and no hanging.

```
local function punct_class (c, s, f)
  local override = M.punct_override[c]
  if override ~= nil then
    return override or nil
  end
  if is_slot(c, f) then
    return nil
```

```
  end
  if s and s.latin_punct and LATIN_PUNCT[c] then
    return nil
  end
  if M.charsets.middle[c] then
    return "middle"
  end
  return PUNCT_CLASS[c]
end

```

(*End of definition for* `punct_class`.)

`LuaCJK.declare_ranges` Fill a table with ⟨*value*⟩ over a list of code-point ranges, each given as `{first, last}`.

```
local function fill_ranges (target, value, ranges)
  for _, range in ipairs(ranges) do
    local first = range[1]
    local last  = range[2] or first
    for code = first, last do
      target[code] = value
    end
  end
end

function M.declare_class (ranges, cjk)
  fill_ranges(M.class_override, cjk, ranges)
end

function M.declare_punct (ranges, group)
  fill_ranges(M.punct_override, group, ranges)
end

function M.reset_classes ()
  M.class_override = {}
  M.punct_override = {}
  M.normal_spaced  = {}
end

```

(*End of definition for* `LuaCJK.declare_ranges`.)

`M.charsets` The four character sets that the punctuation options can replace. `kaiming` holds the marks that keep a full em width in the `kaiming` style; `middle` and `long` are filled from the punctuation groups below, and `nobreak` starts as a copy of `long`.

```
M.charsets = {
```

```
  kaiming = {
    [0xFF0E] = true,  -- .
    [0x3002] = true,  -- 。
    [0xFF1F] = true,  -- ?
    [0xFF01] = true,  -- !
  },
  middle  = {},
  long    = {},
  nobreak = {},
}

for code, group in pairs(PUNCT_CLASS) do
  if group == "middle" then
    M.charsets.middle[code] = true
    PUNCT_CLASS[code]       = nil
  elseif group == "dash" or group == "ellipsis" then
    M.charsets.long[code] = true
  end
end

for code in pairs(M.charsets.long) do
  M.charsets.nobreak[code] = true
end

```

(*End of definition for* `M.charsets`.)

`LuaCJK.set_charset` Replace, extend, or reduce one of the sets above. The ranges arrive from TEX as a list of `{first, last}` pairs.

```
function M.set_charset (kind, mode, ranges)
  local target = M.charsets[kind]
  if not target then
    return
  end
  if mode == "set" then
    target           = {}
    M.charsets[kind] = target
  end
  for _, range in ipairs(ranges) do
    local first = range[1]
    local last  = range[2] or first
    for code = first, last do
      if mode == "remove" then
        target[code] = nil
      else
```

```
        target[code] = true
      end
    end
  end
end

```

(*End of definition for* `LuaCJK.set_charset`.)

`LuaCJK.punct_styles` The built-in styles. Each entry holds the compression switch, the compression ratio, the opening and closing exemption, whether the line-end hanging applies, and the optional fixed width.

```
M.punct_class = function (c, s, f)
  return punct_class(c, s or M.settings[1], f)
end

M.punct_styles = {
  quanjiao = { compress_adjacent = true,  ratio = 0.5,
               keep_open_close = true, hang = true },
  plain    = { compress_adjacent = false, ratio = 0.0,
               keep_open_close = true },
  banjiao  = { compress_adjacent = false, ratio = 0.0, keep_open_close = true,
               fixed_ratio = 0.5, hang = true },
  kaiming  = { compress_adjacent = false, ratio = 0.0, keep_open_close = true,
               fixed_ratio = 0.5, fixed_full = true, hang = true },
  CCT      = { compress_adjacent = false, ratio = 0.0, keep_open_close = true,
               fixed_ratio = 0.7, hang = true },
  hangmobanjiao = { compress_adjacent = false, ratio = 0.0,
                    keep_open_close = true, hang = true },
}

```

(*End of definition for* `LuaCJK.punct_styles`.)

`LuaCJK.punct_codes` The code points of the punctuation groups that lie inside the CJK ranges and the Basic Multilingual Plane, sorted. This is the list of marks that LuaCJK-listings gives a token of their own. The filter is the CJK range table rather than a range list of its own, so a new Unicode release reaches the character table and this list together, and the groups are the ones LuaCJK.punct_class reads.

```
function M.punct_codes ()
  local codes = {}
  for code in pairs(PUNCT_CLASS) do
    codes[code] = true
  end
  for code in pairs(M.charsets.middle) do
```

```
    codes[code] = true
  end
  for code, group in pairs(M.punct_override) do
    codes[code] = group or nil
  end
  local list = {}
  for code in pairs(codes) do
    if code <= 0xFFFF and in_ranges(code, CJK_RANGES) then
      list[#list + 1] = code
    end
  end
  table.sort(list)
  return list
end
```

(*End of definition for* `LuaCJK.punct_codes`.)

`LuaCJK.define_punct_style` Register a style declared by `\LuaCJKDeclarePunctStyle`.

```
function M.define_punct_style (name, compress, ratio, keep)
  M.punct_styles[name] = {
    compress_adjacent = compress,
    ratio             = ratio,
    keep_open_close   = keep,
  }
end

```

(*End of definition for* `LuaCJK.define_punct_style`.)

`punct_style` Read the style name from the TEX token list and return the matching table, falling back to `plain`.

```
M.punct_width = {}
M.punct_kern  = {}

local function first_code (s)
  for _, code in utf8.codes(s) do
    return code
  end
end

function M.set_punct_width (chars, sp)
  for _, code in utf8.codes(chars) do
    M.punct_width[code] = sp
  end
end
```

```
2233
2234 function M.add_normal_spaced (ranges)
2235   for _, range in ipairs(ranges) do
2236     local first = range[1]
2237     local last  = range[2] or first
2238     for code = first, last do
2239       M.normal_spaced[code] = true
2240     end
2241   end
2242 end
2243
2244 function M.set_punct_kern (before, after, sp)
2245   local a, b = first_code(before), first_code(after)
2246   if a and b then
2247     M.punct_kern[a]    = M.punct_kern[a] or {}
2248     M.punct_kern[a][b] = sp
2249   end
2250 end
2251
2252 local function user_width (char)
2253   return M.punct_width[char]
2254 end
2255
2256 local function user_kern (before, after)
2257   local row = M.punct_kern[before]
2258   return row and row[after]
2259 end
2260
2261 function M.edit_punct_style (name, compress, ratio, keep)
2262   local style = M.punct_styles[name]
2263   if not style then
2264     return
2265   end
2266   if compress ~= nil then
2267     style.compress_adjacent = compress
2268   end
2269   if ratio ~= nil then
2270     style.ratio = ratio
2271   end
2272   if keep ~= nil then
2273     style.keep_open_close = keep
2274   end
2275 end
2276
2277 local function style_of (settings)
```

```
  local name = settings and settings.punct_style
  return M.punct_styles[name] or M.punct_styles.plain
end

```

(*End of definition for* `punct_style`.)

`LuaCJK.attr_disabled` The attribute number of the suspension switch.

```
M.attr_disabled = M.attr_disabled or nil

```

(*End of definition for* `LuaCJK.attr_disabled`.)

`disabled` Test the suspension attribute on a node.

```
local function disabled (n)
  if not (M.attr_disabled and n) then
    return false
  end
  local v = has_attribute(n, M.attr_disabled)
  return v ~= nil and v ~= 0
end

```

(*End of definition for* `disabled`.)

A.7 Node rewriting

A CJK glyph carries the identifier of the family that applies to it in an attribute. The pass below replaces it by a copy in that font, and takes a sub-block font or a fallback font instead when one applies.

`M.blocks` `M.block_order` `M.block_fonts` The declared sub-blocks in declaration order, and the font identifier of each block for each family. The order decides which block wins when two ranges overlap. A declaration extends the CJK class and not only the fonts of a family: a block may cover characters outside the built-in CJK ranges (the circled numbers U+2460 to U+2473 are the case), and they are typeset as CJK from then on, which is what lets the range reach a font at all. Cancelling a block removes its font and keeps the class, so its characters fall back to the main font of the family.

```
M.blocks      = {}
M.block_order = {}
M.block_fonts = {}

function M.declare_block (name, ranges)
  if not M.blocks[name] then
```

```
    table.insert(M.block_order, name)
  end
  M.blocks[name] = { ranges = ranges }
  BLOCK_CHARS    = { }
end

function M.set_block_font (main, name, fid)
  local row = M.block_fonts[main]
  if not row then
    row                 = {}
    M.block_fonts[main] = row
  end
  if fid and fid > 0 then
    row[name] = fid
  else
    row[name] = nil
  end
end

```

(*End of definition for* `M.blocks`, `M.block_order`, *and* `M.block_fonts`.)

`block_font` The font of the first block of ⟨*main*⟩ that contains ⟨*char*⟩ and is not canceled in ⟨*settings*⟩.

```
local function block_font (main, char, settings)
  local row = M.block_fonts[main]
  if not row then
    return nil
  end
  local off = settings and settings.blocks_off or {}
  for _, name in ipairs(M.block_order) do
    local block = M.blocks[name]
    if block and not off[name] and row[name] then
      for _, range in ipairs(block.ranges) do
        if char >= range[1] and char <= (range[2] or range[1]) then
          return row[name]
        end
      end
    end
  end
  return nil
end

```

(*End of definition for* `block_font`.)

`M.fallback` The fallback fonts of each family, as identifiers, in the order in which they were declared.

```
M.fallback = {}

function M.set_fallback (main, list)
  M.fallback[main] = list
end

```

(*End of definition for* `M.fallback`.)

`M.yoffset` The vertical offset of each fallback font, in scaled points, positive upwards. A fallback is chosen when the text is shaped, and only its identifier says which one applies, so its offset arrives with the identifier and `pre_shaping` adds it when it moves a glyph into that font. The text and the marks take their offset from the `shift` attribute instead: a font identifier is shared by every declaration that supplies the same font, so the table cannot tell two of them apart.

```
M.yoffset = {}

function M.set_yoffset (fid, sp)
  M.yoffset[fid] = sp
end

```

(*End of definition for* `M.yoffset`.)

`has_char` Whether ⟨*fid*⟩ has a glyph for ⟨*char*⟩. The answer is cached per font and code point, because the pass asks it once per glyph. The cache, keyed by the font identifier and the code point packed into one number.

```
local has_char_cache = {}

local function has_char (fid, char)
  local key    = fid * 0x200000 + char
  local cached = has_char_cache[key]
  if cached ~= nil then
    return cached
  end
  local f             = font.getfont(fid)
  local ok            = f and f.characters and f.characters[char] ~= nil
  has_char_cache[key] = ok
  return ok
end

```

(*End of definition for* `has_char` *and* `has_char_cache`.)

`fallback_font` The first fallback font of ⟨*fid*⟩ that has ⟨*char*⟩, or nothing when the feature is off or no fallback covers the character.

```
local function fallback_font (fid, char, settings)
  if not (settings and settings.auto_fallback) then
    return nil
  end
  local list = M.fallback[fid]
  if not list then
    return nil
  end
  for _, candidate in ipairs(list) do
    if has_char(candidate, char) then
      return candidate
    end
  end
end

```

(*End of definition for* `fallback_font`.)

`target_font` The font a CJK glyph takes: the family recorded on the glyph, then a sub-block font when one is declared for its block, then the punctuation family when the character is a mark it covers, then a fallback font when the font that applies does not have the character. The second value names the step that decided the font, because the step decides where the `Baseline` offset comes from as well.

```
local function target_font (n)
  if not M.attr_font then
    return nil
  end
  local target = has_attribute(n, M.attr_font)
  if not (target and target > 0) then
    return nil
  end
  local kind     = "family"
  local settings = settings_of(n)
  local block    = block_font(target, n.char, settings)
  if block then
    target, kind = block, "block"
  end
  if M.attr_punct_font and punct_class(n.char, nil, n.font) then
    local punct = has_attribute(n, M.attr_punct_font)
    if punct and punct > 0 then
      target, kind = punct, "punct"
    end
```

```
  end
  if not has_char(target, n.char) then
    local fallback = fallback_font(target, n.char, settings)
    if fallback then
      target, kind = fallback, "fallback"
    end
  end
  return target, kind
end

```

(*End of definition for* `target_font`.)

`shift_of` `attribute_shift` `apply_shift`

The vertical offset a CJK glyph takes, in scaled points. The text and the marks read the attribute their family filled, and a sub-block or fallback font reads the table that arrived with its identifier; an attribute the node does not carry, or one set to zero, is no offset at all. `apply_shift` adds that offset to the glyph and records the addition in the `shifted` attribute. The usual path meets a glyph once, but a node can reach a second list, as one does when a packed box is unboxed into another, and the record is what keeps a second visit from adding the offset twice.

```
local function attribute_shift (n, attr)
  if not attr then
    return nil
  end
  local sp = has_attribute(n, attr)
  if sp and sp ~= 0 then
    return sp
  end
end

local function shift_of (n, kind, target)
  if kind == "family" then
    return attribute_shift(n, M.attr_shift)
  elseif kind == "punct" then
    return attribute_shift(n, M.attr_punct_shift)
  end
  return M.yoffset[target]
end

local function apply_shift (n, kind, target)
  if M.attr_shifted and has_attribute(n, M.attr_shifted) then
    return
  end
  local sp = shift_of(n, kind, target)
```

```
  if sp then
    n.yoffset = (n.yoffset or 0) + sp
    if M.attr_shifted then
      set_attribute(n, M.attr_shifted, 1)
    end
  end
end

```

(*End of definition for* `shift_of`, `attribute_shift`, *and* `apply_shift`.)

`placeholder` Whether a glyph is the one luaotfload substitutes for a character the font does not have: the box it draws, or the slot it leaves empty.

```
local function placeholder (c)
  return c == 0 or (c >= 0xF0000 and c <= 0x10FFFD)
end

```

(*End of definition for* `placeholder`.)

`pre_shaping` Set each CJK glyph in the font that sets it, before the list is shaped, and let a character that attaches to the one in front of it follow that font. The substitution comes before shaping because shaping is what attaches a mark to its base and what a variation selector and a joiner act on; a glyph copied into another font afterwards keeps the form of the font it was shaped in. A glyph already in its font keeps it, and the font is the only thing the pass leaves alone: the `Baseline` offset is added whether the glyph has to be moved or is already set in its family's font, because a document that selects that font itself through `\fontspec` puts the characters there before the pass runs. The addition is recorded in the `shifted` attribute, so a glyph whose node reaches a second list, as one does when a packed box is unboxed into another, takes the offset once. The shift is the one the glyph carries, or the one its fallback font arrived with, so a family selected inside a group moves only its own characters. The joiner or selector is marked, because shaping may leave a space glyph and a negative kern where the font had no use for the character, and `drop_orphans` cannot recognize them afterwards. A CJK character that its CJK font does not have is remembered, because the box luaotfload leaves for it is removed only where the package moved the character into that font: a document with no CJK font at all still shows the box, which is what tells the reader that nothing sets the text.

```
local missing_chars = {}

local function pre_shaping (head)
  local last = nil
```

```
  local cur  = head
  while cur do
    if cur.id == GLYPH then
      if is_attached(cur.char) then
        if M.attr_attached then
          set_attribute(cur, M.attr_attached, 1)
        end
        if last then
          cur.font = last.font
        end
      elseif is_cjk(cur.char, settings_of(cur), cur.font) then
        local target, kind = target_font(cur)
        if target and target > 0 then
          apply_shift(cur, kind, target)
          if target ~= cur.font then
            if not has_char(target, cur.char) then
              missing_chars[target * 0x200000 + cur.char] = true
            end
            cur.font = target
          end
        end
      end
      last = cur
    end
    cur = cur.next
  end
  return head
end

M.pre_shaping = pre_shaping

```

(*End of definition for* `pre_shaping`.)

`drop_orphans` Remove what shaping left of the two kinds of character that must not reach the spacing pass. A marked joiner or selector has done its work by now: the font either used it, in which case it is a ligature or a variant glyph and the node is gone, or it left a space glyph whose advance a negative kern cancels. Both the glyph and the kern are freed, so the text around it closes up and the two CJK characters on its sides take their glue. A CJK character that its CJK font does not have leaves the box glyph the font loader draws; it is freed as well, because the character was moved into a font that does not set it rather than left in the text font.

```
local function drop_orphans (head)
```

```
  local prev = nil
  local cur  = head
  while cur do
    local nxt  = cur.next
    local drop = false
    if cur.id == GLYPH and M.attr_attached
       and has_attribute(cur, M.attr_attached) ~= nil then
      drop = true
      if nxt and nxt.id == KERN and nxt.kern == -(cur.width or 0) then
        local after = nxt.next
        free_node(nxt)
        nxt = after
      end
    elseif cur.id == GLYPH and placeholder(cur.char)
       and prev and prev.id == GLYPH
       and missing_chars[prev.font * 0x200000 + prev.char] then
      drop = true
    end
    if drop then
      if prev then
        prev.next = nxt
      else
        head = nxt
      end
      free_node(cur)
      cur = nxt
    else
      prev = cur
      cur  = nxt
    end
  end
  return head
end

```

(*End of definition for* `drop_orphans`.)

`copy_glyph` Copy a glyph node, changing only its font. Only the fallback for a luaotfload without `pre_shaping_filter` uses it. The `Baseline` offset has already been added to the node by `apply_shift`, so the copy carries it and the record in the `shifted` attribute with the other fields.

```
local function copy_glyph (n, fid)
  local g = new_node("glyph")
  g.font = fid
  for _, field in ipairs {
```

```
    "char",     "attr",    "lang",
    "left",     "right",   "uchyph",
    "xoffset",  "yoffset", "expansion_factor",
    "data",     "subtype",
  } do
    g[field] = n[field]
  end
  return g
end
```

(*End of definition for* `copy_glyph`.)

`rewrite_fonts` Replace each CJK glyph by a copy in the font its attribute names, after shaping. This is the fallback for a luaotfload without `pre_shaping_filter`; the package then behaves as it did before the substitution moved in front of shaping, and every zero-width character is removed at the input, so no attached character reaches this point. The `Baseline` offset is added first, because here it is this pass and not `pre_shaping` that meets the glyph; a glyph already in its font keeps it and takes the offset all the same. A glyph that the old font did not have may sit next to the placeholder luaotfload emitted for it; the placeholder is freed at the same time, because it would otherwise add an advance of its own.

```
local function rewrite_fonts (head)
  local prev = nil
  local cur  = head
  while cur do
    local nxt = cur.next
    if cur.id == GLYPH and is_cjk(cur.char, settings_of(cur), cur.font) then
      local target, kind = target_font(cur)
      if target and target > 0 then
        apply_shift(cur, kind, target)
        if target ~= cur.font then
          if nxt and nxt.id == GLYPH and nxt.font == cur.font
             and placeholder(nxt.char) then
            local after = nxt.next
            free_node(nxt)
            nxt = after
          end
          local g = copy_glyph(cur, target)
          g.next  = nxt
          if prev then
            prev.next = g
          else
            head = g
```

```
            end
            free_node(cur)
            cur = g
          end
        end
      end
      prev = cur
      cur  = nxt
    end
    return head
end
```

(*End of definition for* `rewrite_fonts`.)

`class_of_char` Map a character to `CJK` or `LATIN`, with the font the glyph was set in, which `\LuaCJK.is_cjk` needs for the code points below U+0100.

```
local function class_of_char (c, s, f)
  if is_cjk(c, s, f) then
    return "CJK"
  end
  return "LATIN"
end
```

(*End of definition for* `class_of_char`.)

`class_of_node` Return the class at both edges of a node. A box carries the class of the text nearest its edge, so a kern, a spacing glue, or a rule between the last character and the edge does not hide it; a filling glue does, since the text does not reach the edge of a box it stretches, and that is what keeps `\numberline`'s `\hfil` from spacing a CJK title away from the English ones. A box that holds no text and has ink of its own is a Latin atom, because an object box is not CJK material; a box of no size is transparent, and formulas count as Latin. A vertical box is read the same way, whichever of its edges faces the text: an underline is a vertical box whose last node is the rule, and the text it lies under is what the edge holds. A box the edge reaches is read in turn, and when it has no class itself the text behind it is the text nearest the edge: the frame that xcolor draws over a box as a second, classless sibling does not hide the content it encloses. A paragraph line begins with the marker TEX writes for its paragraph settings, `local_par`, which is passed over in the same way, so the first character of a `\parbox` reaches the left edge; the line ends in `\parfillskip`, a filling glue, so the text does not reach the right edge and the box takes no glue there. Whether a node carries neither text

`transparent_node`

nor advance, so that the text reaches the edge past it. A kern of no width, such as the italic correction that follows the last letter of an italic box, carries no advance either.

```
local function transparent_node (n)
  return n.id == WHATSIT or n.id == MARK or n.id == INS
      or n.id == ADJUST or n.id == PENALTY
      or (n.id == KERN and n.kern == 0)
end

```

box_visible

Whether a box occupies an advance of its own. A box that occupies none is passed over, so the boundary rules read the text on both sides of it as one run: an empty box does not break the run, and the ink of `\rlap`, `\llap` and `\clap` lies on the text around it instead of separating it.

```
local function box_visible (n)
  return n.width ~= 0
end

```

skippable

Whether the text reaches past a node: a node that carries neither text nor advance, or a box that occupies none. The spacing pass walks back over such a node to the material in front of it, so an empty box is not a boundary of its own and a source space beside it is the space of the boundary it stands in.

```
local function skippable (n)
  return transparent_node(n)
      or ((n.id == HLIST or n.id == VLIST) and not box_visible(n))
end

```

attached_box

Whether a box belongs to the character in front of it. TEX writes the shift of a raised or lowered box into the box node, and `\textsuperscript` and `\textsubscript` build one, so the baseline of such a box is not the baseline of the text around it. A mark is often wrapped in a box that is not shifted itself, which is what `\@makefnmark` and `\raisebox` produce: the outer box has no shift, and everything it holds is raised or lowered. Such a box belongs to the character in front of it just the same, because it holds no text on the baseline its own shift speaks of. A wrapped mark is a line of text: the box that `\textsuperscript` or `\raisebox` shifts holds characters, and so does a footnote mark. A frame is not, because what it shifts is a vertical box: `\fbox`, `\framebox` and `\fcolorbox` lower the frame so that its content stands on the baseline, and a box that wraps a vertical list is a block of the line rather than a mark of the character in front of it.

Such a box keeps its boundary glue, and the break that glue offers, which is what xeCJK gives a frame as well.

```
local function raised_only (list)
  local found = false
  local n = list
  while n do
    if n.id == GLYPH or n.id == MATH then
      return false
    elseif n.id == HLIST or n.id == VLIST then
      if n.shift == 0 and not raised_only(n.head) then
        return false
      elseif n.shift ~= 0 and n.head and n.head.id == VLIST then
        return false
      end
      found = true
    end
    n = n.next
  end
  return found
end

local function attached_box (n)
  return (n.id == HLIST or n.id == VLIST)
     and (n.shift ~= 0 or raised_only(n.head))
end

```

edge_inked
inked_rule

Whether the edge of a list carries ink: a glyph or a formula does, a rule that has a size does, and a box does when the edge of its own content does. A glue, a kern, and the other nodes that paint nothing do not, and they do not hide the ink in front of them either, since the question is only whether ink ends the box.

```
local function inked_rule (n)
  return n.height > 0 or n.depth > 0 or n.width > 0
end

local function edge_inked (list)
  local ink = false
  local n   = list
  while n do
    if n.id == GLYPH or n.id == MATH then
      ink = true
    elseif n.id == RULE then
      if inked_rule(n) then
        ink = true
```

```
      end
    elseif n.id == HLIST or n.id == VLIST then
      ink = edge_inked(n.head)
    end
    n = n.next
  end
  return ink
end
```

first_punct

The punctuation group of the first visible character of a list, the character itself, whether ink stands in front of it inside the box, and whether a character was found at all, so that a box with no text can be skipped. A formula counts as material with no mark.

```
local function first_punct (list)
  local ink = false
  local n   = list
  while n do
    if n.id == GLYPH then
      return punct_class(n.char, settings_of(n), n.font), n, ink, true
    elseif n.id == HLIST or n.id == VLIST then
      local p, g, i, found = first_punct(n.head)
      if found then
        return p, g, ink or i, true
      end
      ink = ink or edge_inked(n.head)
    elseif n.id == MATH then
      return nil, nil, ink, true
    elseif n.id == RULE and inked_rule(n) then
      ink = true
    end
    n = n.next
  end
  return nil, nil, ink, false
end
```

last_punct

The same for the last visible character of a list, and whether ink follows it inside the box.

```
local function last_punct (list)
  local n     = list
  local punct = nil
  local glyph = nil
  local ink   = false
```

```
  local found = false
  while n do
    if n.id == GLYPH then
      punct = punct_class(n.char, settings_of(n), n.font)
      glyph = n
      ink   = false
      found = true
    elseif n.id == HLIST or n.id == VLIST then
      local p, g, i, deep = last_punct(n.head)
      if deep then
        punct = p
        glyph = g
        ink   = i
        found = true
      else
        ink = ink or edge_inked(n.head)
      end
    elseif n.id == MATH then
      punct = nil
      glyph = nil
      ink   = false
      found = true
    elseif found and n.id == RULE and inked_rule(n) then
      ink = true
    end
    n = n.next
  end
  return punct, glyph, ink, found
end

local function filling_glue (n)
  return n.id == GLUE
     and ((n.stretch_order or 0) ~= 0 or (n.shrink_order or 0) ~= 0)
end

local function edge_transparent (n)
  return transparent_node(n) or n.id == KERN or n.id == RULE
      or n.id == LOCAL_PAR
      or (n.id == GLUE and not filling_glue(n))
end

local function has_text (list)
  local n = list
  while n do
```

```
    if n.id == GLYPH or n.id == MATH then
      return true
    elseif n.id == HLIST or n.id == VLIST then
      if has_text(n.head) then
        return true
      end
    end
    n = n.next
  end
  return false
end

local function first_box_class (list)
  local n = list
  while n do
    if n.id == GLYPH then
      return class_of_char(n.char, settings_of(n), n.font)
    elseif n.id == HLIST or n.id == VLIST then
      local c = first_box_class(n.head)
      if c then
        return c
      end
    elseif n.id == MATH then
      return "LATIN"
    elseif filling_glue(n) then
      return nil
    elseif not edge_transparent(n) then
      return nil
    end
    n = n.next
  end
  return nil
end

local function last_box_class (list)
  local n    = list
  local last = nil
  while n do
    if n.id == GLYPH then
      last = class_of_char(n.char, settings_of(n), n.font)
    elseif n.id == HLIST or n.id == VLIST then
      local c = last_box_class(n.head)
      if c then
        last = c
      end
```

```
    elseif n.id == MATH then
      last = "LATIN"
    elseif filling_glue(n) then
      last = nil
    elseif not edge_transparent(n) then
      last = nil
    end
    n = n.next
  end
  return last
end

local function class_of_node (n)
  if n.id == GLYPH then
    local c = class_of_char(n.char, settings_of(n), n.font)
    return c, c
  elseif n.id == HLIST or n.id == VLIST then
    local first = first_box_class(n.head)
    local last  = last_box_class(n.head)
    if not first and not last
       and (n.height ~= 0 or n.depth ~= 0) and not has_text(n.head) then
      return "LATIN", "LATIN"
    end
    return first, last
  elseif n.id == MATH then
    return "LATIN", "LATIN"
  end
  return nil, nil
end

```

(*End of definition for* `class_of_node` *and others.*)

`is_inserted` Test whether a node was inserted by LuaCJK.

```
local function is_inserted (n)
  return M.attr_inserted
    and has_attribute(n, M.attr_inserted) ~= nil
end

```

(*End of definition for* `is_inserted`.)

`inherit_attributes` Give a new node the attribute list of the node it is spliced next to. LuaCJK creates glue, kerns, and penalties after TEX has attached the surrounding attributes to the text, so a package that marks spans with attributes, such as lua-ul, would

otherwise read the new node as the end of its span and start a new one. Assigning to the ⟨*attr*⟩ field is reference counted, and later per-attribute changes are copy-on-write, so the source node is never modified.

```
local function inherit_attributes (new, source)
  if source and source.attr then
    new.attr = source.attr
  end
end

```

(*End of definition for* `inherit_attributes`.)

`inherit_common_attributes` Give a new node the attributes of ⟨*left*⟩ that ⟨*right*⟩ carries as well, with the same value. A boundary between two characters belongs to neither of them: inside a span both carry the attribute that marks it, and the span stays continuous, while at its edge only one does, and a package that marks spans with attributes reads the new node as the end of the span. The attributes of this package are inherited as they were, because they describe how the text is set and the glue between two characters takes the policy of the one in front of it. The attributes another package registered are compared one by one: the list a node carries cannot be enumerated through the interface, so the registered numbers are the ones this can see. Assigning to the ⟨*attr*⟩ field is reference counted and later per-attribute changes are copy-on-write, so neither neighbour is modified.

```
local function inherit_common_attributes (new, left, right)
  inherit_attributes(new, left)
  if not (left and right and luatexbase.attributes) then
    return
  end
  for _, id in pairs(luatexbase.attributes) do
    if node.has_attribute(left, id) ~= node.has_attribute(right, id) then
      node.unset_attribute(new, id)
    end
  end
end

```

(*End of definition for* `inherit_common_attributes`.)

`make_glue` Create a glue node from a skip register, inherit the attributes of ⟨*source*⟩, and mark the node as inserted. An optional ⟨*other*⟩ gives the character on the other side of the boundary, and the node then keeps only the attributes both sides carry. The node keeps the stretch and shrink orders of the register.

```
local function make_glue (spec, source, other)
```

```
  if not spec then
    return nil
  end
  local s = spec
  local g = new_node("glue")
  if other then
    inherit_common_attributes(g, source, other)
  else
    inherit_attributes(g, source)
  end
  g.width         = s.width
  g.stretch       = s.stretch
  g.shrink        = s.shrink
  g.stretch_order = s.stretch_order or 0
  g.shrink_order  = s.shrink_order  or 0
  if M.attr_inserted then
    set_attribute(g, M.attr_inserted, 1)
  end
  return g
end

```

(*End of definition for* `make_glue`.)

`set_glue` Give an existing glue node the dimensions of a skip register, including its stretch and shrink orders.

```
local function set_glue (g, spec)
  if not spec then
    return false
  end
  local s   = spec
  g.width   = s.width
  g.stretch = s.stretch
  g.shrink  = s.shrink
  g.stretch_order = s.stretch_order or 0
  g.shrink_order  = s.shrink_order  or 0
  if M.attr_inserted then
    set_attribute(g, M.attr_inserted, 1)
  end
  return true
end

```

(*End of definition for* `set_glue`.)

make_kern Create a kern node, inherit the attributes of ⟨*source*⟩, and mark the node as inserted. An optional ⟨*other*⟩ gives the character on the other side of the boundary, and the node then keeps only the attributes both sides carry.

```
local function make_kern (width, source, other)
  local k = new_node("kern")
  if other then
    inherit_common_attributes(k, source, other)
  else
    inherit_attributes(k, source)
  end
  k.kern = width
  if M.attr_inserted then
    set_attribute(k, M.attr_inserted, 1)
  end
  return k
end

```

(*End of definition for* `make_kern`.)

A.8 The passes

spacing Walk a list and insert the glue between CJK and Latin material. The walk tracks the last visible node and the class at its end, so that a formula acts as one Latin atom at its left edge; inside the formula the boundary rules are not applied. A box is an atom as well, and the class at its edge is the class of the text nearest that edge: a kern, a spacing glue, or a rule between the last character and the edge does not hide it, so the boundary glue is added by the class of the text itself and not by a skip the document wrote inside the box. A filling glue does hide the text, because the text does not reach the edge of a box it stretches, and that is what keeps a `\numberline`'s `\hfil` from spacing a CJK title away from an English one. A box that holds no text and has ink is a Latin atom, so a rule box takes the glue on both sides. A box that occupies no advance is passed over, so an empty box inside CJK text does not break the run, and the ink of an overlaid one does not split it; a source space beside it is the space of the boundary it stands in, and the pass walks back over the box to it. A box whose own baseline TEX has shifted is attached to the character in front of it: the run ends at it, so it takes no glue on either side and offers no break. That is the box `\textsuperscript` and `\textsubscript` build; the shift is the box's own field, so the outer box of a `\raisebox` or of a footnote mark keeps its boundary. The snapshot a boundary reads is the one the CJK character it touches carries (`\boundary_settings`), so the CJK–Latin glue is the CJK text's and a Latin run set at another size keeps it the

same on both of its sides. A penalty that prohibits the break is passed over the same way, because the glue of that boundary belongs after it; a smaller penalty, and a discretionary, end the run. Decide what to do at the boundary between two visible nodes. Only source-generated glue is a candidate for replacement; other glue is left alone, and a discretionary is opaque, so neither is bridged. A penalty is read by its value. One that prohibits the break is not a barrier: the glue goes after it, so the prohibition still holds at the break the glue offers. A smaller penalty prices the break instead of forbidding it, and the glue would offer the same place a second break at no cost, so a priced break is left alone. A source space between the two characters is the boundary itself, so when one stands in front of a prohibiting penalty it is converted, and the penalty moves in front of it rather than letting the space offer the break. A whatsit, a mark, an insertion, or an adjustment between that space and the character is passed over: a color change in front of the character does not hide the space, which stays the boundary and is never joined by a second glue. A penalty ends that search, because it has a boundary rule of its own. A box that occupies no advance is passed over the same way, so a source space in front of an empty box is the boundary of the characters the box stands between, and no glue is spliced behind the box. A boundary that touches a suspended run takes the glue its two classes give, because `Verb` leaves the material alone and not the text around it: the edge is spaced as the same characters would be in a text font, and the interior keeps its own advance. The mark rules are applied by hand there, because the kinsoku pass cannot see a prohibition across a suspended node: no break follows an opening mark, none precedes a closing, middle, or long mark that begins a run, and two Latin words are never parted. The kern between two adjacent punctuation marks, in scaled points, or nothing when the two are not a pair the style spaces. A length the document gave through `\LuaCJKsetkern` is the blank between the two inks, which the outlines give; without one the style's ratio decides, and it never takes more than the right blank of the mark it compresses, so a mark whose ink nearly fills its advance, such as the fullwidth percent sign, keeps its ink whole. A mark whose advance belongs to the font is never compressed.

pair_kern

```
local function pair_kern (prev, cur, settings, style)
  if not (prev and cur and prev.id == GLYPH and cur.id == GLYPH) then
    return nil
  end
  local before = punct_class(prev.char, settings, prev.font)
  local after  = punct_class(cur.char, settings, cur.font)
  if not before or not after or inert_mark(before) or inert_mark(after) then
    return nil
  end
  local explicit = user_kern(prev.char, cur.char)
```

```
  if explicit then
    local _, prev_rb = M.mark_blanks(prev)
    local cur_lb     = M.mark_blanks(cur)
    if prev_rb and cur_lb then
      return explicit - (prev_rb + cur_lb)
    end
    return explicit
  end
  if not style.compress_adjacent then
    return nil
  end
  local exempt = (style.keep_open_close
      and before == "open" and after == "close")
    or (M.charsets.long[prev.char] and M.charsets.long[cur.char])
  if exempt or not prev.width or prev.width <= 0 then
    return nil
  end
  local cut = math.floor(style.ratio * prev.width + 0.5)
  local _, prev_rb = M.mark_blanks(prev)
  if prev_rb then
    cut = math.min(cut, prev_rb)
  end
  if cut <= 0 then
    return nil
  end
  return -cut
end
```

The punctuation class a formula takes at its edge, or nothing. A formula that sets one opening or closing delimiter alone is that bracket: the math pass marked its only character, and the converted list holds that character between the two math nodes, so each of the two reads the same mark. A formula that holds anything else is not a bracket and takes no class.

math_bracket

```
local function math_bracket (n)
  if not (n and n.id == MATH and M.attr_math_bracket) then
    return nil
  end
  local glyph = nil
  if n.subtype == 0 then
    glyph = n.next
    if not (glyph and glyph.id == GLYPH and glyph.next
            and glyph.next.id == MATH and glyph.next.subtype == 1) then
      glyph = nil
    end
```

```
  elseif n.subtype == 1 then
    glyph = n.prev
    if not (glyph and glyph.id == GLYPH and glyph.prev
            and glyph.prev.id == MATH and glyph.prev.subtype == 0) then
      glyph = nil
    end
  end
  if not glyph then
    return nil
  end
  local value = has_attribute(glyph, M.attr_math_bracket)
  if value == 1 then
    return "half-open"
  elseif value == 2 then
    return "half-close"
  end
  return nil
end

```

The punctuation class of the node that faces the boundary: the class of the glyph, the bracket a formula is, or the class of the text nearest the edge of a box.

edge_punct

```
local function edge_punct (n, at_end, settings)
  if not n then
    return nil
  end
  if n.id == GLYPH then
    return punct_class(n.char, settings, n.font)
  elseif n.id == MATH then
    return math_bracket(n)
  elseif n.id == HLIST or n.id == VLIST then
    if at_end then
      return last_punct(n.head)
    end
    return first_punct(n.head)
  end
  return nil
end

```

Decide the glue a boundary takes, so that the ordinary boundary and the edge of suspended material are read the same way. A CJK character next to another takes `CJKglue`, and next to Latin material `CJKecglue`; a character of the normal-spaced set takes neither, and neither does the boundary on either side of it. A formula that is one delimiter is a bracket, and `\edge_punct` answers with that

boundary_spec

class, so it is read like the half-width bracket below. A run of dashes or ellipses is one unit: the marks are joined without glue, so that justification cannot pull a two-em dash apart, and the absence of a break opportunity replaces the explicit prohibition. A punctuation mark takes no CJK–Latin advance on either side: the advance of the mark already carries its spacing, and the glue would add a third of an em that the typography does not ask for. The boundary between two marks keeps a glue of zero width instead, so that it stays a break opportunity and the kinsoku penalty can forbid the break. A half-width bracket is the exception, and its two sides are read differently. The pair takes no glue inside, so it hugs the text it encloses and offers no break there, while its outside takes the CJK–Latin glue the two classes give it, in front of an opening bracket and behind a closing one. A closing bracket takes that glue only in front of a CJK character: a Latin letter stays against it, and a mark behind it keeps the zero-width boundary glue of a mark. That is how xeCJK sets its half-width marks, whose outside is spaced on both sides. Nothing is returned when the two classes take no glue at all.

```
M.normal_spaced = {}

function M.set_normal_spaced_chars (chars)
  local set = {}
  for _, code in utf8.codes(chars) do
    set[code] = true
  end
  M.normal_spaced = set
end

local function normal_spaced (n)
  return n and n.id == GLYPH and M.normal_spaced[n.char] == true
end

local function long_punct (n)
  return n and n.id == GLYPH and M.charsets.long[n.char] == true
end

local PUNCT_BOUNDARY_GLUE = { width = 0, stretch = 0, shrink = 0 }

local function boundary_spec (prev_visible, prev_end, cur, cur_start, settings)
  if not prev_end or not cur_start then
    return nil
  end
  if long_punct(prev_visible) and long_punct(cur) then
    return nil
  end
  local prev_punct = edge_punct(prev_visible, true, settings)
```

```
  local cur_punct  = edge_punct(cur, false, settings)
  if prev_end == "CJK" and cur_start == "CJK" then
    return settings.ccglue
  end
  if prev_end ~= cur_start
     and not normal_spaced(prev_visible) and not normal_spaced(cur) then
    if half_mark(prev_punct) then
      if prev_punct ~= "half-close" or cur_punct or cur_start ~= "CJK" then
        return nil
      end
      return settings.ecglue
    elseif cur_punct == "half-open" then
      return prev_punct and PUNCT_BOUNDARY_GLUE or settings.ecglue
    elseif half_mark(cur_punct) then
      return nil
    elseif prev_punct or cur_punct then
      return PUNCT_BOUNDARY_GLUE
    else
      return settings.ecglue
    end
  end
  return nil
end

```

insert_boundary_glue

Insert the glue `\boundary_spec` gives, and the kern that spaces two adjacent punctuation marks. The kern is the one `\pair_kern` gives, and the glue follows it, so the boundary keeps its break opportunity for the kinsoku penalty to forbid and its share of the stretch a justified line needs. The character on each side is the one the list shows, so a whatsit or a penalty between two characters does not hide them from these rules; the glue itself follows the node that carries the break, which may be such a whatsit or penalty. The glue and the kern keep only the attributes both characters carry, so a decoration ends at the last character it covers rather than one glue further. The node the glue goes in front of is the character behind the boundary, unless that character is drawn over a background: `\colorbox` puts a box of no advance in front of its content, holding a rule of the height, depth, and width of that content, and the ink of such a background has to move with the text it covers. The glue therefore goes in front of the background, and the highlight stops at the content on both sides; a box of no advance that draws its own ink, such as `\rlap`, is not a background and stays where the document put it. The nodes are read forward from the character in front, because the pass that removes what shaping left of a joiner or a variation selector relinks the

list without touching the backward links.

```
local function background_of (n, cur)
  if not cur or (cur.id ~= HLIST and cur.id ~= VLIST) then
    return false
  end
  if (n.id ~= HLIST and n.id ~= VLIST) or box_visible(n)
     or n.height ~= cur.height or n.depth ~= cur.depth then
    return false
  end
  local m = n.head
  while m do
    if m.id == RULE and m.width == cur.width then
      return true
    end
    m = m.next
  end
  return false
end

local function boundary_start (prev_visible, prev, cur)
  local before = prev_visible or prev
  local n      = before and before.next or nil
  while n and n ~= cur do
    if background_of(n, cur) then
      return n, before
    end
    before = n
    n      = n.next
  end
  return cur, prev
end

local function insert_boundary_glue (head, prev, prev_visible, prev_end,
                                     cur, cur_start, settings)
  if not prev_end or not cur_start then
    return head
  end
  local style = style_of(settings)
  local prev_punct = edge_punct(prev_visible, true, settings)
  local cur_punct  = edge_punct(cur, false, settings)
  local prev_mark = prev_punct and prev.id == GLYPH
      and not half_mark(prev_punct)
  local cur_mark  = cur_punct and cur.id == GLYPH
      and not half_mark(cur_punct)
  local spec = boundary_spec(prev_visible, prev_end, cur, cur_start, settings)
```

```
  local start, before = boundary_start(prev_visible, prev, cur)

  if prev_mark and cur_mark then
    local kern = pair_kern(prev_visible, cur, settings, style)
    if kern and kern ~= 0 then
      local k = make_kern(kern, prev_visible or prev, cur)
      local g = spec and make_glue(spec, prev_visible or prev, cur) or nil
      k.next  = g or start
      if g then
        g.next = start
      end
      if before then
        before.next = k
      else
        head = k
      end
      return head
    end
  end

  if not spec then
    return head
  end
  local g = make_glue(spec, prev_visible or prev or cur, cur)
  if not g then
    return head
  end
  g.next = start
  if before then
    before.next = g
  else
    head = g
  end
  return head
end

```

restore_source_spaces

Give a source space that a text font command moved across the italic correction its subtype back. `\textbf` and the other text font commands save the skip in front of them, remove it, apply the italic correction, and put it back with `\hskip`, so the glue keeps the dimensions of the space but not its subtype. The move is the only way a normal glue comes to follow an italic-correction kern, and the subtype is the one thing the spacing pass reads, so putting the subtype back is enough; a typeface whose interword space is rigid therefore needs no special case, at the

price of retiming a skip that the document wrote in that very place. A space never has a stretch or shrink order, so a glue that has one was not a space at all: `\hfill` and `\hfil` follow the italic correction of the same command and keep their own subtype, or the fill of the line would become the inter-character glue.

```
local function restore_source_spaces (head)
  local prev = nil
  local cur  = head
  while cur do
    if cur.id == GLUE and cur.subtype ~= SPACESKIP
       and cur.stretch_order == 0 and cur.shrink_order == 0
       and prev and prev.id == KERN and prev.subtype == ITALICCORR then
      cur.subtype = SPACESKIP
    end
    prev = cur
    cur  = cur.next
  end
  return head
end

```

Whether a space was written with `\ ` or `\nobreakspace`: the attribute set around that token is the only record of it.

explicit_space

```
local function explicit_space (n)
  return M.attr_explicit_space ~= nil
     and (has_attribute(n, M.attr_explicit_space) or 0) ~= 0
end

```

Give up the advance of a source space that stands in front of another one, with nothing visible between the two. Two source spaces are one space, and TEX makes one glue of them when they are adjacent; a whatsit, a mark, or a box that occupies no advance is invisible to the boundary rules, so the collapse applies across it as well, and the space nearest the character is the one that carries the boundary. The advance is dropped rather than the node, so the boundary keeps the break opportunity of both spaces, and the node is marked as one this package has dealt with, so a list that is packed twice is not collapsed twice.

collapse_earlier_spaces

```
local function collapse_earlier_spaces (glue)
  local n = glue.prev
  while n do
    if n.id == PENALTY then
      return
    elseif skippable(n) then
      n = n.prev
```

```
    elseif n.id == GLUE and n.subtype == SPACESKIP
           and not explicit_space(n) and not is_inserted(n) then
      n.width   = 0
      n.stretch = 0
      n.shrink  = 0
      if M.attr_inserted then
        set_attribute(n, M.attr_inserted, 1)
      end
      n = n.prev
    else
      return
    end
  end
end

```

handle_source_space

Remove or retime a space typed in the source. Only a space that came from the source is a candidate; an explicit `\hskip` or `\hfill` has a different subtype and is left alone, and the space that a text font command moved across the italic correction has had its subtype put back above. A space between two CJK characters is not kept when `CJKspace` is off, but dropping it must not drop the boundary with it: the space becomes the inter-character glue of `CJKglue`. Freeing it instead would leave the two characters adjacent with no glue, and the pass that would add the glue runs after line breaking, so the break opportunity would be lost. The blank of a pair of marks rides in that glue as its width, so a source space between two marks no longer defeats their compression or the length `\LuaCJKsetkern` gives them: the two marks are spaced as they are when no space separates them, and a break at the boundary discards the blank with the glue. A run of long punctuation is the exception: `\boundary_spec` joins its marks without glue, and a source space between two of them adds neither an advance nor a stretch of its own, so a newline in the source cannot space the run apart. The node stays a break opportunity, and a blank `\LuaCJKsetkern` sets between the two marks is kept as its width. A space the document wrote with `\ ` or `\nobreakspace` is not a source space, and the attribute set around it is the only place that says so: the two are the same `spaceskip` glue, with the same dimensions, until that mark is read. Such a space keeps its advance, on either side of the boundary it stands at, because xeCJK sets it as a space too and `CJKspace` is about the source. A source space that another source space stands in front of, with nothing visible between the two, is the second of one run of spaces, so the one in front of it gives up its advance first.

```
local function handle_source_space (head, glue, prev_visible, prev_end, cur,
                                    cur_start, settings)
```

```
  if glue.subtype ~= SPACESKIP then
    return head
  end
  if explicit_space(glue) then
    return head
  end
  collapse_earlier_spaces(glue)
  local prev_cjk = prev_end == "CJK"
  local cur_cjk  = cur_start == "CJK"
  if prev_cjk and cur_cjk then
    local cjk_space = M.attr_cjk_space
      and (has_attribute(glue, M.attr_cjk_space) or 0) ~= 0
    if not cjk_space then
      local kern = pair_kern(prev_visible, cur, settings, style_of(settings))
      if long_punct(prev_visible) and long_punct(cur) then
        glue.width   = kern or 0
        glue.stretch = 0
        glue.shrink  = 0
      else
        set_glue(glue, settings.ccglue)
        if kern then
          glue.width = glue.width + kern
        end
      end
      if M.attr_inserted then
        set_attribute(glue, M.attr_inserted, 1)
      end
    end
  elseif prev_cjk or cur_cjk then
    local xecglue = M.attr_xecglue
      and (has_attribute(glue, M.attr_xecglue) or 0) ~= 0
    if xecglue then
      set_glue(glue, settings.ecglue)
    end
  end
  return head
end

```

Whether the break beside a suspended run is allowed: the mark rules of `forbids_break` are read at the edge of the run, and two Latin words are left whole.

suspended_break

```
local function suspended_break (prev_visible, cur, prev_end, start_class,
                                settings)
  if not prev_end or not start_class then
```

```
    return false
  end
  if normal_spaced(prev_visible) or normal_spaced(cur) then
    return false
  end
  local prev_mark = (prev_visible.id == GLYPH or prev_visible.id == MATH)
    and edge_punct(prev_visible, true, settings)
  local cur_mark = (cur.id == GLYPH or cur.id == MATH)
    and edge_punct(cur, false, settings)
  if prev_mark == "open" or prev_mark == "half-open" then
    return false
  end
  if cur_mark == "close" or cur_mark == "middle"
     or cur_mark == "half-close" or cur_mark == "latin-close" then
    return false
  end
  if (cur_mark == "dash" or cur_mark == "ellipsis")
     and not (prev_mark == "dash" or prev_mark == "ellipsis") then
    return false
  end
  if prev_end == "LATIN" and start_class == "LATIN"
     and not prev_mark and not cur_mark then
    return false
  end
  return true
end

local function handle_boundary (head, prev, prev_visible, prev_end,
                                cur, start_class, settings)
  if not prev_visible or not start_class then
    return head
  end
  if disabled(prev_visible) or disabled(cur) then
    local occupied = prev
      and (prev.id == GLUE or prev.id == KERN
           or prev.id == PENALTY or prev.id == DISC)
    if not occupied and disabled(prev_visible) ~= disabled(cur)
       and suspended_break(prev_visible, cur, prev_end, start_class, settings)
    then
      local source = disabled(cur) and prev_visible or cur
      local spec = boundary_spec(prev_visible, prev_end, cur, start_class,
                                 settings)
      local g = spec and make_glue(spec, source) or nil
      if g then
```

```
        g.next = cur
        if prev then
          prev.next = g
        else
          head = g
        end
      end
    end
    return head
  end
  if prev and prev.id == GLUE then
    if not is_inserted(prev) then
      return handle_source_space(
        head, prev, prev_visible, prev_end, cur, start_class, settings)
    end
  elseif prev and prev.id == PENALTY then
    if prev.penalty >= 10000 then
      local space, before = nil, nil
      local n = prev_visible
      while n and n ~= prev do
        local nxt = n.next
        if nxt and nxt.id == GLUE and nxt.subtype == SPACESKIP then
          space, before = nxt, n
          break
        end
        n = nxt
      end
      if space then
        head = handle_source_space(
          head, space, prev_visible, prev_end, cur, start_class, settings)
        local pred = before
        local m    = space
        while m and m ~= prev do
          pred = m
          m    = m.next
        end
        pred.next   = prev.next
        before.next = prev
        prev.next   = space
        return head
      end
      return insert_boundary_glue(
        head, prev, prev_visible, prev_end, cur, start_class, settings)
    end
  elseif prev and prev.id == DISC then
```

```
  elseif prev and skippable(prev) then
    local n = prev
    while n and n.id ~= PENALTY and skippable(n) do
      n = n.prev
    end
    if n and n.id == GLUE then
      if not is_inserted(n) then
        return handle_source_space(
          head, n, prev_visible, prev_end, cur, start_class, settings)
      end
      return head
    end
    return insert_boundary_glue(
      head, prev, prev_visible, prev_end, cur, start_class, settings)
  else
    return insert_boundary_glue(
      head, prev, prev_visible, prev_end, cur, start_class, settings)
  end
  return head
end

local function spacing (head)
  head = restore_source_spaces(head)
  local prev         = nil
  local prev_visible = nil
  local prev_end     = nil
  local math_level   = 0
  local cur          = head
  while cur do
    local nxt = cur.next
    if cur.id == MATH then
      if cur.subtype == 0 then
        if math_level == 0 then
          head = handle_boundary(
            head, prev, prev_visible, prev_end, cur, "LATIN",
            settings_of(prev_visible or cur))
        end
        math_level   = math_level + 1
        prev_visible = nil
        prev_end     = nil
      else
        if math_level > 0 then
          math_level = math_level - 1
        end
```

```
        if math_level == 0 then
          prev_visible = cur
          prev_end     = "LATIN"
        end
      end
    elseif math_level > 0 then
    else
      local start_class, end_class = class_of_node(cur)
      local visible
      if cur.id == HLIST or cur.id == VLIST then
        visible = box_visible(cur)
      else
        visible = start_class ~= nil
      end
      if visible and attached_box(cur) then
        prev_visible = nil
        prev_end     = nil
      elseif visible then
        head = handle_boundary(
          head, prev, prev_visible, prev_end, cur, start_class,
          boundary_settings(prev_visible, prev_end, cur, start_class))
        prev_visible = cur
        prev_end     = end_class
      elseif cur.id == PENALTY or cur.id == DISC then
        if not (cur.id == PENALTY and cur.penalty >= 10000) then
          prev_visible = nil
          prev_end     = nil
        end
      end
    end
    prev = cur
    cur  = nxt
  end
  return head
end

```

(*End of definition for* `spacing` *and others.*)

TEX may break at glue, at kern, and at a discretionary. The first two are places where LuaCJK inserts its own nodes, so after building the list a second pass puts a penalty 10000 in front of every break opportunity that must not be used. A discretionary states the value of its own break, so there the pass sets the node's penalty field rather than inserting a penalty. The hyphens of a word are

not in the list yet when the pass runs: TEX inserts them during line breaking, so a `hyphenate` callback forbids those as well.

apply_kinsoku Add the prohibitions where the settings of the text ask for them. The break points the pass reads are the glue LuaCJK inserts and the source spaces, because the punctuation rules are about those places alone; the mark is about every break, so a glue that carries it is read as a break point as well, which is what keeps a `\quad` or an `\hspace` inside the argument on the same line. The mark that counts is the document's own: a break is forbidden when the material on both sides of it lies in one span, or when a node the document wrote carries a mark itself. The identifier of a span is its own, so two arguments beside each other are two spans and the break between them stays, while the value agrees throughout one argument and a call inside another one joins the span already in force. The glue and the kerns the package inserts take the attributes of the character in front of them, so a mark there records the decoration a span carries and says nothing about the break, and the boundary between the argument and the text around it belongs to that text. A box is read at the edge that faces the break, the way the spacing pass reads it, so that a mark at the edge of the box is neither a line start nor a line end. The rule `CJLineBreak` adds belongs to the character that would start the line, so it reads that character's snapshot: a group that selects `strict` protects the small kana set inside it and no character outside.

Whether a node is marked by a break-prohibited span.

no_break_mark
```
local function no_break_mark (n)
  if not (M.attr_nobreak and n) then
    return false
  end
  local value = has_attribute(n, M.attr_nobreak)
  return value ~= nil and value ~= 0
end
```

Whether two nodes lie in one break-prohibited span: both are marked and the identifiers agree. The identifier is the one of the outermost `\LuaCJKnobreak` that marks the node, so two arguments beside each other are two spans, and an argument inside another one belongs to the span around it.

same_break_span
```
local function same_break_span (a, b)
  if not (no_break_mark(a) and no_break_mark(b)) then
    return false
  end
  return has_attribute(a, M.attr_nobreak) == has_attribute(b, M.attr_nobreak)
end
```

The glyph a discretionary is followed by: the letters of a word are separated from
next_glyph it by a kern or a ligature at most, and a glue, a penalty, another discretionary or a formula ends the word.

```
local function next_glyph (n)
  while n and (n.id == KERN or n.id == LIGATURE) do
    n = n.next
  end
  if n and n.id == GLYPH then
    return n
  end
  return nil
end

```

Forbid the break at a discretionary whose two sides lie in one span, the way the
forbid_marked_disc break at a glue is forbidden. TEX gives a discretionary the attributes of the character in front of it, so the node's own attributes are the mark of that character and not a statement about the break, and they are not read: a discretionary at either edge of the argument leaves the marked text on one side only.

```
local function forbid_marked_disc (disc, left)
  if same_break_span(left, next_glyph(disc.next)) then
    disc.penalty = 10000
  end
end

```

Whether two marks are neighbours that stay together: a dash or an ellipsis is
adjacent_marks left out, because a run of them is one unit of its own, governed by the `nobreak` character set.

```
local function adjacent_marks (prev_mark, next_mark)
  return prev_mark and next_mark
     and prev_mark ~= "dash" and prev_mark ~= "ellipsis"
     and next_mark ~= "dash" and next_mark ~= "ellipsis"
end

```

Whether a punctuation mark is one of the two a run is built from.

run_mark
```
local function run_mark (mark)
  return mark == "dash" or mark == "ellipsis"
end

```

The rules: closing, middle, dash, ellipsis, and Latin closing punctuation must not
forbids_break start a line, opening punctuation must not end one, a half-width bracket follows

the rule of the side it closes, a run of dashes or ellipses stays together, and two adjacent marks are not separated. A dash or an ellipsis is read as the run it begins, so the rule applies to the first mark of a run alone; between the marks of a run the `NoBreakLongPunct` set decides. `AllowBreakBetweenPuncts` lifts the last rule alone: a mark that may not begin or end a line is a rule of the script, so the option cannot permit one. A word space the document wrote between two marks separates them, so the break there stays the document's: the two are no longer adjacent, and the rule above is not what governs the pair. The space the package retimes between two CJK characters is not a word space, and it is marked as inserted, so the boundary of a pair of CJK marks keeps its prohibition through the space.

```
local function forbids_break (prev_mark, prev_char, next_mark, next_char,
                              settings, spaced)
  if next_mark == "close" or next_mark == "middle"
     or next_mark == "half-close" or next_mark == "latin-close" then
    return true
  end
  if run_mark(next_mark) and not run_mark(prev_mark) then
    return true
  end
  if prev_mark == "open" or prev_mark == "half-open" then
    return true
  end
  if M.charsets.nobreak[prev_char] and M.charsets.nobreak[next_char] then
    return true
  end
  if adjacent_marks(prev_mark, next_mark) and not spaced
     and not (settings and settings.allow_break) then
    return true
  end
  return false
end

```

`edge_mark`

The punctuation group, the character, and the glyph at one edge of a visible node: a glyph is its own edge, and a box is read at the edge that faces the break, so a mark inside a box takes part in the rules. A formula carries no mark. The glyph is returned because `CJLineBreak` reads the settings snapshot of the character that would start the line, which a box does not carry. Whether ink stands between the mark and the edge of the box is returned as well, because a hang is the mark's own blank and a box that ends in a frame or in another character has its own ink at the edge the line ends at.

```
local function edge_mark (n, at_end)
```

```
  if n.id == GLYPH then
    return punct_class(n.char, settings_of(n), n.font), n.char, n, false
  elseif n.id == HLIST or n.id == VLIST then
    local mark, g, ink
    if at_end then
      mark, g, ink = last_punct(n.head)
    else
      mark, g, ink = first_punct(n.head)
    end
    if g then
      return mark, g.char, g, ink
    end
    return nil, nil, nil, false
  end
  return nil, nil, nil, false
end

```

next_visible

The next node that carries text, and the node in front of it. A node that carries text is a glyph, a formula, or a box; a box that occupies no advance is passed over, and an explicit break barrier ends the search. The node in front is returned because the rule that carries the hang of a mark stands where the line begins and not at the break itself: TEX drops the glue and the kerns of a line's head, so a rule written in front of them keeps the space the document put there, and the line starts the width of that space further in. A visible node that the search meets first has no node of its own in front, which is what tells the caller to use the break itself.

```
local function next_visible (n)
  local before = nil
  while n do
    if n.id == GLYPH or n.id == MATH then
      return n, before
    elseif n.id == HLIST or n.id == VLIST then
      if box_visible(n) then
        return n, before
      end
    elseif n.id == PENALTY or n.id == DISC then
      return nil, nil
    end
    before, n = n, n.next
  end
  return nil, nil
end

```

is_break_point

Glue and kerns are break opportunities; only the ones LuaCJK inserted and the

source spaces are considered.

```
local function is_break_point (n)
  if n.id == GLUE then
    return is_inserted(n) or n.subtype == SPACESKIP
  elseif n.id == KERN then
    return is_inserted(n)
  end
  return false
end

local function apply_kinsoku (head)
  local prev = nil
  local last = nil
  local cur  = head
  while cur do
    if cur.id == GLYPH
       or ((cur.id == HLIST or cur.id == VLIST) and box_visible(cur)) then
      last = cur
    elseif cur.id == DISC then
      forbid_marked_disc(cur, last)
    elseif last and (is_break_point(cur)
                     or (cur.id == GLUE and no_break_mark(cur))) then
      local nxt                              = next_visible(cur.next)
      local settings                         = settings_of(last)
      local prev_mark, prev_char             = edge_mark(last, true)
      local next_mark, next_char, next_node = nil, nil, nil
      if nxt then
        next_mark, next_char, next_node = edge_mark(nxt, false)
      end
      local next_settings                    = settings_of(next_node)
      local cj                               = next_settings.cj_strict
                                               and next_char and M.cj[next_char]
      local cur_marked                       = no_break_mark(cur)
                                               and not is_inserted(cur)
      local spaced                           = cur.id == GLUE
                                               and cur.subtype == SPACESKIP
                                               and not is_inserted(cur)
      if nxt and (cur_marked
                  or same_break_span(last, nxt) or cj
                  or (settings.kinsoku
                      and not disabled(last) and not disabled(nxt)
                      and forbids_break(prev_mark, prev_char,
                                        next_mark, next_char, settings,
                                        spaced)))
```

```
      then
        if not (prev and prev.id == PENALTY and prev.penalty >= 10000) then
          local p = new_node("penalty")
          inherit_attributes(p, last)
          p.penalty = 10000
          if M.attr_inserted then
            set_attribute(p, M.attr_inserted, 1)
          end
          p.next = cur
          if prev then
            prev.next = p
          else
            head = p
          end
          cur = p
        end
      end
    end
    prev = cur
    cur  = cur.next
  end
  return head
end

```

(*End of definition for* `apply_kinsoku` *and others.*)

`apply_fixed_widths` Styles such as `banjiao` give every punctuation mark a fixed advance. The glyph advance cannot be changed, so a compensating kern goes right after the glyph. A penalty in front of that kern keeps TEX from treating it as an extra break opportunity. The advance of a ratio style and the place of the ink in it come from the outline: the mark keeps the smaller of its two blanks and the style takes the rest from the blank on the outer side, which is the side that faces away from the text the mark encloses. A closing mark therefore keeps its ink where it is, an opening mark moves left by what it loses, and a middle mark is centred in the advance; a mark whose ink is not against the edge of its box (the fullwidth percent sign, whose ink stands in the middle) keeps a blank on both sides and grows back towards its natural advance, so the ink is never cut. A width the document sets with `\LuaCJKsetwidth` or `PunctWidth` is an advance in its own right, and the ink keeps the place its class gives it. A dash or an ellipsis is not shortened by the style: its advance is the length of the stroke, and halving it would let the two halves of a two-em dash overlap. A half-width bracket is already narrow, so the ratio of a style leaves it alone; a width the document sets applies to it like any other. A

Latin closing mark is left alone by the whole of this pass: its advance is the one the Latin font gave it, so neither a ratio nor `PunctWidth` rewrites it. A mark keeps the kern it was given. TEX runs the box-packing callback again for every box that holds the list, so a mark whose advance was corrected would be corrected a second time: the penalty and the kern the pass inserted are the record, and a mark that already carries them is passed over.

```
local function ink_shift (class, delta)
  if class == "open" or class == "half-open" then
    return delta
  elseif class == "middle" then
    return math.floor(delta / 2 + 0.5)
  end
  return 0
end

local function fixed_width (cur, settings, style)
  if inert_mark(punct_class(cur.char, settings, cur.font)) then
    return nil
  end
  local target = user_width(cur.char)
  if not target and settings.punct_width then
    target = settings.punct_width
  end
  if target then
    return target, false
  end
  local ratio = style.fixed_ratio
  if not ratio or M.charsets.long[cur.char] then
    return nil
  end
  local full = style.fixed_full and M.charsets.kaiming
  if full and full[cur.char] then
    return nil
  end
  local f    = font.getfont(cur.font)
  local size = f and f.size or 0
  if size > 0 then
    return math.floor(ratio * size + 0.5), true
  end
  return nil
end

% \begin{macro}[int]{style_metrics}
% \changes{v0.1.0}{2026/11/24}{The advance a ratio style gives a mark and the
```

```
%   place of its ink come from the outline, so a mark whose ink is not against
%   the edge of its box keeps a blank on both sides instead of losing part of
%   the ink.}
% The advance and the ink shift of a ratio style, from the outline of the
% mark: the smaller blank is kept on both sides, so the style takes the rest
% of the outer blank and the advance grows by whatever the ink does not allow
% it to lose.  A middle mark is centred in the target advance.  A file that
% cannot be read falls back to the advance the style asks for, with the ink
% moved by its class.
%    \begin{macrocode}
local function style_metrics (g, class, target)
  local lb, rb = M.mark_blanks(g)
  if not lb then
    return target, ink_shift(class, target - g.width)
  end
  local w = g.width
  if class == "middle" then
    return target, math.floor((target - (w - lb - rb)) / 2 + 0.5) - lb
  end
  local outer = (class == "open" or class == "half-open") and lb or rb
  local margin = math.max(math.min(lb, rb), outer - (w - target))
  return w - outer + margin, (outer == lb) and (margin - lb) or 0
end

% \begin{macro}[int]{fixed_advance}
% The advance a mark really takes, which is what the hang is measured inside:
% the width the document set, or the one a ratio style can give without
% cutting the ink.
%    \begin{macrocode}
local function fixed_advance (g, settings)
  local target, ratio = fixed_width(g, settings, style_of(settings))
  if not target then
    return g.width
  end
  if not ratio then
    return target
  end
  return style_metrics(g, punct_class(g.char, settings, g.font), target)
end

local function fixed_done (n)
  local p = n.next
  if p and p.id == PENALTY and p.penalty == 10000 and is_inserted(p) then
    local k = p.next
    return k ~= nil and k.id == KERN and is_inserted(k)
```

```
  end
  return false
end

local function apply_fixed_widths (head)
  local cur = head
  while cur do
    local settings = settings_of(cur)
    local style    = style_of(settings)
    local class    = cur.id == GLYPH
      and punct_class(cur.char, settings, cur.font)
    if class and not disabled(cur) and not fixed_done(cur) then
      local target, ratio = fixed_width(cur, settings, style)
      if target then
        local advance, shift = target, ink_shift(class, target - cur.width)
        if ratio then
          advance, shift = style_metrics(cur, class, target)
        end
        local delta = advance - cur.width
        if delta ~= 0 then
          local p = new_node("penalty")
          inherit_attributes(p, cur)
          p.penalty = 10000
          if M.attr_inserted then
            set_attribute(p, M.attr_inserted, 1)
          end
          local k   = make_kern(delta, cur)
          p.next    = k
          k.next    = cur.next
          cur.next  = p
          if shift ~= 0 then
            cur.xoffset = cur.xoffset + shift
          end
        end
      end
    end
    cur = cur.next
  end
  return head
end

```

(*End of definition for* `apply_fixed_widths`.)

A CJK character in math mode reaches the math list as a `math_char` noad.

Its family points at the math font, which usually has no CJK glyph, so LuaCJK replaces the nucleus by a `sub_box` holding the character in the CJK font.

MATH_CHAR
MATH_TEXT_CHAR

The node types of the two kinds of math character noad.

```
local MATH_CHAR      = node.id("math_char")
local MATH_TEXT_CHAR = node.id("math_text_char")

```

(*End of definition for* MATH_CHAR *and* MATH_TEXT_CHAR.)

any_cjk_math_enabled

Whether any noad of the list asks for CJK in math mode. The switch is part of the settings of the text, so the answer is per list.

```
local function any_cjk_math_enabled (head)
  local cur = head
  while cur do
    if settings_of(cur).cjk_math then
      return true
    end
    cur = cur.next
  end
  return false
end

```

(*End of definition for* any_cjk_math_enabled.)

SUB_MLIST

The node type of a nested math list.

```
local SUB_MLIST = node.id("sub_mlist")

```

(*End of definition for* SUB_MLIST.)

mark_math_brackets

Mark a formula that sets one opening or closing delimiter alone, so that the boundary rules can treat it as the bracket it is instead of as a Latin atom. The math list is still available here, before TEX converts it: such a formula is a single noad whose nucleus is one character and whose subtype is the math class `open` or `close`; the numbers of those classes are read from `\node.subtypes` rather than written here, and the mark is set on the noad and on its nucleus, whose attribute the conversion carries to the glyph it builds. A formula that also has a superscript, a subscript, a numerator, or a denominator is not a bracket, and neither is a formula of two atoms; both are left alone.

```
local function noad_class (name)
  local class = node.subtypes("noad")
  for sub = 0, 24 do
```

```
    if class[sub] == name then
      return sub
    end
  end
  return nil
end

local OPEN_NOAD  = noad_class("open")
local CLOSE_NOAD = noad_class("close")

local function mark_math_brackets (head)
  local n = head
  if not (M.attr_math_bracket and n and n.next == nil) then
    return head
  end
  if n.sup or n.sub or n.num or n.denom then
    return head
  end
  local nucleus = n.nucleus
  if not (nucleus and nucleus.id == MATH_CHAR) then
    return head
  end
  local value = nil
  if n.subtype == OPEN_NOAD then
    value = 1
  elseif n.subtype == CLOSE_NOAD then
    value = 2
  end
  if value then
    set_attribute(n, M.attr_math_bracket, value)
    set_attribute(nucleus, M.attr_math_bracket, value)
  end
  return head
end

```

(*End of definition for* `mark_math_brackets`.)

`rewrite_math` Rewrite the CJK noads of a math list, or warn once when the feature is off. Walk a math list, recursing into sub lists (fractions, braces, scripts) so that nested CJK characters are rewritten too. The declaration is split from the definition because of that recursion.

```
local rewrite_math_list

```

M.math_fallback

The fallback chain of each CJK math family, as a flat list that alternates the identifier of the font a character is tested in and the number of the math family that sets it. The text font of the family itself comes first, so a character the family has is set by the family.

```
M.math_fallback = {}

function M.set_math_fallback (main, list)
  M.math_fallback[main] = list
end

```

math_family_for

The first math family of the chain of ⟨*family*⟩ whose font has ⟨*char*⟩, or the family itself when none does. The fallback fonts are consulted only when AutoFallBack is on, which the settings of the noad carry, so the switch reaches math mode like it reaches the text.

```
local function math_family_for (char, family, settings)
  local list = M.math_fallback[family]
  if not list then
    return family
  end
  for i = 1, #list, 2 do
    if i > 1 and not (settings and settings.auto_fallback) then
      break
    end
    if has_char(list[i], char) then
      return list[i + 1]
    end
  end
  return family
end

```

rewrite_noad

Point the nucleus of one noad at the math family of its CJK family, or at the first fallback family that has the character, and recurse into the sub lists of its numerator, denominator, superscript, and subscript. The family number comes from the attribute TEX set for the current group, which is why `\setCJKmathfont` keeps its meaning inside a group: the family of the node changes, not a global font.

```
local function point_at_family (char, family)
  char.fam = family
  if M.attr_font then
    unset_attribute(char, M.attr_font)
  end
end
```

```
3692
3693 local function rewrite_field (s, noad, on, family)
3694   if not s then
3695     return
3696   end
3697   if s.id == MATH_CHAR or s.id == MATH_TEXT_CHAR then
3698     if is_cjk(s.char, settings_of(noad)) then
3699       point_at_family(s,
3700         math_family_for(s.char, family, settings_of(noad)))
3701     end
3702   elseif s.id == SUB_MLIST then
3703     rewrite_math_list(s.head, on, family)
3704   end
3705 end
3706
3707 local function rewrite_noad (noad, enabled, family)
3708   if disabled(noad) then
3709     return
3710   end
3711   local on = enabled or settings_of(noad).cjk_math
3712   if not on or not family then
3713     return
3714   end
3715   rewrite_field(noad.nucleus, noad, on, family)
3716   for _, field in ipairs({ "num", "denom", "sup", "sub" }) do
3717     rewrite_field(noad[field], noad, on, family)
3718   end
3719 end
3720
3721 rewrite_math_list = function (head, enabled, family)
3722   local cur = head
3723   while cur do
3724     rewrite_noad(cur, enabled, family)
3725     cur = cur.next
3726   end
3727   return head
3728 end
3729
```

Whether the one-time warning has been shown.

LuaCJK.warned_cjk_math

```
3730 M.warned_cjk_math = M.warned_cjk_math or false
3731
```

Warn once when a CJK character appears in math mode while the feature is off.

warn_disabled_cjk_math

Search a math list, including its sub lists, for a CJK character.

find_cjk_math

```
local function find_cjk_math (head)
  local cur = head
  while cur do
    local nuc = cur.nucleus
    if nuc then
      if (nuc.id == MATH_CHAR or nuc.id == MATH_TEXT_CHAR)
         and is_cjk(nuc.char, settings_of(cur)) then
        return true
      elseif nuc.id == SUB_MLIST and find_cjk_math(nuc.head) then
        return true
      end
    end
    for _, field in ipairs({ "num", "denom", "sup", "sub" }) do
      local sub = cur[field]
      if sub and sub.id == SUB_MLIST and find_cjk_math(sub.head) then
        return true
      end
    end
    cur = cur.next
  end
  return false
end

local function warn_disabled_cjk_math (head)
  if M.warned_cjk_math then
    return
  end
  if find_cjk_math(head) then
    M.warned_cjk_math = true
    texio.write_nl("term and log",
      "Package LuaCJK Warning: CJK character in math mode, "
      .. "but CJKmath is not enabled.")
    texio.write_nl("term and log",
      "(LuaCJK)                Use \\LuaCJKsetup{CJKmath = true} " ..
      "to typeset it with the CJK font.")
  end
end

local function math_family_of (head)
  local cur = head
  while cur do
    if M.attr_math_font then
      local family = has_attribute(cur, M.attr_math_font)
```

```
      if family and family > 0 then
        return family
      end
      local nuc = cur.nucleus
      if nuc then
        family = has_attribute(nuc, M.attr_math_font)
        if family and family > 0 then
          return family
        end
      end
    end
    cur = cur.next
  end
  return nil
end

local function rewrite_math (head)
  if not any_cjk_math_enabled(head) then
    warn_disabled_cjk_math(head)
    return head
  end
  return rewrite_math_list(head, true, math_family_of(head))
end

```

(*End of definition for* `rewrite_math` *and others.*)

A line that ends at a closing or middle mark, or that starts with an opening mark, measures the mark's advance short of its ink: the blank half stands in the margin. A line that never counted that blank refuses the mark it could have held, and is set loose instead. The hang therefore has to be known while the line is broken, and by then only the node list is left, so it is expressed as a zero-size rule of negative width beside the break: a closing mark that ends a line is followed by such a rule and the break glue carries the same width, so the two cancel while the line is set and the rule alone shortens the line when the break is taken; a glue and a rule in the same order let an opening mark begin a line. A kern would not do, because a kern that follows a character is a break opportunity of its own and would take a kinsoku penalty, while TEX never discards a rule at a line break.

Character protrusion is no help either, because it does not move the ink at the end of a line: `\protrudechars` together with `\rpcode` makes `\rightmarginkern` report the amount while the glyph keeps its position, measured at 10pt with a right code of 1000 in a short line, an overfull line, and a line whose width is the natural width. Only `\lpcode` moves the line, which is what the visible effect of microtype's protrusion on a Latin line consists of.

bound_width The advance that a mark keeps inside the line when it ends one, or nothing when the option is off. Without it the hang is the blank the font carries, or half of the advance when the file has no outline to read.

```
local function bound_width (settings)
  local value = settings and settings.bound_width
  if value then
    return value
  end
  return nil
end

```

(*End of definition for* `bound_width`.)

The hang of a mark is the blank its glyph carries, so the ink of a line-final or line-initial mark stands on the measure instead of a half em inside it. xeCJK reads that blank from the font with the XeTeX primitive `\XeTeXglyphbounds`; LuaTeX has no such query, so the outline is read from the font file, once per file and once per glyph.

Only the tables a measurement needs are parsed, and the bytes of a glyph are read when it is first asked for, so a sixteen-megabyte CJK face does not sit in memory a second time. The box is the one the control points span, which is what XeTeX reports as well.

font_files One entry per font file, holding its directory, its character map, and the boxes already measured.

```
local font_files = {}

```

(*End of definition for* `font_files`.)

u16 The big-endian integers of the file format.

i16

u32

```
local function u16 (s, o) return s:byte(o + 1) * 256 + s:byte(o + 2) end

local function i16 (s, o)
  local v = u16(s, o)
  if v >= 32768 then v = v - 65536 end
  return v
end

local function u32 (s, o) return u16(s, o) * 65536 + u16(s, o + 2) end
```

(*End of definition for* `u16`, `i16`, *and* `u32`.)

`font_file` The parsed directory of one font file, or nothing when it cannot be read. The handle stays open, because one file serves every mark of the document.

```
local function font_file (filename)
  local f = font_files[filename]
  if f ~= nil then
    return f or nil
  end
  local fh = io.open(filename, "rb")
  if not fh then
    font_files[filename] = false
    return nil
  end
  local function read (off, len)
    fh:seek("set", off)
    return fh:read(len) or ""
  end
  local head = read(0, 12)
  if #head < 12 then
    font_files[filename] = false
    return nil
  end
  local count = u16(head, 4)
  local dir = read(12, count * 16)
  local tables = {}
  for i = 0, count - 1 do
    local rec = i * 16
    local tag = dir:sub(rec + 1, rec + 4)
    tables[tag] = { offset = u32(dir, rec + 8), length = u32(dir, rec + 12) }
  end
  local t = { fh = fh, read = read, tables = tables, boxes = {} }
  if tables.head then
    t.upem = u16(read(tables.head.offset + 18, 2), 0)
  end
  if not (t.upem and t.upem > 0) then
    fh:close()
    font_files[filename] = false
    return nil
  end
  font_files[filename] = t
  return t
end
```

(*End of definition for* `font_file`.)

`font_cmap` The character map, preferring a Unicode subtable: a format 12 table for the third

platform, then a format 4 one, then whichever remains.

```
local function font_cmap (f)
  local off = f.tables.cmap and f.tables.cmap.offset
  if not off then return end
  local head = f.read(off, 4)
  local n = u16(head, 2)
  local recs = f.read(off + 4, n * 8)
  local choice
  for i = 0, n - 1 do
    local p = i * 8
    local pid, eid = u16(recs, p), u16(recs, p + 2)
    local sub = off + u32(recs, p + 4)
    local fmt = u16(f.read(sub, 2), 0)
    local rank
    if fmt == 12 and pid == 3 and eid == 10 then rank = 3
    elseif fmt == 12 then rank = 2
    elseif fmt == 4 and pid == 3 and eid == 1 then rank = 2
    elseif fmt == 4 then rank = 1 end
    if rank and (not choice or rank > choice.rank) then
      choice = { rank = rank, sub = sub, format = fmt }
    end
  end
  if not choice then return end
  if choice.format == 4 then
    local head4 = f.read(choice.sub, 14)
    local segX2 = u16(head4, 6)
    local seg = segX2 // 2
    f.cmap = {
      format = 4, data = f.read(choice.sub, u16(head4, 2)), segments = seg,
      end_of = 14, start_of = 14 + segX2 + 2,
      delta_of = 14 + segX2 + 2 + segX2,
      range_of = 14 + segX2 + 2 + segX2 + segX2,
    }
  else
    local head12 = f.read(choice.sub, 16)
    f.cmap = { format = 12, data = f.read(choice.sub, u32(head12, 4)) }
  end
end
```

(*End of definition for* `font_cmap`.)

`font_glyph` The glyph index of a character, or zero.

```
local function font_glyph (f, code)
  if f.cmap then
    local c = f.cmap
```

```
    if c.format == 4 then
      local end_of, start_of, delta_of, range_of = c.end_of, c.start_of, c.delta_of, c
      for i = 0, c.segments - 1 do
        if code <= u16(c.data, end_of + i * 2) then
          local start = u16(c.data, start_of + i * 2)
          if code < start then return 0 end
          local delta = i16(c.data, delta_of + i * 2)
          local ro = u16(c.data, range_of + i * 2)
          if ro == 0 then return (code + delta) % 65536 end
          local addr = range_of + i * 2 + ro + (code - start) * 2
          local g = u16(c.data, addr)
          if g == 0 then return 0 end
          return (g + delta) % 65536
        end
      end
      return 0
    elseif c.format == 12 then
      local p = 16
      while p + 12 <= #c.data do
        local s, e, g = u32(c.data, p), u32(c.data, p + 4), u32(c.data, p + 8)
        if code >= s and code <= e then return g + (code - s) end
        p = p + 12
      end
      return 0
    end
    return 0
  end
  return 0
end
```

(*End of definition for* `font_glyph`.)

`font_advance` The advance of a glyph from `hmtx`; a glyph past the last entry takes the last advance, as the format says.

```
local function font_advance (f, gid)
  if f.advances then
    return f.advances[gid] or f.last_advance or 0
  end
  local hhea = f.tables.hhea and f.tables.hhea.offset
  local hmtx = f.tables.hmtx and f.tables.hmtx.offset
  if not (hhea and hmtx) then return f.upem end
  local nhm = u16(f.read(hhea + 34, 2), 0)
  if nhm == 0 then return f.upem end
  local data = f.read(hmtx, nhm * 4)
  local advances, last = {}, 0
```

```
  for i = 0, nhm - 1 do
    last = u16(data, i * 4)
    advances[i] = last
  end
  f.advances, f.last_advance = advances, last
  return advances[math.min(gid, nhm - 1)] or last
end
```

(*End of definition for* `font_advance`.)

`cff_index` A CFF index: where its data starts and the offset of each item, which is one-based from that start.

```
local function cff_index (f, p)
  local head = f.read(p, 3)
  local count = u16(head, 0)
  if count == 0 then return { count = 0, next = p + 2, data = p + 2 } end
  local off_size = head:byte(3)
  local offs = f.read(p + 3, (count + 1) * off_size)
  local function off (i)
    local v = 0
    local base = (i - 1) * off_size
    for k = 1, off_size do v = v * 256 + offs:byte(base + k) end
    return v
  end
  local data = p + 3 + (count + 1) * off_size
  return { count = count, data = data, off = off, next = data + off(count + 1) - 1 }
end
```

(*End of definition for* `cff_index`.)

`cff_string` One item of a CFF index, which is a charstring or a dictionary.

```
local function cff_string (f, idx, i)
  local a, b = idx.off(i), idx.off(i + 1)
  return f.read(idx.data + a - 1, b - a)
end
```

(*End of definition for* `cff_string`.)

`cff_dict` A CFF dictionary as its operators and operands. Only the two operators that point at the charstrings and the private dictionary are read, so a real number is skipped rather than converted.

```
local function cff_dict (s)
  local out, stack = {}, {}
  local i, n = 1, #s
  while i <= n do
```

```
    local b = s:byte(i)
    if b <= 21 then
      i = i + 1
      local op = b
      if b == 12 then
        op = 1200 + s:byte(i)
        i = i + 1
      end
      out[op] = stack
      stack = {}
    elseif b == 28 then
      local v = s:byte(i + 1) * 256 + s:byte(i + 2)
      if v >= 32768 then v = v - 65536 end
      stack[#stack + 1] = v
      i = i + 3
    elseif b == 29 then
      local v = s:byte(i + 1) * 16777216 + s:byte(i + 2) * 65536
                + s:byte(i + 3) * 256 + s:byte(i + 4)
      if v >= 2147483648 then v = v - 4294967296 end
      stack[#stack + 1] = v
      i = i + 5
    elseif b == 30 then
      i = i + 1
      while i <= n do
        local c = s:byte(i)
        i = i + 1
        if c % 16 == 15 then break end
      end
    elseif b >= 32 and b <= 246 then
      stack[#stack + 1] = b - 139
      i = i + 1
    elseif b >= 247 and b <= 250 then
      stack[#stack + 1] = (b - 247) * 256 + s:byte(i + 1) + 108
      i = i + 2
    elseif b >= 251 and b <= 254 then
      stack[#stack + 1] = -(b - 251) * 256 - s:byte(i + 1) - 108
      i = i + 2
    else
      i = i + 1
    end
  end
  return out
end
```

(*End of definition for* `cff_dict`.)

cff_load The charstrings of a CFF font and the two subroutine indexes a charstring may call into.

```
local function cff_load (f)
  local base = f.tables["CFF "] and f.tables["CFF "].offset
  if not base then return end
  local hdr = f.read(base + 2, 1)
  local p = base + hdr:byte(1)
  local names = cff_index(f, p)
  local tops = cff_index(f, names.next)
  local strings = cff_index(f, tops.next)
  local gsubrs = cff_index(f, strings.next)
  local top = cff_dict(cff_string(f, tops, 1))
  local cs = top[17] and top[17][1]
  if not cs then return end
  f.charstrings = cff_index(f, base + cs)
  f.gsubrs = gsubrs
  local priv = top[18]
  if priv then
    local size, offset = priv[1], priv[2]
    local pdict = cff_dict(f.read(base + offset, size))
    if pdict[19] then
      f.subrs = cff_index(f, base + offset + pdict[19][1])
    end
  end
  f.cff = true
end
```

(*End of definition for* `cff_load`.)

cff_box The horizontal extent of a Type 2 charstring, read where the ink really stands: a handle of a curve lies outside the ink that curve draws, so the box is taken at the extremum of the curve, which is what the font paints. A straight segment contributes its end point, and the horizontal line operators alternate the axis from the one their name gives. A curve operator takes four operands, and the alternating ones start on the axis their name gives: the fourth operand of a vertical curve is the abscissa of its end point, and the one operand of a $4n + 1$ run closes the last curve on the other axis. The four operators of the flex family draw two curves of their own, so a glyph that joins two strokes without drawing them apart is measured as well. Hints are counted only so that their masks can be skipped. Each number is read from the charstring that holds it, since a subroutine carries its own operands, and the four bytes of the deprecated fixed form are read in full.

```
local function cff_box (f, cs)
  local x, xmin, xmax = 0, math.huge, -math.huge
```

```
  local stack, stems = {}, 0
  local function record (v)
    if v < xmin then xmin = v end
    if v > xmax then xmax = v end
  end
  local function move (dx)
    x = x + (dx or 0)
    record(x)
  end
  local function line (dx)
    x = x + (dx or 0)
    record(x)
  end
  local function curve (dx1, dx2, dx3)
    local x0 = x
    local x1 = x0 + (dx1 or 0)
    local x2 = x1 + (dx2 or 0)
    local x3 = x2 + (dx3 or 0)
    record(x0)
    record(x3)
    local a, b, c = x1 - x0, x2 - x1, x3 - x2
    local q2, q1 = a - 2 * b + c, 2 * (b - a)
    local roots = {}
    if math.abs(q2) < 1e-9 then
      if math.abs(q1) > 1e-9 then roots[1] = -a / q1 end
    else
      local disc = q1 * q1 - 4 * q2 * a
      if disc >= 0 then
        local root = math.sqrt(disc)
        roots[1] = (-q1 + root) / (2 * q2)
        roots[2] = (-q1 - root) / (2 * q2)
      end
    end
    for _, t in ipairs(roots) do
      if t > 0 and t < 1 then
        local u = 1 - t
        record(u * u * u * x0 + 3 * u * u * t * x1
               + 3 * u * t * t * x2 + t * t * t * x3)
      end
    end
    x = x3
  end
  local function number (code, b, i)
    if b >= 32 and b <= 246 then return b - 139, i + 1 end
    if b >= 247 and b <= 250 then
```

```
      return (b - 247) * 256 + code:byte(i + 1) + 108, i + 2
    end
    if b >= 251 and b <= 254 then
      return -(b - 251) * 256 - code:byte(i + 1) - 108, i + 2
    end
    if b == 28 then
      local v = code:byte(i + 1) * 256 + code:byte(i + 2)
      if v >= 32768 then v = v - 65536 end
      return v, i + 3
    end
    if b == 255 then
      local v = code:byte(i + 1) * 16777216 + code:byte(i + 2) * 65536
                + code:byte(i + 3) * 256 + code:byte(i + 4)
      if v >= 2147483648 then v = v - 4294967296 end
      return v / 65536, i + 5
    end
    return nil, i
  end
  local run
  run = function (code, depth)
    local i, n = 1, #code
    while i <= n do
      local b = code:byte(i)
      local v
      v, i = number(code, b, i)
      if v ~= nil then
        stack[#stack + 1] = v
      else
        i = i + 1
        local s = #stack
        if b == 21 or b == 22 then
          move(stack[1])
        elseif b == 4 then
          move(0)
        elseif b == 5 then
          for k = 1, s - 1, 2 do line(stack[k]) end
        elseif b == 6 or b == 7 then
          local horiz = (b == 6)
          for k = 1, s do
            line(horiz and stack[k] or 0)
            horiz = not horiz
          end
        elseif b == 8 then
          for k = 1, s - 5, 6 do curve(stack[k], stack[k + 2], stack[k + 4]) end
        elseif b == 24 then
```

```
4126            local k = 1
4127            while k <= s - 5 do
4128              curve(stack[k], stack[k + 2], stack[k + 4])
4129              k = k + 6
4130            end
4131            line(stack[k])
4132          elseif b == 25 then
4133            local k = 1
4134            while k <= s - 6 do
4135              line(stack[k])
4136              k = k + 2
4137            end
4138            curve(stack[k], stack[k + 2], stack[k + 4])
4139          elseif b == 26 or b == 27 then
4140            local k, first = 1, 0
4141            if s % 4 == 1 then first, k = stack[1], 2 end
4142            while k <= s - 3 do
4143              if b == 26 then
4144                curve(first, stack[k + 1], 0)
4145              else
4146                curve(stack[k], stack[k + 1], stack[k + 3])
4147              end
4148              first = 0
4149              k = k + 4
4150            end
4151          elseif b == 30 or b == 31 then
4152            local vertical = (b == 30)
4153            local k = 1
4154            while k <= s - 3 do
4155              if vertical then
4156                curve(0, stack[k + 1], stack[k + 3])
4157              else
4158                curve(stack[k], stack[k + 1], (k + 4 == s) and stack[s] or 0)
4159              end
4160              vertical = not vertical
4161              k = k + 4
4162            end
4163          elseif b == 12 then
4164            local op = code:byte(i)
4165            i = i + 1
4166            if op == 34 then
4167              curve(stack[1], stack[2], stack[4])
4168              curve(stack[5], stack[6], stack[7])
4169            elseif op == 35 then
4170              curve(stack[1], stack[3], stack[5])
```

```
4171              curve(stack[7], stack[9], stack[11])
4172            elseif op == 36 then
4173              curve(stack[1], stack[3], stack[5])
4174              curve(stack[6], stack[7], stack[9])
4175            elseif op == 37 then
4176              local dx, dy = 0, 0
4177              for k = 1, 10, 2 do
4178                dx, dy = dx + stack[k], dy + stack[k + 1]
4179              end
4180              curve(stack[1], stack[3], stack[5])
4181              curve(stack[7], stack[9],
4182                    (math.abs(dx) > math.abs(dy)) and stack[11] or -dx)
4183            end
4184          elseif b == 1 or b == 3 or b == 18 or b == 23 then
4185            stems = stems + s // 2
4186          elseif b == 19 or b == 20 then
4187            stems = stems + s // 2
4188            i = i + (stems + 7) // 8
4189          elseif b == 10 or b == 29 then
4190            local sub = table.remove(stack)
4191            local arr = (b == 10) and f.subrs or f.gsubrs
4192            if arr and arr.count > 0 and sub then
4193              local bias = (arr.count < 1240) and 107
4194                        or (arr.count < 33900 and 1131 or 32768)
4195              local idx = sub + bias + 1
4196              if idx >= 1 and idx <= arr.count and depth < 8 then
4197                run(cff_string(f, arr, idx), depth + 1)
4198              end
4199            end
4200          elseif b == 11 or b == 14 then
4201            return
4202          end
4203          stack = {}
4204        end
4205      end
4206    end
4207    run(cs, 0)
4208    if xmin == math.huge then return nil end
4209    return xmin, xmax
4210  end
```

(*End of definition for* `cff_box`.)

`font_box` The box of one character in font units: a `CFF` outline from its charstring, a `glyf` outline from the ten-byte header that starts it, and nothing for a character the font

does not have.

```
local function font_box (f, code)
  local known = f.boxes[code]
  if known ~= nil then
    return known or nil
  end
  if f.cff == nil then cff_load(f) end
  if f.cmap == nil then font_cmap(f) end
  local gid = font_glyph(f, code)
  if not gid or gid == 0 then
    f.boxes[code] = false
    return nil
  end
  local advance = font_advance(f, gid)
  local xmin, xmax
  if f.charstrings then
    if gid > f.charstrings.count then return nil end
    xmin, xmax = cff_box(f, cff_string(f, f.charstrings, gid + 1))
  else
    local glyf = f.tables.glyf and f.tables.glyf.offset
    local loca = f.tables.loca and f.tables.loca.offset
    local head = f.tables.head and f.tables.head.offset
    if not (glyf and loca and head) then return nil end
    local long = u16(f.read(head + 50, 2), 0) == 1
    local function loc (i)
      if long then return u32(f.read(loca + i * 4, 4), 0) end
      return u16(f.read(loca + i * 2, 2), 0) * 2
    end
    local a, b = loc(gid), loc(gid + 1)
    if b <= a then return nil end
    local hdr = f.read(glyf + a, 10)
    xmin, xmax = i16(hdr, 2), i16(hdr, 6)
  end
  if not xmin then return nil end
  local box = { xmin = xmin, xmax = xmax, advance = advance }
  f.boxes[code] = box
  return box
end
```

(*End of definition for* `font_box`.)

`mark_blank` The blank of a mark on the side that hangs, in scaled points, or nothing when the file cannot be read or is one member of a collection. The advance is the one the style gives the mark, so a compressed mark is measured inside its own advance,

where \apply_fixed_widths puts its ink.

```
local function mark_blank (g, mark, target)
  local font = font.getfont(g.font)
  if not (font and font.filename and font.size and (font.index or 0) == 0) then
    return nil
  end
  local f = font_file(font.filename)
  if not f then
    return nil
  end
  local box = font_box(f, g.char)
  if not box then
    return nil
  end
  local scale = font.size / f.upem
  local delta = target - box.advance * scale
  local blank
  if mark == "open" then
    blank = box.xmin * scale + delta
  else
    blank = target - box.xmax * scale
    if mark == "middle" then
      blank = blank - delta / 2
    end
  end
  if blank < 0 then
    blank = 0
  elseif blank > target then
    blank = target
  end
  return math.floor(blank + 0.5)
end
```

(*End of definition for* mark_blank.)

mark_blanks The blank of a glyph on each side, in scaled points at the size of the text, or nothing when the file cannot be read or is one member of a collection. The right blank is measured from the advance the node has, because that is where the character after it starts. The spacing pass reads this before the reader is documented, so it is published on the module table.

```
local function mark_blanks (g)
  local font = font.getfont(g.font)
  if not (font and font.filename and font.size and (font.index or 0) == 0) then
    return nil
```

```
  end
  local f = font_file(font.filename)
  if not f then
    return nil
  end
  local box = font_box(f, g.char)
  if not box then
    return nil
  end
  local scale = font.size / f.upem
  return math.floor(box.xmin * scale + 0.5),
         math.floor(g.width - box.xmax * scale + 0.5)
end
M.mark_blanks = mark_blanks
```

(*End of definition for* `mark_blanks`.)

`hang_amount` How much of a mark's advance stands outside the line: the part that `PunctBoundWidth` leaves out, the blank the font carries for the mark, or half of the advance the style gives it when the file has no outline to read. The advance is the one `\apply_fixed_widths` gives it, so a compressed style hangs less.

```
local function hang_amount (g, mark, settings)
  local target = fixed_advance(g, settings)
  local bound  = bound_width(settings)
  if bound then
    return math.max(0, target - bound)
  end
  local blank = mark_blank(g, mark, target)
  if blank then
    return blank
  end
  return math.floor(0.5 * target + 0.5)
end

```

(*End of definition for* `hang_amount`.)

`open_hang` `item_label` `head_lead` `hang_rule` The hang of an opening mark that a line begins with, as a length and the glyph it belongs to, or zero when the mark does not open or its style does not hang; the rule that carries that length into the line, of negative width and with the attributes of the mark; and the head of a paragraph, found by walking over what carries no text of the paragraph: the local paragraph node, the whatsits, the indentation, which LuaTEX puts in front of the text as an empty box of the width

of \parindent, and the label box of a list item, which the list writes in front of the item's text. A box is passed over when it carries no content, when it occupies no advance, or when a penalty of zero follows it, and a penalty is passed over unless it forces a break. The penalty of zero is the one the list writes after the label box of an item, so the label takes an advance in the line without being part of the paragraph, and a mark that begins the item hangs into it the way it hangs into the indentation of a paragraph. A box that carries text of the paragraph is the head itself: the mark it holds at the edge that faces the line is the mark the line begins with, read the way a break reads it, so a paragraph whose first item is a box hangs that mark as well, and \mbox around the opening mark of a paragraph leaves its blank in the indentation exactly as plain text does. A box that carries ink in front of that mark leaves nothing to hang: the ink of a frame is the edge the line starts at, so \fbox around the mark stops at the indentation instead of moving its frame into it.

```
local function open_hang (n)
  local lead, _, glyph, ink = edge_mark(n, false)
  if not (glyph and not ink and not disabled(glyph) and lead == "open"
          and style_of(settings_of(glyph)).hang) then
    return 0, nil
  end
  return hang_amount(glyph, "open", settings_of(glyph)) or 0, glyph
end

local function item_label (b)
  local n = b.next
  return n ~= nil and n.id == PENALTY and n.penalty == 0
end

local function head_lead (head)
  local indent = tex.parindent or 0
  local prev   = nil
  local cur    = head
  while cur do
    if cur.id == GLYPH then
      return cur, prev
    elseif cur.id == HLIST or cur.id == VLIST then
      if cur.head and cur.width ~= 0 and not item_label(cur) then
        return cur, prev
      end
    elseif cur.id == GLUE then
      if cur.width ~= indent or cur.stretch ~= 0 or cur.shrink ~= 0 then
        return nil, nil
      end
```

```
    elseif cur.id == KERN then
      if cur.kern ~= 0 then
        return nil, nil
      end
    elseif cur.id == PENALTY then
      if cur.penalty < 0 then
        return nil, nil
      end
    elseif cur.id == MATH or cur.id == DISC or cur.id == RULE then
      return nil, nil
    end
    prev = cur
    cur  = cur.next
  end
  return nil, nil
end

local function hang_rule (width, glyph, prev, next_)
  local r  = new_node("rule")
  r.width  = -width
  r.height = 0
  r.depth  = 0
  inherit_attributes(r, glyph)
  r.next = next_
  if prev then
    prev.next = r
  end
  return r
end

```

(*End of definition for* `open_hang` *and others.*)

`hang_bounds` Bracket every break point that a mark ends or starts, so the line measures the hang and the text inside a line does not move. A break that a preceding penalty already forbids is left alone, so a fixed width does not carry a rule that can never be used. The two boundaries that carry no glue of their own are bracketed as well: the start of the paragraph, and the penalty that holds a break the document forced, where the rule that shifts the line is all there is to insert. The rule stands where the line begins and not at the break itself: TEX drops the glue and the kerns of a line's head, so a rule written in front of them keeps the space the document put there, and the line starts the width of that space further in. Those two are left out when `\leftskip` reaches the line with a fill, which is how a centered or a flush-right paragraph is set: the line then begins at the centering position rather

than at the edge of the measure, and the rule would move it instead of leaving the mark's blank in the margin. The hang at a break stays in every case, because the line breaker has already counted it when it decided that the mark fits; where the lines are centered or flushed right it is also what keeps the ink of a line end on the alignment edge, since the widened glue moves the whole line by the blank it takes out. The last line of such a paragraph is bracketed by hand, because no break glue ends it: the rule alone goes behind the last mark, with no glue to widen, since no break decision depends on it. It is put there only when `\leftskip` carries a fill, because otherwise the material starts at the margin and the rule leaves the position of the ink alone – in a justified or a ragged-right paragraph the trailing fill absorbs it, so the line would only carry a node that does nothing. A mark inside a box hangs only when nothing inked stands between it and the edge of that box. The blank of a mark is the space its font leaves in the advance, and that space ends the line only where the mark's advance does: a frame rule, or another character, is the ink the edge of the line meets, so `\fbox` and `\framebox` keep their own edge on the measure instead of carrying the whole box past it, while the `\mbox` around a mark still leaves its blank in the margin the way plain text does.

```
local function finite_skip (s)
  return not s or (s.stretch_order == 0 and s.shrink_order == 0)
end

local function hang_bounds (head)
  local left  = finite_skip(tex.leftskip)
  local first, before = head_lead(head)
  if first and left then
    local width, glyph = open_hang(first)
    if width > 0 then
      local r = hang_rule(width, glyph, before, first)
      if not before then
        head = r
      end
    end
  end
  local prev = nil
  local last = nil
  local cur  = head
  while cur do
    if cur.id == GLYPH
       or ((cur.id == HLIST or cur.id == VLIST) and box_visible(cur)) then
      last = cur
    elseif cur.id == PENALTY and cur.penalty <= -10000 then
```

```
      local nxt, before = next_visible(cur.next)
      local width, glyph = 0, nil
      if nxt and left then
        width, glyph = open_hang(nxt)
      end
      if width > 0 then
        hang_rule(width, glyph, before or cur, nxt)
      end
    elseif is_break_point(cur) and last and cur.id == GLUE
           and not (prev and prev.id == PENALTY and prev.penalty >= 10000) then
      local nxt, before                    = next_visible(cur.next)
      local tail, _, tail_glyph, tail_ink = edge_mark(last, true)
      local w_end                          = 0
      local w_start, lead_glyph            = 0, nil
      if nxt then
        w_start, lead_glyph = open_hang(nxt)
      end
      if tail_glyph and not tail_ink and not disabled(tail_glyph)
         and (tail == "close" or tail == "middle")
         and style_of(settings_of(tail_glyph)).hang then
        w_end = hang_amount(tail_glyph, tail, settings_of(tail_glyph))
      end
      if w_end > 0 or w_start > 0 then
        local delta = w_end + w_start
        cur.width = cur.width + delta
        if w_end > 0 then
          local r  = new_node("rule")
          r.width  = -w_end
          r.height = 0
          r.depth  = 0
          inherit_attributes(r, tail_glyph)
          r.next = cur
          if prev then
            prev.next = r
          else
            head = r
          end
          prev = r
        end
        if w_start > 0 then
          local r  = new_node("rule")
          r.width  = -w_start
          r.height = 0
          r.depth  = 0
          inherit_attributes(r, lead_glyph)
```

```
          local anchor = before or cur
          r.next      = nxt
          anchor.next = r
        end
      end
    end
    prev = cur
    cur  = cur.next
  end
  if last and not left then
    local tail, _, glyph, ink = edge_mark(last, true)
    if glyph and not ink and not disabled(glyph)
       and (tail == "close" or tail == "middle")
       and style_of(settings_of(glyph)).hang then
      local w_end = hang_amount(glyph, tail, settings_of(glyph))
      if w_end > 0 then
        hang_rule(w_end, glyph, last, last.next)
      end
    end
  end
  return head
end

```

(*End of definition for* `hang_bounds`.)

TEX can break at the glue that LuaCJK inserts between CJK characters. At the end of a paragraph the last two such break points get a penalty, so that at least three CJK characters travel to the last line together instead of a single one.

`apply_check_single` Insert the widow penalties. Only a paragraph whose last visible node is a CJK glyph, or a box that ends with one, is examined, and only when its settings ask for it. A box is read from its own list, because the text of `\underline` and of an `\mbox` sits in one and the character it holds is the character that ends the paragraph; a formula inside the box ends the text, and everything else a box holds — a rule, a kern, a glue, a whatsit — carries no character. What a box ends with is also counted as the characters it holds, not as one character: its list is already packed, so no break can be taken inside it, and three characters in a box travel together without a penalty. The scan back to the previous character passes over what carries no text — a whatsit such as a color change, a mark, an insertion, the adjustment of a box, a penalty, and a kern this package inserted — so such a node does not end the run. The penalty goes in front of the glue that the break

is taken at, and it is left out when a penalty already stands in front of that glue, so a second run over the same list adds nothing.

```
local function trailing_cjk (list)
  local items = {}
  local node  = list
  while node do
    items[#items + 1] = node
    node = node.next
  end
  local count = 0
  local glyph
  for i = #items, 1, -1 do
    local item = items[i]
    if item.id == GLYPH then
      if not is_cjk(item.char, settings_of(item), item.font) then
        return count, glyph
      end
      count = count + 1
      glyph = glyph or item
    elseif item.id == HLIST or item.id == VLIST then
      local inner, deep = trailing_cjk(item.head)
      if inner == 0 then
        return count, glyph
      end
      count = count + inner
      glyph = glyph or deep
    elseif item.id == MATH then
      return count, glyph
    end
  end
  return count, glyph
end

local function apply_check_single (head)
  local trail = {}
  local node  = head
  while node do
    trail[#trail + 1] = node
    node = node.next
  end
  local last, chars, last_glyph
  for i = #trail, 1, -1 do
    local item = trail[i]
    if item.id == GLYPH or item.id == HLIST or item.id == VLIST then
```

```
      local n = 0
      local glyph
      if item.id == GLYPH then
        glyph = item
        if is_cjk(item.char, settings_of(item), item.font) then
          n = 1
        end
      else
        n, glyph = trailing_cjk(item.head)
      end
      if n > 0 then
        last, chars, last_glyph = i, n, glyph
      end
      break
    end
  end
  if not last then
    return head
  end
  local settings = settings_of(last_glyph)
  if not settings.check_single then
    return head
  end
  local penalty  = settings.widow_penalty
  local inserted = 0
  local i        = last
  while i > 1 and inserted < 2 and chars < 3 do
    local j = i - 1
    while j >= 1 and (trail[j].id == GLUE or transparent_node(trail[j])
                      or (trail[j].id == KERN and is_inserted(trail[j]))) do
      j = j - 1
    end
    if j < 1 then
      break
    end
    local item  = trail[j]
    local n     = 0
    local glyph
    if item.id == GLYPH then
      if is_cjk(item.char, settings_of(item), item.font) then
        n, glyph = 1, item
      end
    elseif item.id == HLIST or item.id == VLIST then
      n, glyph = trailing_cjk(item.head)
    end
```

```
    if n == 0 then
      break
    end
    local at    = nil
    local taken = false
    for k = j + 1, i - 1 do
      local between = trail[k]
      if between.id == GLUE then
        at = k
        break
      elseif between.id == PENALTY then
        taken = true
      end
    end
    if at and not taken then
      local p = new_node("penalty")
      inherit_attributes(p, glyph)
      p.penalty          = penalty
      p.next             = trail[at]
      trail[at - 1].next = p
    end
    inserted = inserted + 1
    chars    = chars + n
    i        = j
  end
  return head
end

```

(*End of definition for* `apply_check_single`.)

`LuaCJK.process` Run the passes over a list, in the order in which they depend on each other: what shaping left of the zero-width characters first, so that the spacing pass sees the text as it is set; then the spacing, the fixed widths, the prohibitions, and the widow penalties. Without `pre_shaping_filter` the font is moved here instead, after shaping, because there was no pass in front of it. The hang of a mark is inserted only when the list is a paragraph about to be broken, because a box packed by `\hpack` has no break to hang at.

```
local function process (head, linebreak)
  if M.shaping then
    head = drop_orphans(head)
  else
    head = rewrite_fonts(head)
  end
```

```
  head = spacing(head)
  head = apply_fixed_widths(head)
  if linebreak then
    head = hang_bounds(head)
  end
  head = apply_kinsoku(head)
  head = apply_check_single(head)
  return head
end

M.process = process

```

(*End of definition for* `LuaCJK.process`.)

`LuaCJK.pre_linebreak` The line-breaking callback.

```
function M.pre_linebreak (head, groupcode)
  return process(head, true)
end

```

(*End of definition for* `LuaCJK.pre_linebreak`.)

`LuaCJK.hpack` The box-packing callback.

```
function M.hpack (head, groupcode, size, packtype, direction, attributelist)
  return process(head)
end

```

(*End of definition for* `LuaCJK.hpack`.)

`LuaCJK.pre_mlist_to_hlist` The math callback. The bracket mark is set first, so that a formula which is one delimiter is read the same way whether or not `CJKmath` is on.

```
function M.pre_mlist_to_hlist (head, display, penalties)
  return rewrite_math(mark_math_brackets(head))
end

```

(*End of definition for* `LuaCJK.pre_mlist_to_hlist`.)

`hyphenate_runs` Hyphenate the runs of a word that lie between its CJK characters. TEX counts a CJK character as a letter, so the patterns see a CJK character and the Latin word beside it as one word, and a point they find there puts a hyphen right next to the character. A CJK character ends a word here instead: the run of text on each side of it is hyphenated on its own, so no point the patterns find can stand between a

CJK character and the letters that follow, and the boundary keeps the glue that CJKecglue gives it.

```
local function hyphenate_runs (head, tail)
  local run  = nil
  local prev = nil
  local cur  = head
  while cur do
    if cur.id == GLYPH and is_cjk(cur.char, settings_of(cur), cur.font) then
      if run then
        lang.hyphenate(run, prev)
      end
      run = nil
    elseif not run then
      run = cur
    end
    prev = cur
    if cur == tail then
      break
    end
    cur = cur.next
  end
  if run then
    lang.hyphenate(run, prev)
  end
end

```

(*End of definition for* `hyphenate_runs`.)

`LuaCJK.hyphenate` The hyphenation callback. TEX hyphenates a word while it breaks the lines, after the pre-linebreaking pass has already built the list, so the discretionaries of a word appear after the prohibitions have been placed and nothing forbids them yet. The callback hyphenates exactly as TEX does, a run of non-CJK text at a time, and then reads the discretionaries it has just created, one word at a time.

```
function M.hyphenate (head, tail)
  hyphenate_runs(head, tail)
  local left = nil
  local cur  = head
  while cur do
    if cur.id == GLYPH then
      left = cur
    elseif cur.id == DISC then
      forbid_marked_disc(cur, left)
    end
```

```
    if cur == tail then
      break
    end
    cur = cur.next
  end
end

```

(*End of definition for* `LuaCJK.hyphenate`.)

The callbacks are registered in this order: the font substitution, which luaotfload calls from its own handler before it shapes a list; the spacing, which runs on both lists; the math; and the hyphenation, which is the only one that runs inside the line breaker rather than around it. The substitution is registered only when luaotfload has the callback; without it, `process` applies the font after shaping instead.

```
if M.shaping then
  luatexbase.add_to_callback("pre_shaping_filter",
    M.pre_shaping, "LuaCJK.font")
end
luatexbase.add_to_callback("pre_linebreak_filter",
  M.pre_linebreak, "LuaCJK.spacing")
luatexbase.add_to_callback("hpack_filter", M.hpack, "LuaCJK.spacing")
luatexbase.add_to_callback("pre_mlist_to_hlist_filter",
  M.pre_mlist_to_hlist, "LuaCJK.math")
luatexbase.add_to_callback("hyphenate", M.hyphenate, "LuaCJK.hyphenate")
⟨/lua-core⟩
```

A.9 Decorations

lua-ul draws each decoration as a pair of nodes: a leaders glue of width w and a glue of width $-w$ that gives the space back, so the text does not move. Changing those numbers is enough to inset the decoration and to leave out the punctuation, which is why LuaCJK adjusts the box instead of drawing the decoration itself. TEX numbers the glue subtypes, and 100 is `\leaders`, 101 `\cleaders`, and 102 `\xleaders`; lua-ul uses the first and this package adds the third where a grid cell may be spread, so the pass takes all three.

The emphasis dots are the exception: a pattern repeated along the span cannot follow the characters when the inter-character glue stretches, and it would dot Latin text as well, so the ink of `\CJKunderdot` goes into the text itself. Its leaders pair carries an empty box and keeps the width, the stretch, and the shrink of the span, while the pass writes one dot behind each CJK character, centred on the advance of that character.

LuaCJKfntef.attr
LuaCJKfntef.settings
LuaCJKfntef.settings_by_key
LuaCJKfntef.dot_attr
LuaCJKfntef.color_attr

The attribute that carries the number of a settings entry, and the entry itself: the inset at both ends, whether punctuation is skipped, and the geometry of the emphasis dots. An entry is keyed by the string the TEX side builds from the same values, so two calls that set a decoration the same way share one entry while two that set it differently keep their own; the key is also part of the cache key of a decoration type, which is what lets a leaders pair name the entry of the call that built it. The dots carry an attribute of their own, so that a box the pass sees twice is not dotted twice, and the number of luacolor's attribute, which is what colors a rule.

```
⟨*lua-fntef⟩
LuaCJKfntef = LuaCJKfntef or {}

local M        = LuaCJKfntef
local GLYPH    = node.id("glyph")
local GLUE     = node.id("glue")
local LEADERS  = 100
local CLEADERS = 101
local XLEADERS = 102
local KERN     = node.id("kern")
local HLIST    = node.id("hlist")
local VLIST    = node.id("vlist")
local RULE     = node.id("rule")
local WHATSIT  = node.id("whatsit")

M.attr            = M.attr     or nil
M.settings        = M.settings or {}
M.settings_by_key = M.settings_by_key or {}
M.next_id         = M.next_id  or 1
M.dot_attr        = M.dot_attr or luatexbase.new_attribute("LuaCJKfntef.dot")
M.color_attr      = luatexbase.attributes
  and luatexbase.attributes["LuaCol@Attribute"]

```

(*End of definition for* `LuaCJKfntef.attr` *and others.*)

LuaCJKfntef.em_descent

The descent of the em box of the current font, in scaled points. A CJK em box is one em square, so its descent is both what a highlight has to add below the baseline to cover a character, and what separates the ink from the bottom of that square.

The value comes from `ascent`, `descent` and `units_per_em` of the font file, cached per file name: opening a CJK font costs about half a second, and one file serves every size.

```
local em_metrics = {}
```

```
4682
4683 local function font_em (filename)
4684   local m = em_metrics[filename]
4685   if m ~= nil then
4686     return m or nil
4687   end
4688   local ok, t = pcall(fontloader.open, filename)
4689   if ok and t and t.units_per_em and t.units_per_em > 0 and t.descent then
4690     m = { descent = t.descent, upem = t.units_per_em }
4691     fontloader.close(t)
4692   else
4693     m = false
4694   end
4695   em_metrics[filename] = m
4696   return m or nil
4697 end
4698
4699 function M.em_descent (id)
4700   local f = tonumber(id)
4701   f = f and f > 0 and font.getfont(f) or nil
4702   local size = f and f.size or tex.sp("1em")
4703   local m = f and f.filename and font_em(f.filename)
4704   local sp
4705   if m then
4706     sp = math.floor(m.descent / m.upem * size + 0.5)
4707   else
4708     sp = math.floor(0.12 * size + 0.5)
4709   end
4710   return string.format("%d", sp)
4711 end
4712
```

(*End of definition for* `LuaCJKfntef.em_descent`.)

A decoration records its settings by selecting an entry: the number of the entry goes into the attribute on the decorated text, and attribute zero means that the text carries no decoration of this package.

`LuaCJKfntef.new_settings` Register one entry and select it. The entry carries the dots of `\CJKunderdot` when the command gave no symbol: the side of one dot, the distance from the baseline to its top, and the color attribute of `format`, zero when the dots follow the color of their text. ⟨*key*⟩ identifies those settings: a call whose key is already known selects the entry it registered, and only a new key registers one. The attribute is written either way, because the text carries the entry of the innermost decoration

and the pairs read the attribute of their own type. ⟨*commute*⟩ is the `commute` mode of the call: with `false` the pass places the ink this decoration covers below its own.

```
function M.new_settings (key, subtract, skip, hidden, dots, size, top, color,
                         commute)
  local id = M.settings_by_key[key]
  if not id then
    id        = M.next_id
    M.next_id = id + 1
    if dots then
      dots = { size = size or 0, top = top or 0, color = color or 0 }
    else
      dots = nil
    end
    M.settings[id]         = { subtract = subtract, skip = skip,
                               hidden = hidden, dots = dots,
                               commute = commute }
    M.settings_by_key[key] = id
  end
  if M.attr then
    tex.setattribute(M.attr, id)
  end
end
```

(*End of definition for* `LuaCJKfntef.new_settings`.)

`kind` The punctuation group of a character, as the core module classifies it.

```
local function kind (c)
  local class = LuaCJK.punct_class
  if class then
    return class(c)
  end
  return nil
end

```

(*End of definition for* `kind`.)

`node_width` The horizontal extent of a node, whether it stores it as a width or as a kern: a `kern` node has no `width` field in LuaTEX, and reading a nil where a dimension belongs put every later interval in the wrong place.

```
local function node_width (n)
  if n.id == KERN then
    return n.kern or 0
  end
```

```
  return n.width or 0
end
```

(*End of definition for* `node_width`.)

`ink_span` `ink_bounds` `ink_of`

How far below the baseline the ink of a decoration reaches: the top and the bottom of everything its leaders paint, walked through the boxes it sits in and through their shifts, which LuaTEX stores positive downwards. A box that paints nothing itself is taken at its own dimensions. The entry of the emphasis dots carries their geometry instead, because their ink is written into the text and not into the leader box. A drawing is the exception: its ink reaches the page as PDF literals, so the nodes say nothing about where it is. `drawn` reports such a box with no ink of its own, and the pair is then reported as having no measurable extent.

```
local drawn

local function ink_span (n, shift, acc)
  if n.id == GLYPH or n.id == RULE then
    acc.top    = math.min(acc.top,    shift - (n.height or 0))
    acc.bottom = math.max(acc.bottom, shift + (n.depth  or 0))
    acc.found  = true
  elseif n.id == HLIST or n.id == VLIST then
    local inner = { top = math.huge, bottom = -math.huge, found = false }
    local m = n.head
    while m do
      ink_span(m, shift + (n.shift or 0), inner)
      m = m.next
    end
    if not inner.found then
      inner.top    = shift - (n.height or 0)
      inner.bottom = shift + (n.depth  or 0)
    end
    acc.top    = math.min(acc.top,    inner.top)
    acc.bottom = math.max(acc.bottom, inner.bottom)
    acc.found  = true
  end
end

local function ink_bounds (n)
  if not n then
    return nil
  end
  local acc = { top = math.huge, bottom = -math.huge, found = false }
  ink_span(n, 0, acc)
```

```
  if not acc.found then
    return nil
  end
  return acc.top, acc.bottom
end

local function ink_of (settings, leaders)
  if settings and settings.dots then
    local top = settings.dots.top
    return top - settings.dots.size, top
  end
  if drawn and leaders and leaders.id == HLIST then
    local literal, ink = drawn(leaders)
    if literal and not ink then
      return nil
    end
  end
  return ink_bounds(leaders)
end

```

(*End of definition for* `ink_span`, `ink_bounds`, *and* `ink_of`.)

`nested_pair` Whether ⟨*n*⟩ opens the leaders pair of a decoration that is nested inside the span a walk is measuring: a leaders glue followed by the glue that gives its width back. lua-ul inserts the pair of an inner decoration inside the span of the outer one, and the two nodes cancel in width, stretch, and shrink, so a walk steps over both at once. Counting the leaders glue alone ended the walk at the first nested decoration, which is why an outer `\CJKunderdot` found no character to dot.

```
local function nested_pair (n)
  return n.id == GLUE
     and (n.subtype == LEADERS or n.subtype == CLEADERS
          or n.subtype == XLEADERS)
     and n.next and n.next.id == GLUE and n.next.width == -n.width
end

```

(*End of definition for* `nested_pair`.)

`segments` The intervals of a span of width ⟨*w*⟩ that the decoration may cover: the first and the last ⟨*inset*⟩ are taken off, and with ⟨*skip-punct*⟩ every punctuation mark inside is cut out as well. A cut is clamped to the span, so a mark that straddles the end keeps the part that fits.

```
local function segments (w, start, inset, skip_punct)
```

```
  local cuts = {}
  if skip_punct then
    local x = 0
    local g = start
    while g and x < w do
      local nxt = g.next
      if nested_pair(g) then
        nxt = nxt.next
      else
        local width = node_width(g)
        if g.id == GLYPH and kind(g.char) then
          cuts[#cuts + 1] = { x, math.min(x + width, w) }
        end
        x = x + width
      end
      g = nxt
    end
  end
  local result = {}
  local low    = math.min(inset, w)
  local high   = math.max(w - inset, low)
  for _, cut in ipairs(cuts) do
    if cut[1] > low then
      result[#result + 1] = { low, math.min(cut[1], high) }
    end
    low = math.max(low, cut[2])
  end
  if low < high then
    result[#result + 1] = { low, high }
  end
  return result
end

```

(*End of definition for* `segments`.)

`settings_at` The settings of the decoration that ⟨*node*⟩ belongs to, read from the attribute the node carries, or nil when it carries none.
The pass asks this of the leaders glue of a pair and never of the text: lua-ul inserts the pair before the span, so the first node of the span is the one the innermost decoration marked, which is the outer call only when nothing is nested inside it. The leaders glue carries the attribute list of the type the call built, and a type is cached per settings entry, so it names the entry of its own call.

```
local function settings_at (node_)
```

```
  if not node_ or not M.attr then
    return nil
  end
  local id = node.has_attribute(node_, M.attr)
  if not id or id == 0 then
    return nil
  end
  return M.settings[id]
end
```

(*End of definition for* `settings_at`.)

`single_rule` The rule that a decoration box reduces to, and how far the box moves it down. A box holding nothing but kerns, glues and one rule qualifies. `\leaders` repeats the box, and only whole copies fit, so the last fraction of the span stays empty: with a one-point object that is a visible gap at the right end of a highlight. A box that reduces to one rule is therefore drawn as a single rule of the full width instead.

```
local function single_rule (list, down)
  local found
  local n = list
  while n do
    if n.id == RULE then
      if found then
        return nil
      end
      found = n
    elseif n.id == HLIST then
      if found then
        return nil
      end
      local r, d = single_rule(n.head, down + (n.shift or 0))
      if not r then
        return nil
      end
      found, down = r, d
    elseif n.id == KERN then
      -- horizontal material, no vertical effect
    elseif n.id == GLUE and not n.leader then
      -- a plain glue between the pieces
    else
      return nil
    end
    n = n.next
```

```
  end
  if found then
    return found, down
  end
  return nil
end

local function box_rule (box)
  if not box or box.id ~= HLIST then
    return nil
  end
  local rule, down = single_rule(box.head, box.shift or 0)
  -- Only a solid block qualifies: the rule must fill the box, or the
  -- pattern would lose whatever sits beside it.
  if rule and rule.width == box.width then
    return rule, down
  end
  return nil
end

```

(*End of definition for* `single_rule`.)

`copy_attributes` Copy every attribute of one node onto another. The ink of a leaders box takes the attributes of the glue, not of the box it repeats, so a piece whose rule carries a color of its own would lose it.

```
local function copy_attributes (target, source)
  local a = source.attr
  while a do
    node.set_attribute(target, a.number, a.value)
    a = a.next
  end
end

```

(*End of definition for* `copy_attributes`.)

`hide_list` Take the ink out of the list a packed box holds. A glyph becomes a kern of the same width and a rule is dropped, because the box recorded its dimensions when it was packed and only the ink has to go; a box inside it is walked in turn, so the numerator of a fraction and the rule of its bar are hidden like a plain number. A box of the emphasis dots is left alone: its rule is the ink of the decoration, which `hidden` keeps, not of the text it covers.

```
local function hide_list (head)
```

```
  local prev, n = nil, head
  while n do
    local nxt = n.next
    if n.id == GLYPH then
      local k = node.new("kern")
      k.kern  = n.width or 0
      k.next  = nxt
      copy_attributes(k, n)
      if prev then
        prev.next = k
      else
        head = k
      end
      node.free(n)
      prev = k
    elseif n.id == RULE then
      if prev then
        prev.next = nxt
      else
        head = nxt
      end
      node.free(n)
    elseif n.id == HLIST
           and node.has_attribute(n, M.dot_attr) ~= nil then
      prev = n
    elseif n.id == HLIST or n.id == VLIST then
      n.head = hide_list(n.head)
      prev   = n
    else
      prev = n
    end
    n = nxt
  end
  return head
end

```

(*End of definition for* `hide_list`.)

`hide_text` Take the ink out of a decorated span and keep its dimensions. Every glyph becomes a kern of the same width, so the text still occupies the same space and the line breaks of the paragraph are the ones TeX found before the pass ran; one rule of no width carries the height and the depth, because a kern has neither. The span is the text the leaders pair covers, so the walk stops at the width of that pair. A box of the span has its own dimensions, so `hide_list` takes its ink out instead,

and the box is left in the list where it stands. The pair of an inner decoration is dropped whole: it cancels in width, and the ink of that decoration goes with the text the span covers.

```
local function hide_text (start, w)
  local x, height, depth = 0, 0, 0
  local prev = nil
  local n    = start
  while n and x < w do
    local nxt = n.next
    if nested_pair(n) then
      nxt = nxt.next
      if prev then
        prev.next = nxt
      else
        start = nxt
      end
    else
      local width = node_width(n)
      if n.id == GLYPH then
        height = math.max(height, n.height or 0)
        depth  = math.max(depth,  n.depth  or 0)
        local k = node.new("kern")
        k.kern  = width
        copy_attributes(k, n)
        k.next  = nxt
        if prev then
          prev.next = k
        else
          start = k
        end
        node.free(n)
        prev = k
      else
        if n.id == HLIST or n.id == VLIST then
          height = math.max(height, n.height or 0)
          depth  = math.max(depth,  n.depth  or 0)
          if n.id == VLIST
             or node.has_attribute(n, M.dot_attr) == nil then
            n.head = hide_list(n.head)
          end
        end
        prev = n
      end
      x = x + width
```

```
    end
    n = nxt
  end
  if height ~= 0 or depth ~= 0 then
    local r = node.new("rule")
    r.width, r.height, r.depth = 0, height, depth
    if prev then
      r.next    = prev.next
      prev.next = r
    else
      r.next = start
      start  = r
    end
  end
  return start
end

```

(*End of definition for* `hide_text`.)

`dot_box` One emphasis dot, written into the text after the character it belongs to. The box takes no space and adds neither height nor depth, so the decorated text keeps its width and the line keeps its leading; the dot is centred on the advance of the character by a kern of half of it, and the inner box carries the ink at the depth of the call. The attributes travel from the character, so a dot takes the color of its text, and `format` replaces the color attribute with the value the TEX side resolved. The dot itself is marked, so that a box the pass sees twice is not dotted twice.

```
local function dot_box (glyph, dots)
  local outer = node.new("hlist")
  outer.width, outer.height, outer.depth = 0, 0, 0
  outer.dir = "TLT"
  local kern = node.new("kern")
  kern.kern  = -math.floor(((glyph.width or 0) + dots.size) / 2 + 0.5)
  local box = node.new("hlist")
  box.width, box.height, box.depth = dots.size, dots.size, 0
  box.shift = dots.top
  box.dir   = "TLT"
  local rule = node.new("rule")
  rule.width, rule.height, rule.depth = dots.size, dots.size, 0
  copy_attributes(kern, glyph)
  copy_attributes(box, glyph)
  copy_attributes(outer, glyph)
  copy_attributes(rule, glyph)
  if dots.color > 0 and M.color_attr then
```

```
    node.set_attribute(rule, M.color_attr, dots.color)
  end
  node.set_attribute(outer, M.dot_attr, 1)
  box.head   = rule
  kern.next  = box
  outer.head = kern
  return outer
end
```

(*End of definition for* `dot_box`.)

`insert_dots` Write one dot under each CJK character of the span of a leaders pair. The walk stops at the width of the pair, and an inner box of the text is walked with the interval of its own, so a character inside a box is dotted as well. A dot goes where the centre of its character lies inside the span that `subtract` leaves, and `skip` leaves the punctuation marks undotted; the two bounds are measured on the span and travel into a box unchanged, because the centre is measured from the start of the span and not from the start of the box. A character that already carries a dot behind it is passed over, because the pass also runs when a box is packed a second time. A decoration nested in the span adds no width of its own, so the walk steps over its pair and goes on with the characters below it.

```
local function insert_dots (start, w, settings, dots, offset, low, high)
  offset = offset or 0
  if not low then
    local subtract = settings.subtract or 0
    low  = math.min(subtract, w)
    high = math.max(w - subtract, low)
  end
  local x        = 0
  local n        = start
  while n and x < w do
    local nxt = n.next
    if nested_pair(n) then
      nxt = nxt.next
    else
      local width = node_width(n)
      if n.id == GLYPH and width > 0
         and LuaCJK.is_cjk(n.char, LuaCJK.settings_of(n), n.font) then
        local centre = offset + x + width / 2
        local dotted = nxt and nxt.id == HLIST
                       and node.has_attribute(nxt, M.dot_attr) ~= nil
        if centre >= low and centre <= high and not dotted
           and not (settings.skip and kind(n.char)) then
```

```
5047            local dot = dot_box(n, dots)
5048            dot.next = nxt
5049            n.next   = dot
5050          end
5051        elseif n.id == HLIST then
5052          insert_dots(n.head, n.width or 0, settings, dots, offset + x,
5053            low, high)
5054        end
5055        x = x + width
5056      end
5057      n = nxt
5058    end
5059  end
5060
```

(*End of definition for* `insert_dots`.)

`replace_pair` Replace the leaders glue of one decoration. The replacement covers the same span, with the same natural width, stretch, and shrink, in the same orders: the pieces are leaders of the intervals that `segments` allows, with their share of the dimensions of the pair, and kerns where a mark is cut out or the span is shortened. The last piece takes the remainder, so the sums are exact, and the negative glue that follows the leaders is kept as the tail.

A pair whose leader box reduces to one solid rule gets a leader box of zero width instead: TEX fills such a rule to the whole width of the glue, so the rule covers the interval exactly and grows with the glue, where a repeated rule box would leave the last fraction of the interval empty.

The ink of a pair whose decoration asked for room is shifted down by ⟨*offset*⟩ before it is written: a rule takes it in its height and depth, and a leader box in its shift, which is positive downwards.

```
5061  local function replace_pair (head, prev, leaders, settings, offset)
5062    local w     = leaders.width
5063    local neg   = leaders.next
5064    local subtract = settings and settings.subtract or 0
5065    local skip     = settings and settings.skip or false
5066    local parts = segments(w, neg.next, subtract, skip)
5067    if settings and settings.hidden then
5068      neg.next = hide_text(neg.next, w)
5069    end
5070    local rule, down = box_rule(leaders.leader)
5071    local block
5072    if rule then
5073      block = node.copy(rule)
5074      block.width  = 0
```

```
5075      block.height = rule.height - down - offset
5076      block.depth  = rule.depth + down + offset
5077      block.next   = nil
5078    end
5079    local covered = {}
5080    for _, part in ipairs(parts) do
5081      if part[2] > part[1] then
5082        covered[#covered + 1] = part
5083      end
5084    end
5085    local stretch = leaders.stretch or 0
5086    local shrink  = leaders.shrink or 0
5087    local given_st, given_sh = 0, 0
5088    local first, last
5089    local cursor = 0
5090    local function append (node_)
5091      if last then
5092        last.next = node_
5093      else
5094        first = node_
5095      end
5096      last = node_
5097    end
5098    local function gap (to)
5099      if to > cursor then
5100        local k = node.new("kern")
5101        k.kern  = to - cursor
5102        append(k)
5103        cursor = to
5104      end
5105    end
5106    for i, part in ipairs(covered) do
5107      gap(part[1])
5108      local piece    = node.copy(leaders)
5109      piece.width    = part[2] - part[1]
5110      piece.next     = nil
5111      if i < #covered then
5112        piece.stretch  = math.floor(stretch * piece.width / w + 0.5)
5113        piece.shrink   = math.floor(shrink  * piece.width / w + 0.5)
5114        given_st = given_st + piece.stretch
5115        given_sh = given_sh + piece.shrink
5116      else
5117        piece.stretch  = stretch - given_st
5118        piece.shrink   = shrink - given_sh
5119      end
```

```
    if block then
      piece.leader = node.copy(block)
      copy_attributes(piece, piece.leader)
    elseif offset ~= 0 and piece.leader then
      if piece.leader.id == RULE then
        local rule_ = node.copy(piece.leader)
        rule_.height = rule_.height - offset
        rule_.depth  = rule_.depth  + offset
        piece.leader = rule_
      else
        piece.leader       = node.copy(piece.leader)
        piece.leader.shift = (piece.leader.shift or 0) + offset
      end
    end
    append(piece)
    cursor = part[2]
  end
  gap(w)
  if #covered == 0 then
    -- Every interval was cut away, so the pair keeps its dimensions in one
    -- glue of no width, or the line would lose the stretch it had.
    local g = node.new("glue")
    g.width, g.stretch, g.shrink = 0, stretch, shrink
    g.stretch_order, g.shrink_order =
      leaders.stretch_order or 0, leaders.shrink_order or 0
    append(g)
  end
  if first then
    if prev then
      prev.next = first
    else
      head = first
    end
    last.next = neg
  end
  return head
end
```

(*End of definition for* `replace_pair`.)

`collect_place` The spans of one list, and the ink each of them draws. A decoration whose entry has `commute` off does not commute with the decorations it covers: the inks of one group are packed inside the band their own positions make, outermost first, with the leftover shared as equal gaps. That needs every ink of the group before the

first of them is placed, which is why the pass collects the spans first and writes them afterwards.

The spans are intervals of the list — lua-ul writes the pair of an inner decoration inside the span of the outer one — so the ones still open are kept with the width at which each ends, and each span becomes a frame that holds its children. `ink_of` gives the canonical top and bottom of its ink; a drawing has none, keeps its place, and is left out of the packing.

```
local function collect (head)
  local roots, order, open = { }, { }, { }
  local x = 0
  local n = head
  while n do
    local nxt = n.next
    while #open > 0 and x >= open[#open].stop do
      table.remove(open)
    end
    if n.id == GLUE
       and (n.subtype == LEADERS or n.subtype == CLEADERS
            or n.subtype == XLEADERS)
       and n.next
       and n.next.id == GLUE and n.next.width == -n.width then
      local parent = open[#open]
      local frame  = { settings = settings_at(n), children = { } }
      frame.top, frame.bottom = ink_of(frame.settings, n.leader)
      if frame.top then
        frame.height = frame.bottom - frame.top
      end
      frame.stop = x + n.width
      order[#order + 1] = frame
      if parent then
        parent.children[#parent.children + 1] = frame
      else
        roots[#roots + 1] = frame
      end
      open[#open + 1] = frame
      nxt = n.next.next
    else
      x = x + node_width(n)
    end
    n = nxt
  end
  return roots, order
end

```

```
% Pack one group into its band and give every span its offset.
%    \begin{macrocode}
local function place (root)
  local order = { }
  local function flatten (frame)
    order[#order + 1] = frame
    for _, child in ipairs(frame.children) do
      flatten(child)
    end
  end
  flatten(root)
  local top, bottom, heights, inked = nil, nil, 0, 0
  for _, frame in ipairs(order) do
    if frame.height then
      top     = top and math.min(top, frame.top) or frame.top
      bottom  = bottom and math.max(bottom, frame.bottom) or frame.bottom
      heights = heights + frame.height
      inked   = inked + 1
    end
  end
  if not top then
    return
  end
  if bottom < top + heights then
    bottom = top + heights
  end
  local gap    = inked > 1 and (bottom - top - heights) / (inked - 1) or 0
  local cursor = top
  for _, frame in ipairs(order) do
    if frame.height then
      frame.offset = cursor - frame.top
      cursor       = cursor + frame.height + gap
    else
      frame.offset = 0
    end
  end
end
```

(*End of definition for* `collect` *and* `place`.)

`adjust` Walk a horizontal list and rewrite every leaders pair whose own leaders glue carries a settings entry. A pair of the emphasis dots keeps the nodes lua-ul made, because its leader box is empty and there is no pattern to place; what the pass adds is the ink, one dot behind each CJK character of the span. The entry is read from the pair and not from the text it covers, so the outer of two nested

decorations keeps its own inset, its own `skip`, and its own dots instead of the ones the inner call left on the characters. The offset comes from the frame of the span, and only a group whose outermost decoration has `commute` off is packed: the flag is a setting of the package, so the members of a group share it, and a group that is not packed costs one walk over its spans and no measurement at all.

```
local function adjust (head)
  if not M.attr then
    return head
  end
  local roots, order = collect(head)
  for _, root in ipairs(roots) do
    if root.settings and root.settings.commute == false then
      place(root)
    end
  end
  local prev = nil
  local n    = head
  local i    = 0
  while n do
    local nxt = n.next
    if n.id == GLUE
       and (n.subtype == LEADERS or n.subtype == CLEADERS
            or n.subtype == XLEADERS)
       and n.next
       and n.next.id == GLUE and n.next.width == -n.width then
      local neg      = n.next
      local settings = settings_at(n)
      local w        = n.width
      i = i + 1
      local frame  = order[i]
      local offset = frame and frame.offset or 0
      if settings and settings.dots then
        local dots = settings.dots
        if offset ~= 0 then
          dots = { size = dots.size, top = dots.top + offset,
                   color = dots.color }
        end
        insert_dots(neg.next, w, settings, dots, 0)
        if settings.hidden then
          neg.next = hide_text(neg.next, w)
        end
        n    = neg
        nxt  = neg.next
```

```
      elseif settings or box_rule(n.leader) then
        head = replace_pair(head, prev, n, settings, offset)
        n    = neg
        nxt  = neg.next
      end
    end
    prev = n
    n    = nxt
  end
  return head
end
```

(*End of definition for* `adjust`.)

`glue_width` The width a glue node actually has in a packed box. The node keeps its natural width, and the box records how much of the glue's stretch or shrink was used: `glue_set` is the fraction, and `glue_sign` says whether it is applied to the stretch (1) or to the shrink (2), against the glue whose order is `glue_order`. This is the only place where the final width of a decoration is available, since it is fixed when the line is packed.

```
local function glue_width (box, g)
  local w    = g.width or 0
  local set  = box.glue_set or 0
  local sign = box.glue_sign or 0
  local order = box.glue_order or 0
  if sign == 1 and (g.stretch_order or 0) == order then
    w = w + math.floor(set * (g.stretch or 0) + 0.5)
  elseif sign == 2 and (g.shrink_order or 0) == order then
    w = w - math.floor(set * (g.shrink or 0) + 0.5)
  end
  return w
end
```

(*End of definition for* `glue_width`.)

`drawn` Whether a leader box holds a drawing rather than ink of its own. A drawing reaches the page as PDF literals, so the box holds whatsits and no glyph and no rule; the dots and the symbols hold a rule or a glyph, and a symbol wrapped in an `ActualText` sequence holds both. The name is declared with the ink walk, which asks a box this question before it trusts the dimensions it reads.

```
drawn = function (box)
  if not box or box.id ~= HLIST then
```

```
    return false
  end
  local literal, ink = false, false
  local n = box.head
  while n do
    if n.id == GLYPH or n.id == RULE then
      ink = true
    elseif n.id == WHATSIT then
      literal = true
    elseif n.id == HLIST then
      local l, i = drawn(n)
      literal = literal or l
      ink     = ink or i
    end
    n = n.next
  end
  return literal, ink
end

```

(*End of definition for* `drawn`.)

`literal` One PDF literal, placed at the current point: the mode `0` the backend wraps in a translation, which is the mode l3draw uses for its own operations.

```
local function literal (data)
  local n = node.new("whatsit", node.subtype("pdf_literal"))
  n.data  = data
  n.mode  = 0
  return n
end

```

(*End of definition for* `literal`.)

`scale_ink` Make the period of a copy equal to the width of its slot. A leader box is one period wide, and the slot the append callback divides the piece into is almost never the same width: the two differ by the remainder divided by the number of copies, which is a hundredth of a point or less, but it is still a phase shift between two copies and shows as a hairline gap or a thin doubling in the drawing. The ink of the copy is therefore painted inside a scope that scales it horizontally by ⟨*width*⟩ divided by the period: the path then ends exactly where the next copy's begins, and its period is the slot's. The stroke keeps its thickness, because only the horizontal axis is scaled; the pen's horizontal extent changes by the same fraction.

```
local function scale_ink (box, scale, advance)
```

```
  if not box or box.id ~= HLIST or not box.head then
    return
  end
  if math.abs(scale - 1) < 0.0000005 then
    return
  end
  local q = literal("q " .. string.format("%.6f", scale) .. " 0 0 1 0 0 cm")
  local z = literal("Q 1 0 0 1 " .. string.format("%.6f", advance) .. " 0 cm")
  q.next = box.head
  local last = q
  while last.next do
    last = last.next
  end
  last.next  = z
  box.head   = q
end
```

(*End of definition for* `scale_ink`.)

`tile` `write_copies` Write the copies of a repeated decoration into the piece it covers. The append callback is the one place where the width of the piece is final, so there it is divided into the nearest whole number of patterns and the piece is rebuilt from copies of the leader box, each as wide as that division: the copies are adjacent, the pattern reaches both ends of the span, and its period follows the stretch of the line. The width is rounded down, so that the remainder is a few scaled points at most and is left as a kern.

The copies are written here rather than repeated by `\leaders`, because a copy of a leader glue is aligned at the left edge of the enclosing box and not at the left edge of the glue, and only a copy that lies wholly inside the glue is drawn: a span that is not a whole number of patterns loses one at one end, and a span that does not begin on the grid loses every copy that straddles its edge, which is what a symbol of a single indented character showed. `\xleaders` cannot spread the pattern either, because the glue it puts between two copies separates two butt caps.

The ink of a drawn copy is scaled to the width it is given, so that one copy's path ends exactly where the next one's begins: the period of the drawing is one em, and the width the piece is divided into is almost never the same. A symbol is a glyph, which does not scale, so its copy keeps its size and is centred in its cell instead.

A drawn copy also loses the record of the drawing's origin, which l3draw puts in front of every drawing: it is a box of no width that holds nothing but whatsits,

and a label that every copy records is multiply defined.

```
local function only_whatsits (head)
  local n = head
  while n do
    if n.id ~= WHATSIT then
      return false
    end
    n = n.next
  end
  return true
end

local function strip_record (head)
  local n = head
  while n do
    local nxt = n.next
    if n.id == HLIST then
      if (n.width or 0) == 0 and n.head and n.head.id == WHATSIT
         and only_whatsits(n.head) then
        head = node.remove(head, n)
      else
        n.head = strip_record(n.head)
      end
    end
    n = nxt
  end
  return head
end

local function write_copies (box, prev, n, w, scale)
  local period = n.leader.width
  local k      = math.floor(w / period + 0.5)
  local width  = math.floor(w / k)
  local first, last
  for _ = 1, k do
    local copy = node.copy(n.leader)
    copy.head  = node.copy_list(n.leader.head)
    copy.width = width
    if scale then
      copy.head = strip_record(copy.head)
      scale_ink(copy, width / period, period / 65536 * 72 / 72.27)
    elseif width ~= period then
      local shift = node.new("kern")
      shift.kern  = math.floor((width - period) / 2)
      shift.next  = copy.head
```

```
5385        copy.head   = shift
5386      end
5387      copy_attributes(copy, n)
5388      copy.next = nil
5389      if last then
5390        last.next = copy
5391      else
5392        first = copy
5393      end
5394      last = copy
5395    end
5396    local rest = w - k * width
5397    if rest > 0 then
5398      local tail = node.new("kern")
5399      tail.kern  = rest
5400      last.next  = tail
5401      last       = tail
5402    end
5403    last.next = n.next
5404    if prev then
5405      prev.next = first
5406    else
5407      box.head  = first
5408    end
5409    return last
5410  end
5411
5412  local function tile (box)
5413    local prev = nil
5414    local n    = box.head
5415    while n do
5416      local nxt = n.next
5417      if n.id == GLUE and n.subtype == LEADERS and n.leader then
5418        local literal, ink = drawn(n.leader)
5419        local pattern = literal and not ink
5420        local symbol  = ink and not box_rule(n.leader)
5421        if pattern or symbol then
5422          local w      = glue_width(box, n)
5423          local period = n.leader.width
5424          if period > 0 and w >= period then
5425            n = write_copies(box, prev, n, w, pattern)
5426          end
5427        end
5428      end
5429      prev = n
```

```
    n    = nxt
  end
end
```

(*End of definition for* `tile` *and* `write_copies`.)

`LuaCJKfntef.hpack` The box-packing callback. The pass also runs before a box is appended to a vertical list, because a box built earlier is not packed again; that is where a decoration is divided into whole patterns.

```
function M.hpack (head)
  return adjust(head)
end

luatexbase.add_to_callback("hpack_filter", M.hpack, "LuaCJKfntef.adjust")
luatexbase.add_to_callback("pre_append_to_vlist_filter",
  function (box)
    if box.id == node.id("hlist") then
      box.head = adjust(box.head)
      tile(box)
    end
    return box
  end, "LuaCJKfntef.adjust")
⟨/lua-fntef⟩
```

(*End of definition for* `LuaCJKfntef.hpack`.)

The *LuaCJKfntef package is a thin layer over lua-ul, which provides the underlining engine. It is generated from this file and may be loaded on its own; it loads LuaCJK first.

```
⟨*fntef⟩
\sys_if_engine_luatex:F
  {
    \msg_new:nnn { LuaCJKfntef } { engine }
      { LuaCJKfntef ~ requires ~ LuaTeX. }
    \msg_critical:nn { LuaCJKfntef } { engine }
    \tex_endinput:D
  }
\RequirePackage { LuaCJK, lua-ul, luacolor, l3draw }
\lua_load_module:n { LuaCJKfntef }
\newattribute \g__LuaCJK_fntef_settings_attr
\int_const:Nn \c__LuaCJK_fntef_settings_attr_int { \int_use:N \allocationnumber }
\g__LuaCJK_fntef_settings_attr = 0
\lua_now:e
  { LuaCJKfntef.attr = \int_use:N \c__LuaCJK_fntef_settings_attr_int }
```

\LuaCJKfntefsetup The keys of the package as a whole, as opposed to the keys of one decoration. \l__LuaCJK_fntef_commute_bool is a local variable, so a setting in a group covers the decorations started inside that group and no other.

```
\bool_new:N \l__LuaCJK_fntef_commute_bool
\bool_set_true:N \l__LuaCJK_fntef_commute_bool
\keys_define:nn { LuaCJKfntef / options }
  {
    commute  .bool_set:N = \l__LuaCJK_fntef_commute_bool,
    commute  .default:n  = { true },
  }
\NewDocumentCommand \LuaCJKfntefsetup { m }
  { \keys_set:nn { LuaCJKfntef / options } {#1} }
```

(*End of definition for* \LuaCJKfntefsetup*. This function is documented on page 43.*)

\CJKunderline Underline the argument. The optional argument takes thickness, depth, textformat, and any other key is accepted and ignored, like the key of any other decoration. The requested geometry, the format of the text, the color that an underline type is cached by, the color operations of a drawn pattern, and the name of the settings entry of the call. The emphasis dots add whether the ink is a dot per CJK character rather than a pattern, the side and the position of one dot, and the color attribute of the dots when format is given.

```
\dim_new:N \l__LuaCJK_fntef_depth_dim
\dim_new:N \l__LuaCJK_fntef_thickness_dim
\tl_new:N \l__LuaCJK_fntef_textformat_tl
\tl_new:N \l__LuaCJK_fntef_format_tl
\dim_new:N \l__LuaCJK_fntef_subtract_dim
\tl_new:N \l__LuaCJK_fntef_skip_tl
\tl_new:N \l__LuaCJK_fntef_skip_js_tl
\tl_new:N \l__LuaCJK_fntef_hidden_js_tl
\bool_new:N \l__LuaCJK_fntef_silent_bool
\bool_new:N \l__LuaCJK_fntef_hidden_bool
\bool_new:N \l__LuaCJK_fntef_top_given_bool
\bool_new:N \l__LuaCJK_fntef_bottom_given_bool
\dim_new:N \l__LuaCJK_fntef_rule_height_dim
\dim_new:N \l__LuaCJK_fntef_rule_depth_dim
\prop_new:N \g__LuaCJK_fntef_format_prop
\int_new:N \g__LuaCJK_fntef_format_int
\tl_new:N \l__LuaCJK_fntef_cache_key_tl
\tl_new:N \l__LuaCJK_fntef_cache_name_tl
\tl_new:N \l__LuaCJK_fntef_color_key_tl
\tl_new:N \l__LuaCJK_fntef_color_ops_tl
\tl_new:N \g__LuaCJK_fntef_color_ops_tl
\tl_new:N \l__LuaCJK_fntef_settings_key_tl
```

```
\dim_new:N \l__LuaCJK_fntef_top_dim
\dim_new:N \l__LuaCJK_fntef_bottom_dim
\dim_new:N \l__LuaCJK_fntef_gap_dim
\dim_new:N \l__LuaCJK_fntef_height_dim
\tl_new:N \l__LuaCJK_fntef_symbol_tl
\bool_new:N \l__LuaCJK_fntef_dots_bool
\tl_new:N \l__LuaCJK_fntef_dots_js_tl
\dim_new:N \l__LuaCJK_fntef_dots_size_dim
\dim_new:N \l__LuaCJK_fntef_dots_top_dim
\int_new:N \g__LuaCJK_fntef_dots_color_int
\dim_set:Nn \l__LuaCJK_fntef_depth_dim     { .2em }
\dim_set:Nn \l__LuaCJK_fntef_thickness_dim { .4pt }
\dim_set:Nn \l__LuaCJK_fntef_gap_dim       { 1.1pt }
\dim_set:Nn \l__LuaCJK_fntef_height_dim    { .3em }
```

(*End of definition for* `\CJKunderline` *and others. This function is documented on page* 43.)

Every decoration takes the same options, so one key set serves them all: the keys a command does not use are accepted and ignored, and a bare item that is not a key is a color name, which is `format`.

`\__LuaCJK_fntef_option_pair:nn`
`\__LuaCJK_fntef_option_bare:n`
`\__LuaCJK_fntef_options:n`

Parse the option list of a decoration and reset the defaults first. A pair goes to the key set as it is; a bare item is a key when it exists and a color name otherwise, so `[subtract]` keeps its meaning and `[red]` is `[format = red]`.

```
\cs_new_protected:Npn \__LuaCJK_fntef_option_pair:nn #1#2
  { \keys_set:nn { LuaCJKfntef / decoration } { #1 = {#2} } }
\cs_new_protected:Npn \__LuaCJK_fntef_option_bare:n #1
  {
    \keys_if_exist:nnTF { LuaCJKfntef / decoration } {#1}
      { \keys_set:nn { LuaCJKfntef / decoration } {#1} }
      { \keys_set:nn { LuaCJKfntef / decoration } { format = #1 } }
  }
\cs_new_protected:Npn \__LuaCJK_fntef_options:n #1
  {
    \__LuaCJK_fntef_reset_decoration:
    \tl_if_novalue:nF {#1}
      {
        \keyval_parse:NNn
          \__LuaCJK_fntef_option_bare:n \__LuaCJK_fntef_option_pair:nn {#1}
      }
  }
```

(*End of definition for* `\__LuaCJK_fntef_option_pair:nn`, `\__LuaCJK_fntef_option_bare:n`, *and* `\__LuaCJK_fntef_options:n`.)

__LuaCJK_fntef_body:n

The text of a decoration, with the format of the call applied. Every command goes through this, so `textformat` means the same thing everywhere; `hidden` is a setting of the decoration pass instead, because a hidden text has to keep its line breaks.

```
\cs_new_protected:Npn \__LuaCJK_fntef_body:n #1
  { \l__LuaCJK_fntef_textformat_tl #1 }
```

(*End of definition for* `\__LuaCJK_fntef_body:n`.)

__LuaCJK_fntef_push_settings:

Hand the decoration settings to Lua, or clear the attribute when they are all at their defaults. The dots are settings of their own: when `\CJKunderdot` gives no `symbol`, the pass puts one dot under each CJK character rather than repeating a pattern, so the entry carries the side of a dot, the distance from the baseline to its top, and the color attribute of `format`.

The seven values and the `commute` flag also make `\l__LuaCJK_fntef_settings_key_tl`, the name of the entry: a call that sets a decoration the way an earlier one did selects the entry registered then instead of registering a second one with the same content. The name is part of the cache key of the type a call builds, so the leaders pair of a decoration names the entry of its own call, whatever the text inside it carries. A decoration whose ink must stay clear of the ink it covers has to say so even when every other setting is at its default, so `commute` is part of the test that lets a call write no entry at all.

```
\cs_new_protected:Npn \__LuaCJK_fntef_push_settings:
  {
    \bool_lazy_and:nnTF
      { \l__LuaCJK_fntef_dots_bool }
      { \tl_if_empty_p:N \l__LuaCJK_fntef_symbol_tl }
      {
        \tl_set:Nn \l__LuaCJK_fntef_dots_js_tl { true }
        \dim_set:Nn \l__LuaCJK_fntef_dots_size_dim { .06em }
        \dim_set:Nn \l__LuaCJK_fntef_dots_top_dim
          { \LuaCJKemdescent + \l__LuaCJK_fntef_depth_dim }
        \tl_if_empty:NF \l__LuaCJK_fntef_format_tl
          {
            \group_begin:
              \luaul_set_color:n { \l__LuaCJK_fntef_format_tl }
              \luaul_apply_color:
              \int_gset:Nn \g__LuaCJK_fntef_dots_color_int
                { \tex_the:D \use:c { LuaCol@Attribute } }
            \group_end:
          }
      }
      {
```

```
      \tl_set:Nn \l__LuaCJK_fntef_dots_js_tl { false }
      \dim_zero:N \l__LuaCJK_fntef_dots_size_dim
      \dim_zero:N \l__LuaCJK_fntef_dots_top_dim
    }
  \bool_lazy_all:nTF
    {
      { \dim_compare_p:nNn { \l__LuaCJK_fntef_subtract_dim } = { 0pt } }
      { \str_if_eq_p:Vn \l__LuaCJK_fntef_skip_tl { 0 } }
      { \bool_not_p:n { \l__LuaCJK_fntef_hidden_bool } }
      { \str_if_eq_p:Vn \l__LuaCJK_fntef_dots_js_tl { false } }
      { \l__LuaCJK_fntef_commute_bool }
    }
    {
      \tl_clear:N \l__LuaCJK_fntef_settings_key_tl
      \lua_now:e
        {
          tex.setattribute
            ( \int_use:N \c__LuaCJK_fntef_settings_attr_int, 0 )
        }
    }
    {
      \str_if_eq:VnTF \l__LuaCJK_fntef_skip_tl { 1 }
        { \tl_set:Nn \l__LuaCJK_fntef_skip_js_tl { true  } }
        { \tl_set:Nn \l__LuaCJK_fntef_skip_js_tl { false } }
      \bool_if:NTF \l__LuaCJK_fntef_hidden_bool
        { \tl_set:Nn \l__LuaCJK_fntef_hidden_js_tl { true  } }
        { \tl_set:Nn \l__LuaCJK_fntef_hidden_js_tl { false } }
      \tl_set:Ne \l__LuaCJK_fntef_settings_key_tl
        {
          \dim_to_decimal_in_sp:n { \l__LuaCJK_fntef_subtract_dim } /
          \l__LuaCJK_fntef_skip_js_tl / \l__LuaCJK_fntef_hidden_js_tl /
          \l__LuaCJK_fntef_dots_js_tl /
          \dim_to_decimal_in_sp:n { \l__LuaCJK_fntef_dots_size_dim } /
          \dim_to_decimal_in_sp:n { \l__LuaCJK_fntef_dots_top_dim } /
          \int_use:N \g__LuaCJK_fntef_dots_color_int /
          \bool_if:NTF \l__LuaCJK_fntef_commute_bool { true } { false }
        }
      \lua_now:e
        {
          LuaCJKfntef.new_settings
            ( [[ \l__LuaCJK_fntef_settings_key_tl ]] ,
              \dim_to_decimal_in_sp:n { \l__LuaCJK_fntef_subtract_dim } ,
              \l__LuaCJK_fntef_skip_js_tl ,
              \l__LuaCJK_fntef_hidden_js_tl ,
              \l__LuaCJK_fntef_dots_js_tl ,
```

```
                \dim_to_decimal_in_sp:n { \l__LuaCJK_fntef_dots_size_dim } ,
                \dim_to_decimal_in_sp:n { \l__LuaCJK_fntef_dots_top_dim } ,
                \int_use:N \g__LuaCJK_fntef_dots_color_int ,
                \bool_if:NTF \l__LuaCJK_fntef_commute_bool { true } { false } )
            }
        }
    }
\cs_new_protected:Npn \__LuaCJK_fntef_prepare_underline:n #1
    {
      \__LuaCJK_fntef_options:n {#1}
      \bool_if:NF \l__LuaCJK_fntef_top_given_bool
        { \dim_set:Nn \l__LuaCJK_fntef_top_dim { -\l__LuaCJK_fntef_depth_dim } }
      \bool_if:NF \l__LuaCJK_fntef_bottom_given_bool
        {
          \dim_set:Nn \l__LuaCJK_fntef_bottom_dim
            { -\l__LuaCJK_fntef_depth_dim - \l__LuaCJK_fntef_thickness_dim }
        }
      \__LuaCJK_fntef_push_settings:
      \dim_set:Nn \l__LuaCJK_fntef_rule_height_dim
        { \l__LuaCJK_fntef_top_dim }
      \dim_set:Nn \l__LuaCJK_fntef_rule_depth_dim
        { -\l__LuaCJK_fntef_bottom_dim }
    }
```

__LuaCJK_fntef_underline:

Build the type of the underline and use it: one rule of the height and the depth that the keys asked for, cached by geometry like every other decoration. A colored rule needs the type of its own: luacolor colors by attribute, while \color would emit whatsits in front of \leaders, which is what __LuaCJK_fntef_use_leader_type:nn does for a format.

```
\cs_new_protected:Npn \__LuaCJK_fntef_underline:
    {
      \tl_set:Ne \l__LuaCJK_fntef_any_code_tl
        {
          \exp_not:N \tex_leaders:D \exp_not:N \tex_vrule:D
            height \dim_use:N \l__LuaCJK_fntef_rule_height_dim
            depth  \dim_use:N \l__LuaCJK_fntef_rule_depth_dim
        }
      \tl_set:Ne \l__LuaCJK_fntef_cache_key_tl
        {
          underline |
          \dim_use:N \l__LuaCJK_fntef_rule_height_dim |
          \dim_use:N \l__LuaCJK_fntef_rule_depth_dim
        }
      \exp_args:NV \__LuaCJK_fntef_use_leader_type:nn
```

```
      \l__LuaCJK_fntef_cache_key_tl \l__LuaCJK_fntef_any_code_tl
  }
\NewDocumentCommand \CJKunderline { s t- o +m }
  {
    \group_begin:
      \__LuaCJK_fntef_prepare_underline:n {#3}
      \__LuaCJK_fntef_underline:
      \__LuaCJK_fntef_body:n {#4}
    \group_end:
  }
```

(*End of definition for* `\__LuaCJK_fntef_push_settings:` *and* `\__LuaCJK_fntef_underline:`.)

`\CJKsout` Strike through the argument.

```
\cs_new_protected:Npn \__LuaCJK_fntef_use_sout:
  {
    \tl_set:Ne \l__LuaCJK_fntef_cache_key_tl
      {
        sout |
        \dim_use:N \l__LuaCJK_fntef_height_dim |
        \dim_use:N \l__LuaCJK_fntef_thickness_dim |
        \l__LuaCJK_fntef_format_tl |
        \l__LuaCJK_fntef_settings_key_tl
      }
    \tl_if_empty:NF \l__LuaCJK_fntef_format_tl
      { \luaul_set_color:n { \l__LuaCJK_fntef_format_tl } }
    \prop_get:NVNF \g__LuaCJK_fntef_format_prop
      \l__LuaCJK_fntef_cache_key_tl \l__LuaCJK_fntef_cache_name_tl
      {
        \int_gincr:N \g__LuaCJK_fntef_format_int
        \tl_set:Ne \l__LuaCJK_fntef_cache_name_tl
          { LuaCJKfntef / st / \int_use:N \g__LuaCJK_fntef_format_int }
        \tl_if_empty:NTF \l__LuaCJK_fntef_format_tl
          {
            \exp_args:Nc \newunderlinetype
              { \l__LuaCJK_fntef_cache_name_tl }
              {
                \tex_leaders:D \tex_vrule:D
                  height \dim_eval:n
                    {
                      \l__LuaCJK_fntef_height_dim
                      + \l__LuaCJK_fntef_thickness_dim / 2
                    }
                  depth \dim_eval:n
                    {
```

```
                    -\l__LuaCJK_fntef_height_dim
                    + \l__LuaCJK_fntef_thickness_dim / 2
                  }
              }
          }
          {
            \exp_args:Nc \newunderlinetype
              { \l__LuaCJK_fntef_cache_name_tl }
              [ \dim_to_decimal_in_sp:n { 1ex } @ \luaul_context_color: ]
              {
                \luaul_apply_color:
                \tex_leaders:D \tex_vrule:D
                  height \dim_eval:n
                    {
                      \l__LuaCJK_fntef_height_dim
                      + \l__LuaCJK_fntef_thickness_dim / 2
                    }
                  depth \dim_eval:n
                    {
                      -\l__LuaCJK_fntef_height_dim
                      + \l__LuaCJK_fntef_thickness_dim / 2
                    }
              }
          }
        \prop_gput:NVV \g__LuaCJK_fntef_format_prop
          \l__LuaCJK_fntef_cache_key_tl \l__LuaCJK_fntef_cache_name_tl
      }
    \use:c { \l__LuaCJK_fntef_cache_name_tl }
  }
\NewDocumentCommand \CJKsout { s t- o +m }
  {
    \group_begin:
      \__LuaCJK_fntef_decoration_setup:n {#3}
      \__LuaCJK_fntef_use_sout:
      \__LuaCJK_fntef_body:n {#4}
    \group_end:
  }
```

(*End of definition for* `\CJKsout`. *This function is documented on page* 46.)

`__LuaCJK_fntef_use_double:` Build the two underline types of the double rule, one for each line, keyed by the geometry so that repeated calls reuse them.

```
\cs_new_protected:Npn \__LuaCJK_fntef_use_double:
  {
    \tl_set:Ne \l__LuaCJK_fntef_cache_key_tl
```

```
      {
        double-top |
        \dim_use:N \l__LuaCJK_fntef_depth_dim |
        \dim_use:N \l__LuaCJK_fntef_thickness_dim
      }
    \tl_set:Ne \l__LuaCJK_fntef_any_code_tl
      {
        \exp_not:N \tex_leaders:D \exp_not:N \tex_vrule:D
          height \dim_eval:n { -\l__LuaCJK_fntef_depth_dim }
          depth \dim_eval:n
            { \l__LuaCJK_fntef_depth_dim + \l__LuaCJK_fntef_thickness_dim }
      }
    \exp_args:NV \__LuaCJK_fntef_use_leader_type:nn
      \l__LuaCJK_fntef_cache_key_tl \l__LuaCJK_fntef_any_code_tl
    \tl_set:Ne \l__LuaCJK_fntef_cache_key_tl
      {
        double-bottom |
        \dim_use:N \l__LuaCJK_fntef_depth_dim |
        \dim_use:N \l__LuaCJK_fntef_thickness_dim |
        \dim_use:N \l__LuaCJK_fntef_gap_dim
      }
    \tl_set:Ne \l__LuaCJK_fntef_any_code_tl
      {
        \exp_not:N \tex_leaders:D \exp_not:N \tex_vrule:D
          height \dim_eval:n
            {
              -\l__LuaCJK_fntef_depth_dim
              - \l__LuaCJK_fntef_thickness_dim
              - \l__LuaCJK_fntef_gap_dim
            }
          depth \dim_eval:n
            {
              \l__LuaCJK_fntef_depth_dim
              + 2 \l__LuaCJK_fntef_thickness_dim
              + \l__LuaCJK_fntef_gap_dim
            }
      }
    \exp_args:NV \__LuaCJK_fntef_use_leader_type:nn
      \l__LuaCJK_fntef_cache_key_tl \l__LuaCJK_fntef_any_code_tl
  }
```

(*End of definition for* `\__LuaCJK_fntef_use_double:.`)

`\__LuaCJK_fntef_decoration_setup:n` The settings that the decoration pass reads. They are pushed on every call, so a command without options starts from the defaults instead of inheriting the

previous call.

```
\keys_define:nn { LuaCJKfntef / decoration }
  {
    subtract  .code:n             =
      {
        \str_if_eq:nnTF {#1} { true }
          { \dim_set:Nn \l__LuaCJK_fntef_subtract_dim { .5em } }
          {
            \str_if_eq:nnTF {#1} { false }
              { \dim_zero:N \l__LuaCJK_fntef_subtract_dim }
              { \dim_set:Nn \l__LuaCJK_fntef_subtract_dim {#1} }
          }
      },
    subtract  .default:n          = { true },
    skip      .code:n             =
      {
        \str_if_eq:nnTF {#1} { true }
          { \tl_set:Nn \l__LuaCJK_fntef_skip_tl { 1 } }
          { \tl_set:Nn \l__LuaCJK_fntef_skip_tl { 0 } }
      },
    skip      .default:n          = { true },
    hidden    .bool_set:N         = \l__LuaCJK_fntef_hidden_bool,
    hidden    .default:n          = { true },
    selectable.bool_set_inverse:N = \l__LuaCJK_fntef_silent_bool,
    selectable.default:n          = { true },
    depth     .dim_set:N          = \l__LuaCJK_fntef_depth_dim,
    depth     .value_required:n   = true,
    thickness .dim_set:N          = \l__LuaCJK_fntef_thickness_dim,
    thickness .value_required:n   = true,
    gap       .dim_set:N          = \l__LuaCJK_fntef_gap_dim,
    gap       .value_required:n   = true,
    height    .dim_set:N          = \l__LuaCJK_fntef_height_dim,
    height    .value_required:n   = true,
    top       .code:n             =
      {
        \dim_set:Nn \l__LuaCJK_fntef_top_dim {#1}
        \bool_set_true:N \l__LuaCJK_fntef_top_given_bool
      },
    top       .value_required:n   = true,
    bottom    .code:n             =
      {
        \dim_set:Nn \l__LuaCJK_fntef_bottom_dim {#1}
        \bool_set_true:N \l__LuaCJK_fntef_bottom_given_bool
      },
    bottom    .value_required:n   = true,
```

```
    format    .tl_set:N           = \l__LuaCJK_fntef_format_tl,
    color     .tl_set:N           = \l__LuaCJK_fntef_format_tl,
    color     .value_required:n   = true,
    symbol    .tl_set:N           = \l__LuaCJK_fntef_symbol_tl,
    sep       .dim_set:N          = \l__LuaCJK_fntef_symbol_sep_dim,
    sep       .value_required:n   = true,
    textformat.tl_set:N           = \l__LuaCJK_fntef_textformat_tl,
    unknown   .code:n             = { },
  }
\cs_new_protected:Npn \__LuaCJK_fntef_reset_decoration:
  {
    \dim_zero:N \l__LuaCJK_fntef_subtract_dim
    \tl_set:Nn \l__LuaCJK_fntef_skip_tl { 0 }
    \bool_set_true:N \l__LuaCJK_fntef_silent_bool
    \bool_set_false:N \l__LuaCJK_fntef_hidden_bool
    \bool_set_false:N \l__LuaCJK_fntef_top_given_bool
    \bool_set_false:N \l__LuaCJK_fntef_bottom_given_bool
    \bool_set_false:N \l__LuaCJK_fntef_dots_bool
    \int_gzero:N \g__LuaCJK_fntef_dots_color_int
    \dim_set:Nn \l__LuaCJK_fntef_depth_dim     { .2em }
    \dim_set:Nn \l__LuaCJK_fntef_thickness_dim { .4pt }
    \dim_set:Nn \l__LuaCJK_fntef_gap_dim       { 1.1pt }
    \dim_set:Nn \l__LuaCJK_fntef_height_dim    { .3em }
    \clist_map_inline:nn { format, symbol, textformat }
      { \tl_clear:c { l__LuaCJK_fntef_##1_tl } }
    \dim_set:Nn \l__LuaCJK_fntef_symbol_sep_dim { .1em }
  }
\cs_new_protected:Npn \__LuaCJK_fntef_decoration_setup:n #1
  {
    \__LuaCJK_fntef_options:n {#1}
    \__LuaCJK_fntef_push_settings:
  }
```

(*End of definition for* `\__LuaCJK_fntef_decoration_setup:n`.)

`_LuaCJK_fntef_dots_setup:n` The same for the emphasis dots, which are a setting of their own: the options are parsed first, because the reset clears the flag, and the flag is set before the settings go to Lua, because the entry carries the side, the position, and the color of a dot.

```
\cs_new_protected:Npn \__LuaCJK_fntef_dots_setup:n #1
  {
    \__LuaCJK_fntef_options:n {#1}
    \bool_set_true:N \l__LuaCJK_fntef_dots_bool
    \__LuaCJK_fntef_push_settings:
  }
```

(*End of definition for* `\__LuaCJK_fntef_dots_setup:n`.)

\CJKunderdblline Two parallel rules. The optional argument takes the whole key set of the other decorations; * and - are accepted for compatibility.

```
\NewDocumentCommand \CJKunderdblline { s t- o +m }
  {
    \group_begin:
      \__LuaCJK_fntef_decoration_setup:n {#3}
      \__LuaCJK_fntef_use_double:
      \__LuaCJK_fntef_body:n {#4}
    \group_end:
  }
```

(*End of definition for* `\CJKunderdblline`*. This function is documented on page 45.*)

\CJKunderwave A wave drawn with l3draw, one period per em. The drawing is kept as code rather than as a box, so the `em` units are resolved when the decoration is used and the pattern follows the current font size.

The drawing goes through `\__LuaCJK_fntef_use_drawn_type:nn` rather than `\__LuaCJK_fntef_use_leader_type:nn`, because its ink is stroked and not a rule that luacolor could color by attribute; the color travels into the drawing itself, so the pattern is built once per font size and color.

The pattern is a pair of arches, each of them the quadratic that runs from one zero crossing to the next one through its apex, written as the cubic that is the same curve: its control points sit at a third and two thirds of the half period, two thirds of the amplitude high, so the curve leaves and meets the zero line at the same slope on both sides of a crossing and the wave has no cusp. The amplitude is a quarter em, so the wave is half an em high from crest to trough, half as high as its period is wide, which is the proportion the reference implementation gives it.

The period is forced to exactly one em by `\hbox to 1em`. l3draw sizes its box by the bounding box of the stroked path, which is a full stroke width wider than the path; repeating that box would leave a gap of one stroke width between two periods. The two `\hss` absorb the excess symmetrically, so no overfull box is reported. The stroke is set to `.04em`, which is `0.4pt` at 10pt, so the pattern keeps its look from the previous release while it scales with the font size.

The period is the width of a CJK character, so a span of whole characters that justification has not stretched, which is the last line of a paragraph, is an exact number of periods: its wave starts at the first character's edge, ends at the last one's, and puts a crest at the same place in every character.

A span that justification has stretched is another matter: TEX repeats only whole copies of a leader box, so the remainder of such a span is the stretch the piece carries, which at one em is up to a character. `\leaders` would leave all of it empty at the right end and `\cleaders` half of it at each end, where the wave stops before

the last character. `\xleaders` divides it between the copies, but that glue separates two butt caps: a cap ends where its path ends, so every gap is a hole in the stroke, and at a tenth of a point it is already visible at the crossings. The piece is therefore divided into whole patterns instead, by `tile` (see the decoration pass): it has the final width and the nearest whole number of periods, and it writes one copy of the leader box per period at that width. The wave then reaches the last character, no glue falls between two periods, and the period follows the line: on a paragraph set ten centimetres wide it is `9.92pt` on a line that was shrunk by a per cent and `10pt` on the last line, which was not stretched at all.

The width the piece is divided into is not the period of the drawing, so the ink of the copy is painted inside a scope that scales it horizontally to that width. Without it the drawing keeps its own period and two copies are a hundredth of a point out of phase, which is a hairline gap or a thin doubling at every crossing; with it one copy's path ends exactly where the next one's begins. The glue the piece is made of has no shrink of its own, so its final width is read from the line's own `glue_set` and `glue_sign`, the latter saying whether the line was stretched or shrunk. A box that never reaches a vertical list, such as `\hbox to`, keeps the plain `\leaders` behaviour: it stops after its last whole pattern, without breaking the stroke. A scratch box for the drawn patterns. Each pattern is built here and then has its height and depth set to zero: the ink is a PDF literal, so only the reported dimensions change, and a decoration never opens up the line it sits on.

`\l__LuaCJK_fntef_pattern_box`

```
\box_new:N \l__LuaCJK_fntef_pattern_box
\dim_new:N \l__LuaCJK_fntef_symbol_kern_dim
\dim_new:N \l__LuaCJK_fntef_symbol_raise_dim
\dim_new:N \l__LuaCJK_fntef_em_descent_dim

\cs_new_protected:Npn \__LuaCJK_fntef_wave:
  {
    \tl_set:Nn \l__LuaCJK_fntef_any_code_tl
      {
        \tex_leaders:D \hbox_to_wd:nn { 1em }
          {
            \tex_hss:D
            \hbox_set:Nn \l__LuaCJK_fntef_pattern_box
              {
                \box_move_up:nn { -.42em }
                  {
                    \hbox:n
                      {
                        \draw_begin:
                          \__LuaCJK_fntef_pattern_color:
                          \draw_set_linewidth:n { .04em }
```

```
                            \draw_path_moveto:n { 0em , 0em }
                            \draw_path_curveto:nnn { .1667em , .1667em }
                              { .3333em , .1667em } { .5em , 0em }
                            \draw_path_curveto:nnn { .6667em , -
.1667em }
                              { .8333em , -.1667em } { 1em , 0em }
                            \draw_path_use_clear:n { stroke }
                          \draw_end:
                        }
                    }
                }
              \box_set_ht:Nn  \l__LuaCJK_fntef_pattern_box { \c_zero_dim }
              \box_set_dp:Nn  \l__LuaCJK_fntef_pattern_box { \c_zero_dim }
              \box_use_drop:N \l__LuaCJK_fntef_pattern_box
              \tex_hss:D
            }
        }
      \tl_set:Nn \l__LuaCJK_fntef_cache_key_tl { wave }
      \exp_args:NV \__LuaCJK_fntef_use_drawn_type:nn
        \l__LuaCJK_fntef_cache_key_tl \l__LuaCJK_fntef_any_code_tl
    }
\cs_new_protected:Npn \__LuaCJK_fntef_silent_begin:
    {
      \bool_if:NT \l__LuaCJK_fntef_silent_bool
        {
          \tex_pdfextension:D literal direct
            { /Span <</ActualText <FEFF200B>>> BDC }
        }
    }
\cs_new_protected:Npn \__LuaCJK_fntef_silent_end:
    {
      \bool_if:NT \l__LuaCJK_fntef_silent_bool
        { \tex_pdfextension:D literal direct { EMC } }
    }
\cs_new_protected:Npn \__LuaCJK_fntef_use_symbol:n #1
    {
      \dim_set:Nn \l__LuaCJK_fntef_any_depth_dim {#1}
      \dim_set:Nn \l__LuaCJK_fntef_em_descent_dim
        {
          \lua_now:e
            {
              tex.print(LuaCJKfntef.em_descent
                ( \int_use:N \tex_attribute:D \c__LuaCJK_font_attr_int ))
            }
          sp
```

```
    }
  \tl_set:Ne \l__LuaCJK_fntef_any_code_tl
    {
      \exp_not:N \tex_leaders:D \exp_not:N \hbox:n
        {
          \exp_not:N \hbox_set:Nn \exp_not:N \l__LuaCJK_fntef_pattern_box
            { \exp_not:V \l__LuaCJK_fntef_symbol_tl }
          \exp_not:N \dim_set:Nn \exp_not:N \l__LuaCJK_fntef_symbol_kern_dim
            {
              \exp_not:N \dim_eval:n
                {
                  ( 1em
                    - \exp_not:N \box_wd:N \l__LuaCJK_fntef_pattern_box ) / 2
                }
            }
          \exp_not:N \dim_set:Nn \exp_not:N \l__LuaCJK_fntef_symbol_raise_dim
            {
              \exp_not:N \dim_eval:n
                {
                  - \dim_use:N \l__LuaCJK_fntef_em_descent_dim
                  - \exp_not:N \l__LuaCJK_fntef_any_depth_dim
                  - \exp_not:N \box_ht:N \l__LuaCJK_fntef_pattern_box
                }
            }
          \exp_not:N \box_set_ht:Nn \exp_not:N \l__LuaCJK_fntef_pattern_box
            { \exp_not:N \c_zero_dim }
          \exp_not:N \box_set_dp:Nn \exp_not:N \l__LuaCJK_fntef_pattern_box
            { \exp_not:N \c_zero_dim }
          \exp_not:N \skip_horizontal:n
            { \exp_not:N \l__LuaCJK_fntef_symbol_kern_dim }
          \exp_not:N \__LuaCJK_fntef_silent_begin:
          \exp_not:N \box_move_up:nn
            { \exp_not:N \l__LuaCJK_fntef_symbol_raise_dim }
            { \exp_not:N \box_use_drop:N \l__LuaCJK_fntef_pattern_box }
          \exp_not:N \__LuaCJK_fntef_silent_end:
          \exp_not:N \skip_horizontal:n
            { \exp_not:N \l__LuaCJK_fntef_symbol_kern_dim }
        }
    }
  \tl_set:Ne \l__LuaCJK_fntef_cache_key_tl
    {
      symbol |
      \dim_use:N \l__LuaCJK_fntef_em_descent_dim |
      \dim_use:N \l__LuaCJK_fntef_any_depth_dim |
      \bool_if:NTF \l__LuaCJK_fntef_silent_bool { silent } { text } |
```

```
        \tl_to_str:V \l__LuaCJK_fntef_symbol_tl
      }
    \exp_args:NV \__LuaCJK_fntef_use_leader_type:nn
      \l__LuaCJK_fntef_cache_key_tl \l__LuaCJK_fntef_any_code_tl
  }
\NewDocumentCommand \CJKunderwave { s t- o +m }
  {
    \group_begin:
      \__LuaCJK_fntef_decoration_setup:n {#3}
      \tl_if_empty:NTF \l__LuaCJK_fntef_symbol_tl
        { \__LuaCJK_fntef_wave: } { \__LuaCJK_fntef_use_symbol:n { .42em } }
      \__LuaCJK_fntef_body:n {#4}
    \group_end:
  }
```

(*End of definition for* `\CJKunderwave` *and* `\l__LuaCJK_fntef_pattern_box`*. This function is documented on page 45.*)

`\CJKxout` A repeated slash, as in ulem's `\xout`. The drawing is kept as code so that it scales with the font size, and the stroke width is `.04em` like the wave. The slash runs from `-.15em` to `.8em`, so it crosses the whole body of a CJK glyph instead of stopping in its upper half.

l3draw moves a drawing so that the bottom of its bounding box sits on the baseline of the box, which shifts a path that goes below the baseline upwards. The `\raise` of `-.17em` takes the slash back down by the part that lies below the baseline (`.15em`) plus half the stroke, so the ink lands where the path says.

Like the wave, the slash goes through `\__LuaCJK_fntef_use_drawn_type:nn`, so it is stroked in the color of `format` or of its text, and it is divided into whole cells, each written by the pass, for the reason given there.

```
\cs_new_protected:Npn \__LuaCJK_fntef_slash:
  {
    \tl_set:Nn \l__LuaCJK_fntef_any_code_tl
      {
        \tex_leaders:D \hbox:n
          {
            \hbox_set:Nn \l__LuaCJK_fntef_pattern_box
              {
                \box_move_up:nn { -.17em }
                  {
                    \hbox:n
                      {
                        \draw_begin:
                          \__LuaCJK_fntef_pattern_color:
                          \draw_set_linewidth:n { .04em }
```

```
                            \draw_path_moveto:n { 0em , -.15em }
                            \draw_path_lineto:n { .42em , .8em }
                            \draw_path_use_clear:n { stroke }
                          \draw_end:
                        }
                    }
                }
              \box_set_ht:Nn  \l__LuaCJK_fntef_pattern_box { \c_zero_dim }
              \box_set_dp:Nn  \l__LuaCJK_fntef_pattern_box { \c_zero_dim }
              \box_use_drop:N \l__LuaCJK_fntef_pattern_box
            }
        }
      \tl_set:Nn \l__LuaCJK_fntef_cache_key_tl { xout }
      \exp_args:NV \__LuaCJK_fntef_use_drawn_type:nn
        \l__LuaCJK_fntef_cache_key_tl \l__LuaCJK_fntef_any_code_tl
    }
\NewDocumentCommand \CJKxout { s t- o +m }
    {
      \group_begin:
        \__LuaCJK_fntef_decoration_setup:n {#3}
        \__LuaCJK_fntef_slash:
        \__LuaCJK_fntef_body:n {#4}
      \group_end:
    }
```

(*End of definition for* `\CJKxout`*. This function is documented on page 45.*)

`\LuaCJKemdescent` The depth of the em box of the current font, as a dimension. A CJK em box is one em square, so an object of height `1em` placed at this depth covers a character the way a PDF reader selects it.

```
\cs_new:Npn \LuaCJKemdescent
  {
    \lua_now:e
      {
        tex.print(LuaCJKfntef.em_descent
          ( \int_use:N \tex_attribute:D \c__LuaCJK_font_attr_int ))
      }
    sp
  }
```

(*End of definition for* `\LuaCJKemdescent`*. This function is documented on page 45.*)

`\CJKunderanyline` Underline with an arbitrary object: the object is put in a box, lowered by ⟨*depth*⟩, and repeated by `\leaders`. The underline type is cached by the object, the depth,

and the color, so repeating a command with the same arguments costs nothing. The leader code being built and its depth.

__LuaCJK_fntef_any_code_tl
\l__LuaCJK_fntef_any_depth_dim

```
\tl_new:N \l__LuaCJK_fntef_any_code_tl
\dim_new:N \l__LuaCJK_fntef_any_depth_dim

\NewDocumentCommand \CJKunderanyline { m m +m }
  {
    \group_begin:
      \__LuaCJK_fntef_decoration_setup:n { }
      \dim_set:Nn \l__LuaCJK_fntef_any_depth_dim { -#1 }
      \tl_set:Ne \l__LuaCJK_fntef_any_code_tl
        {
          \exp_not:N \tex_leaders:D \exp_not:N \hbox:n
            {
              \exp_not:N \box_move_up:nn
                { \dim_use:N \l__LuaCJK_fntef_any_depth_dim }
                { \exp_not:N \hbox:n { \exp_not:n {#2} } }
            }
        }
      \tl_set:Ne \l__LuaCJK_fntef_cache_key_tl
        { anyline | #1 | \tl_to_str:n {#2} }
      \exp_args:NV \__LuaCJK_fntef_use_leader_type:nn
        \l__LuaCJK_fntef_cache_key_tl \l__LuaCJK_fntef_any_code_tl
      \__LuaCJK_fntef_body:n {#3}
    \group_end:
  }
```

(*End of definition for* `\CJKunderanyline`*,* `\l__LuaCJK_fntef_any_code_tl`*, and* `\l__LuaCJK_fntef_any_depth_dim`*. This function is documented on page* 46.)

__LuaCJK_fntef_use_leader_type:nn

Build the underline type of a decoration that is made of `\leaders`, which repeats one rule or one symbol along the decorated text, and use it. A type is built once per leader code and color and then reused, so a document pays for each combination of geometry and color only once.

The type keeps the color that was in force when it was built: luacolor colors a rule by attribute, and lua-ul puts the attributes of the built box on every leader it inserts. A type cached by the geometry alone would therefore keep the color of the first call, so the color is part of the key: `format` resolved to the operations of its color, which is what lua-ul puts in the context of its own type, and otherwise the value of luacolor's attribute, which is the color of the surrounding text. A requested color is also applied inside the box, so that it holds wherever the leader is placed. The key holds the resolved operations rather than the name of the color, because a name may be defined again: `\colorlet{mark}{red}` followed

by `\colorlet{mark}{blue}` gives two keys, and the second decoration is drawn in blue instead of reusing the type that the first one built.

The key carries the settings entry of the call as well, because the pair the type builds has to name the entry of its own call: the attribute list a leader carries is the one captured when the type was defined, and a type shared by two calls with different settings would hand both of them the first one's inset, `skip`, `hidden`, and dots. Two calls with the same settings share one entry, so the type is still built once for them.

The name of the type already carries a counter, so the context that lua-ul puts in the name of the type it defines is a constant: one that expands again when the type is used would rebuild the type with whatever color happened to be current then.

A drawn pattern takes its color another way. Its ink comes from l3draw, which strokes with the color that l3color holds, so luacolor's attribute never reaches it; an attribute on a leader box that holds a drawing only leaks, because luacolor writes the color in front of the repeated box and finds no node after it to restore the previous one. The color operations are therefore read from `\current@color`, the resolved command that luacolor itself would write, and the drawing emits them inside its own graphics scope. A `format` is resolved in a group of its own, so the attribute it sets ends with that group and no text outside the pattern takes its color.

```
\cs_new_protected:Npn \__LuaCJK_fntef_use_leader_type:nn #1#2
  { \__LuaCJK_fntef_use_type:nnn {#1} {#2} { attribute } }
\cs_new_protected:Npn \__LuaCJK_fntef_use_drawn_type:nn #1#2
  { \__LuaCJK_fntef_use_type:nnn {#1} {#2} { drawn } }
\cs_new_protected:Npn \__LuaCJK_fntef_use_type:nnn #1#2#3
  {
    \tl_if_empty:NTF \l__LuaCJK_fntef_format_tl
      {
        \tl_set:Ne \l__LuaCJK_fntef_color_key_tl
          { ambient | \tex_the:D \use:c { LuaCol@Attribute } }
      }
      {
        \luaul_set_color:n { \l__LuaCJK_fntef_format_tl }
        \tl_set:Ne \l__LuaCJK_fntef_color_key_tl
          { format | \luaul_context_color: }
      }
    \str_if_eq:nnT {#3} { drawn } { \__LuaCJK_fntef_capture_color: }
    \tl_set:Ne \l__LuaCJK_fntef_cache_key_tl
      {
        \exp_not:n {#1}
        | \dim_to_decimal_in_sp:n { 1em }
```

```
      | \l__LuaCJK_fntef_color_key_tl
      | \l__LuaCJK_fntef_settings_key_tl
    }
  \prop_get:NVNF \g__LuaCJK_fntef_format_prop
    \l__LuaCJK_fntef_cache_key_tl \l__LuaCJK_fntef_cache_name_tl
    {
      \int_gincr:N \g__LuaCJK_fntef_format_int
      \tl_set:Ne \l__LuaCJK_fntef_cache_name_tl
        { LuaCJKfntef / ul / \int_use:N \g__LuaCJK_fntef_format_int }
      \str_if_eq:nnTF {#3} { drawn }
        {
          \exp_args:Nc \newunderlinetype
            { \l__LuaCJK_fntef_cache_name_tl } [ LuaCJKfntef ] {#2}
        }
        {
          \tl_if_empty:NTF \l__LuaCJK_fntef_format_tl
            {
              \exp_args:Nc \newunderlinetype
                { \l__LuaCJK_fntef_cache_name_tl } [ LuaCJKfntef ] {#2}
            }
            {
              \exp_args:Nc \newunderlinetype
                { \l__LuaCJK_fntef_cache_name_tl } [ LuaCJKfntef ]
                { \luaul_apply_color: #2 }
            }
        }
      \prop_gput:NVV \g__LuaCJK_fntef_format_prop
        \l__LuaCJK_fntef_cache_key_tl \l__LuaCJK_fntef_cache_name_tl
    }
  \use:c { \l__LuaCJK_fntef_cache_name_tl }
}
```

(*End of definition for* `\__LuaCJK_fntef_use_leader_type:nn`.)

`\__LuaCJK_fntef_capture_color:`
`\__LuaCJK_fntef_pattern_color:`
`\__LuaCJK_fntef_literal_direct:n`

Read the color operations of the pattern. Without `format` they are the ones of the text the pattern decorates, which `\current@color` already holds; with it, the color is resolved the way luacolor resolves it and read from the same place, in a group that keeps the attribute to itself. The drawing then writes them, expanded, so that the literal keeps the color of the call that built it: a literal whose payload still held the variable would expand it at shipout, with whatever color is current then.

```
\cs_new_protected:Npn \__LuaCJK_fntef_capture_color:
  {
    \tl_if_empty:NTF \l__LuaCJK_fntef_format_tl
```

```
      { \tl_set_eq:Nc \l__LuaCJK_fntef_color_ops_tl { current@color } }
      {
        \group_begin:
          \luaul_apply_color:
          \tl_gset_eq:Nc \g__LuaCJK_fntef_color_ops_tl { current@color }
        \group_end:
        \tl_set_eq:NN \l__LuaCJK_fntef_color_ops_tl \g__LuaCJK_fntef_color_ops_tl
      }
  }
\cs_new_protected:Npn \__LuaCJK_fntef_pattern_color:
  {
    \tl_if_empty:NF \l__LuaCJK_fntef_color_ops_tl
      {
        \exp_args:NV \__LuaCJK_fntef_literal_direct:n
          \l__LuaCJK_fntef_color_ops_tl
      }
  }
\cs_new_protected:Npn \__LuaCJK_fntef_literal_direct:n #1
  {
    \tex_expanded:D
      { \exp_not:N \tex_pdfextension:D literal direct {#1} }
  }
```

(*End of definition for* `\__LuaCJK_fntef_capture_color:`, `\__LuaCJK_fntef_pattern_color:`, *and* `\__LuaCJK_fntef_literal_direct:n`.)

`\CJKunderanysymbol` One symbol per em, which is what one wants for a repeated mark: the cell is as wide as a CJK character, so a row of symbols lines up with the characters above, and the symbol is centered in its cell. The append callback writes one copy of the symbol per cell, from the left edge of the span, because `\leaders` aligns its copies at the edge of the enclosing box and draws none that straddles the edge of the glue. The depth is handed to `\LuaCJKfntef_use_symbol:n`, which puts the top of the symbol that far below the baseline.

```
\NewDocumentCommand \CJKunderanysymbol { O{} m m +m }
  {
    \group_begin:
      \__LuaCJK_fntef_decoration_setup:n {#1}
      \tl_set:Nn \l__LuaCJK_fntef_symbol_tl {#3}
      \__LuaCJK_fntef_use_symbol:n {#2}
      \__LuaCJK_fntef_body:n {#4}
    \group_end:
  }
```

(*End of definition for* `\CJKunderanysymbol`. *This function is documented on page 46.*)

\CJKunderdot

Emphasise with dots below the glyphs. The optional argument takes the key set of `\CJKunderline`, like the other decorations. A symbol given with `symbol` is a pattern repeated along the span, as `\CJKunderanysymbol` repeats its own; the default dot is a setting of the decoration pass, which puts one under each CJK character, so the ink travels with the character rather than with the beginning of the span.

```
\NewDocumentCommand \CJKunderdot { s t- o +m }
  {
    \group_begin:
      \__LuaCJK_fntef_dots_setup:n {#3}
      \tl_if_empty:NTF \l__LuaCJK_fntef_symbol_tl
        { \__LuaCJK_fntef_underdot:n { \__LuaCJK_fntef_body:n {#4} } }
        {
          \__LuaCJK_fntef_use_symbol:n { .3em }
          \__LuaCJK_fntef_body:n {#4}
        }
    \group_end:
  }
```

__LuaCJK_fntef_underdot:n

Apply the dots inside a group. The underline type of the dots carries no ink: it gives the decoration the pair of nodes that lua-ul provides, and with them the width, the stretch, and the shrink of the text. The pass then writes the dots into the text itself, which is also where `subtract` and `skip` act: a character whose centre lies in the part that `subtract` takes off gets no dot, and `skip` leaves the punctuation undotted. The underline type of the dots: a leader box of nothing, which leaves the pair without a pattern to repeat. It is cached like every other type, so its leaders glue names the settings entry of the call that built it.

__LuaCJK_fntef_dots:

```
\cs_new_protected:Npn \__LuaCJK_fntef_dots:
  {
    \tl_set:Nn \l__LuaCJK_fntef_any_code_tl { \tex_leaders:D \hbox:n { } }
    \tl_set:Nn \l__LuaCJK_fntef_cache_key_tl { dots }
    \exp_args:NV \__LuaCJK_fntef_use_leader_type:nn
      \l__LuaCJK_fntef_cache_key_tl \l__LuaCJK_fntef_any_code_tl
  }
\cs_new_protected:Npn \__LuaCJK_fntef_underdot:n #1
  {
    \group_begin:
      \__LuaCJK_fntef_dots:
      #1
    \group_end:
  }
```

(*End of definition for* `\CJKunderdot`, `\__LuaCJK_fntef_underdot:n`, *and* `\__LuaCJK_fntef_dots:`. *This function is documented on page* 46.)

`CJKfilltwosides` (*env.*) Set the body in a `minipage` of the given width, with the lines separated by `\\`. The inter-character glue is given an infinite stretch inside the box, so a line of CJK characters reaches the width even when there is no Latin word space to stretch. A `minipage` is a `\parbox`, so the body takes the settings of one and the box keeps the settings of the kernel: the paragraph indent and the paragraph skip are zero, so every paragraph after a paragraph break starts at the left edge of the box and is filled like the first; the leading is the document's own, which an enclosing alignment would otherwise replace; `\\` is the break of running text even in a table cell, so the break belongs to the box and not to the alignment; and the box leaves vertical mode, so the text after the environment continues on its line. A space written after the environment is ignored, as xeCJK ignores it. A `\footnote` in the body is set at the bottom of the box, below a rule, and numbered with the box's own counter, as it is in a `minipage`. Under tagging, latex-lab's minipage module ends the paragraph structure around the box, opens a `text-minipage` for the body, and reopens the paragraph after it, so the body is a structure of its own rather than a marked-content sequence of the surrounding paragraph. The optional argument places the box on the baseline like the same argument of `\parbox`: `t` for the first line, `c` for the text axis (the default), and `b` for the last line. A width of zero or less keeps the surrounding line width, which reaches the box as `\linewidth`.

```
\cs_new_protected:Npn \__LuaCJK_fntef_fill_begin:nn #1#2
  {
    \group_begin:
    \LuaCJKsetup { CJKglue = 0pt plus 1filll }
    \dim_compare:nNnTF {#2} > { 0pt }
      { \__LuaCJK_fntef_fill_box:nn {#1} {#2} }
      { \__LuaCJK_fntef_fill_box:nn {#1} { \linewidth } }
  }
\cs_new_protected:Npn \__LuaCJK_fntef_fill_box:nn #1#2
  {
    \use:e
      { \exp_not:N \minipage [#1] { \dim_eval:n {#2} } }
  }
\cs_new_protected:Npn \__LuaCJK_fntef_fill_end:
  {
    \endminipage
    \group_end:
    \ignorespacesafterend
  }
\NewDocumentEnvironment { CJKfilltwosides } { O{c} m }
  { \__LuaCJK_fntef_fill_begin:nn {#1} {#2} }
  { \__LuaCJK_fntef_fill_end:               }
```

`CJKfilltwosides*` (*env.*) The starred form behaves like the plain one; a width of zero or less keeps the surrounding line width in both.

```
\NewDocumentEnvironment { CJKfilltwosides* } { O{c} m }
  { \__LuaCJK_fntef_fill_begin:nn {#1} {#2} }
  { \__LuaCJK_fntef_fill_end:               }
⟨/fntef⟩
```

A.10 Listings and ctex

The block listing environments are patched by making the CJK characters active while the character table of a listing is built. The active character is defined through the `\lccode` of two helper characters: the token in the body keeps category code 12, so it can be stored in the token list of listings and typeset later. The definitions are made once; each listing only sets the category codes again in its own group. A character outside the table is typeset the moment listings reads it, which puts it next to the wrong text and counts it as one column.

A run of CJK characters is one token. It is appended to the token list character by character, and the run is output only where listings ends a letter run: at a Latin letter, a digit, another character, or a space. The `emph` and `keywords` options match an entry against that token, so a CJK word is highlighted the way xeCJK highlights it. A CJK mark is a token of its own, because the word in front of it would otherwise enter the token with it; the marks are the ones LuaCJK.punct_codes lists inside the character table.

The two appending macros of listings are wrapped to end the run, and the saved macro has to be the last token of the wrapper: `\lst@Append` takes the character that follows the active character in the input as its argument, so a brace or the `\fi` of a conditional behind it would be taken instead.

With `breaklines` on, the run is set one character at a time, so that listings has a break point after each of them: its break is the discretionary it puts behind every chunk it outputs, and a chunk of the whole run carries none inside it. `\lst@Output` is the macro that decides, because the end of a line flushes the token through `\lst@PrintToken` and never reaches the wrappers above. The style is resolved once, on the whole token, and the pieces are set under it with the word detection switched off, so `emph` and `keywords` still name the whole word. The token is a local assignment, so it is reset again after the group that holds the pieces, and `breakatwhitespace` does not split the run at all, since its break would be suppressed inside it.

The marks arrive from Lua as a comma-separated list of code points, printed with the catcode table of other characters: by the time the character table is built,

listings has made every printable ASCII character active, so a digit printed with the current table would not be a number.

The ranges are those of `\LuaCJK.is_cjk`, limited to the Basic Multilingual Plane: each definition costs an entry of the string pool, and the supplementary planes would triple it for ideographs a listing almost never holds. A range includes its first and its last code point, since a mark that opens a range would otherwise return to the character handling above.

```
⟨*listings⟩
\RequirePackage { LuaCJK, listings }
\makeatletter
\newif\ifLuaCJK@listings@cjkrun
\newif\ifLuaCJK@listings@defined
\newif\ifLuaCJK@listings@breakrun
\lst@AddToHook{InitVars}{\LuaCJK@listings@cjkrunfalse}
\def\LuaCJK@listings@process#1{%
  \lst@whitespacefalse
  \LuaCJK@listings@setcjkflag
  \advance\lst@length\@ne
  \lst@Append#1}
\def\LuaCJK@listings@processpunct#1{%
  \lst@whitespacefalse
  \LuaCJK@listings@cjkrunfalse
  \LuaCJK@listings@setcjkflag
  \advance\lst@length\@ne
  \lst@Append#1%
  \lst@Output
  \lst@letterfalse
  \LuaCJK@listings@cjkrunfalse}
\def\LuaCJK@listings@setcjkflag{%
  \ifLuaCJK@listings@cjkrun
    \lst@lettertrue
  \else
    \lst@ifletter \lst@Output \else \lst@OutputOther \lst@lettertrue \fi
    \LuaCJK@listings@cjkruntrue
  \fi}
\let\LuaCJK@listings@letter\lst@AppendLetter
\let\LuaCJK@listings@other\lst@AppendOther
\def\lst@AppendLetter{%
  \ifLuaCJK@listings@cjkrun
    \lst@Output \lst@lettertrue \LuaCJK@listings@cjkrunfalse
  \fi
  \LuaCJK@listings@letter}
\def\lst@AppendOther{%
  \ifLuaCJK@listings@cjkrun
```

```
    \lst@Output \lst@letterfalse \LuaCJK@listings@cjkrunfalse
  \fi
  \LuaCJK@listings@other}
\let\LuaCJK@listings@output\lst@Output
\def\lst@Output{%
  \LuaCJK@listings@breakrunfalse
  \ifLuaCJK@listings@cjkrun
    \ifnum\lst@length>\tw@
      \lst@ifbreaklines
        \lst@ifbreakatwhitespace\else
          \LuaCJK@listings@breakruntrue
        \fi
      \fi
    \fi
  \fi
  \ifLuaCJK@listings@breakrun
    \LuaCJK@listings@splitrun
  \else
    \LuaCJK@listings@output
  \fi}
\def\LuaCJK@listings@splitrun{%
  \lst@CheckDelay
  \ifnum\lst@length=\z@\else
    \ifx\lst@currstyle\relax
      \let\lst@thestyle\lst@identifierstyle
    \else
      \let\lst@thestyle\lst@currstyle
    \fi
    \lsthk@Output
    \begingroup
      \let\lst@currstyle\lst@thestyle
      \let\lst@ifkeywords\iffalse
      \edef\LuaCJK@listings@run{\the\lst@token}%
      \expandafter\LuaCJK@listings@emit\LuaCJK@listings@run\@empty\@nil
    \endgroup
    \lst@ResetToken
  \fi
  \let\lst@lastother\relax}
\def\LuaCJK@listings@emit#1#2\@nil{%
  \lst@token{#1}%
  \lst@length\tw@
  \lst@OutputToken
  \ifx\@empty#2\@empty\else
    \expandafter\LuaCJK@listings@emit#2\@nil
  \fi}
```

```
\def\LuaCJK@listings@define#1#2{%
  \@tempcnta=#1\relax
  \loop
    \lccode`\~=\@tempcnta
    \lccode`\/=\@tempcnta
    \lowercase{\gdef~{\LuaCJK@listings@process/}}%
    \ifnum\@tempcnta<#2\relax
    \advance\@tempcnta\@ne
  \repeat}
\def\LuaCJK@listings@definepunct#1{%
  \lccode`\~=#1\relax
  \lccode`\/=#1\relax
  \lowercase{\gdef~{\LuaCJK@listings@processpunct/}}}
\def\LuaCJK@listings@catcodes#1#2{%
  \@tempcnta=#1\relax
  \loop
    \catcode\@tempcnta=13\relax
    \ifnum\@tempcnta<#2\relax
    \advance\@tempcnta\@ne
  \repeat}
\def\LuaCJK@listings@ranges#1{%
  #1{"00B7}{"00B7}%
  #1{"1100}{"11FF}%
  #1{"2014}{"2014}%
  #1{"2015}{"2015}%
  #1{"2E3A}{"2E3B}%
  #1{"2018}{"2019}%
  #1{"201C}{"201D}%
  #1{"2025}{"2025}%
  #1{"2026}{"2026}%
  #1{"2027}{"2027}%
  #1{"22EF}{"22EF}%
  #1{"2E80}{"9FFF}%
  #1{"A960}{"A97C}%
  #1{"AC00}{"D7C6}%
  #1{"D7CB}{"D7FB}%
  #1{"F900}{"FAFF}%
  #1{"FE30}{"FE4F}%
  #1{"FF00}{"FFEF}}
\def\LuaCJK@listings@punctuate#1,{%
  \ifx\relax#1\relax\else
    \LuaCJK@listings@definepunct{#1}%
    \expandafter\LuaCJK@listings@punctuate
  \fi}
\def\LuaCJK@listings@punctuation{%
```

```
  \edef\LuaCJK@listings@codes{%
    \directlua{tex.sprint(-2,table.concat(LuaCJK.punct_codes(),","))}}%
  \expandafter\LuaCJK@listings@punctuate\LuaCJK@listings@codes,\relax,}
\def\LuaCJK@listings@activate{%
  \ifLuaCJK@listings@defined\else
    \LuaCJK@listings@ranges\LuaCJK@listings@define
    \global\LuaCJK@listings@definedtrue
  \fi
  \LuaCJK@listings@ranges\LuaCJK@listings@catcodes
  \LuaCJK@listings@punctuation}
```

`\lstinline` reads its argument with changed category codes, so the patch is limited to the block listing environments.

```
\lst@AddToHook{SelectCharTable}{%
  \ifx\lst@inlinechars\@empty
    \LuaCJK@listings@activate
  \fi}
\lst@AddToHook{OutputBox}{%
  \ifx\lst@inlinechars\@empty
    \LuaCJKdisable
  \fi}
\makeatother
```
⟨/listings⟩

ctex inputs an engine file named `ctex-engine-`⟨*name*⟩`.def`, where ⟨*name*⟩ is the detected engine unless the `engine` option names another. LuaCJK's file is therefore called `ctex-engine-luacjk.def` and is selected with `engine = luacjk`. The middle part must differ from `luatex`: ctex installs `ctex-engine-luatex.def`, a second file of that name would collide on CTAN, and the two files would describe different backends under one name.

The file defines the handful of `\ctex_`⟨*name*⟩ commands that the ctex kernel calls, and maps the parts of the luatexja interface that ctex and some documents still use.

⟨∗ctex⟩
```
\typeout{LuaCJK~ctex~engine~loaded}
\sys_if_engine_luatex:F
  {
    \msg_new:nnn { LuaCJK } { ctex-engine }
      { The ~ ctex ~ engine ~ file ~ of ~ LuaCJK ~ requires ~ LuaTeX. }
    \msg_critical:nn { LuaCJK } { ctex-engine }
    \tex_endinput:D
  }
```

CJKfntef loads the legacy CJK package, which redefines the font family interface LuaCJK owns, so this file refuses it in both orders: the run stops when CJKfntef is already loaded, and the `package/CJKfntef/before` hook stops it when it is requested later. The message points at LuaCJKfntef, which does the same work through lua-ul. ctex's own LuaTeX backend refuses the package for the same reason, and that code does not run when the engine file is ours.

```
\msg_new:nnnn { LuaCJK } { cjkfntef }
  {
    Package ~ `CJKfntef' ~ is ~ not ~ compatible ~ with ~ the ~ LuaCJK ~
    backend ~ of ~ ctex.
  }
  { Use ~ the ~ `LuaCJKfntef' ~ package ~ instead. }
\cs_if_exist:cT { ver@CJKfntef.sty }
  { \msg_critical:nn { LuaCJK } { cjkfntef } }
\hook_gput_code:nnn { package / CJKfntef / before } { LuaCJK }
  { \msg_critical:nn { LuaCJK } { cjkfntef } }
```

The file loads the package it serves.

```
\RequirePackage { LuaCJK }
```

The fontset of ctex defines these shortcuts itself and checks for an existing definition, so the ones of LuaCJK have to go first.

```
\clist_map_inline:nn
  { \songti, \heiti, \fangsong, \kaishu, \lishu, \youyuan, \yahei, \pingfang }
  { \cs_undefine:N #1 }
⟨/ctex⟩
```

The interface that the ctex kernel and its classes rely on.

```
⟨*ctex⟩
\cs_new_protected:Npn \ctex_add_to_selectfont:n #1
  { \hook_gput_code:nnn { selectfont } { LuaCJK } {#1} }
\cs_new_eq:NN \ctex_ltj_zero_globaldefs: \prg_do_nothing:
\cs_new_protected:Npn \CTEX@alchar #1 {#1}
\cs_new_eq:NN \CTEX@beginallalchar \prg_do_nothing:
\cs_new_eq:NN \CTEX@endallalchar \prg_do_nothing:
\clist_map_inline:nn { \ccwd, \cht, \cdp, \cwd } { \dim_new:N #1 }
\cs_new_protected:Npn \ctex_update_ccwd:
  { \dim_set:Nn \ccwd { \f@size \p@ } }
\cs_new_protected:Npn \ctex_update_em_unit:
  { \dim_set:Nn \ccwd { \f@size \p@ } }
\cs_new_protected:Npn \ctex_update_kanjisize:
  {
    \dim_set:Nn \cwd { \ccwd }
    \dim_set:Nn \cht { \f@size \p@ }
```

```
    \dim_set:Nn \cdp { \c_zero_dim }
  }
⟨/ctex⟩
```

ctex keeps its own copy of the CJK glue so that it can tell whether the document changed it. The two copies are kept in step here.

```
⟨*ctex⟩
\skip_new:N \l__ctex_ccglue_skip
\skip_set_eq:NN \l__ctex_ccglue_skip \l__LuaCJK_ccglue_skip
\cs_new_protected:Npn \ctex_ltj_set_kanjiskip:N #1
  {
    \skip_set_eq:NN \l__LuaCJK_ccglue_skip #1
    \bool_set_true:N \l__LuaCJK_ccglue_touched_bool
    \__LuaCJK_settings_push:
  }
\cs_new_protected:Npn \ctex_ltj_set_xkanjiskip:N #1
  {
    \skip_set_eq:NN \l__LuaCJK_ecglue_skip #1
    \bool_set_true:N \l__LuaCJK_ecglue_touched_bool
    \__LuaCJK_settings_push:
  }
\cs_new_protected:Npn \ctex_update_ccglue:
  {
    \skip_set_eq:NN \l__LuaCJK_ccglue_skip \l__ctex_ccglue_skip
    \bool_set_true:N \l__LuaCJK_ccglue_touched_bool
    \__LuaCJK_settings_push:
  }
\prg_new_conditional:Npnn \ctex_if_ccglue_touched: { TF }
  {
    \skip_if_eq:nnTF
      { \l__ctex_ccglue_skip } { \l__LuaCJK_ccglue_skip }
      { \prg_return_false:   } { \prg_return_true: }
  }
\ctex_at_end:n
  {
    \cs_set_protected:Npn \__ctex_update_stretch_auxii:
      {
        \ctex_if_ccglue_touched:TF
          { \ctex_update_ccwd: } { \__ctex_update_stretch_auxiii: }
      }
  }
⟨/ctex⟩
```

The gap between CJK and Latin text has its own default under ctex, `.25em`,

and follows the size in the same way.

```
⟨*ctex⟩
\skip_set:Nn \l__LuaCJK_ecglue_skip { .25em plus 1pt minus 1pt }
\skip_new:N \l__ctex_xkanjiskip_skip
\skip_set_eq:NN \l__ctex_xkanjiskip_skip \l__LuaCJK_ecglue_skip
\tl_new:N \l__ctex_xkanjiskip_tl
\tl_set:Nn \l__ctex_xkanjiskip_tl { .25em plus 1pt minus 1pt }
\cs_new_protected:Npn \ctex_update_xkanjiskip:
  {
    \skip_if_eq:nnT
      { \l__LuaCJK_ecglue_skip } { \l__ctex_xkanjiskip_skip }
      {
        \skip_set:Nn \l__ctex_xkanjiskip_skip { \l__ctex_xkanjiskip_tl }
        \skip_set_eq:NN
          \l__LuaCJK_ecglue_skip \l__ctex_xkanjiskip_skip
        \bool_set_true:N \l__LuaCJK_ecglue_touched_bool
        \__LuaCJK_settings_push:
      }
  }
\clist_map_inline:nn { \ctex_update_xkanjiskip: , \ctex_update_kanjisize: }
  { \ctex_add_to_selectfont:n {#1} }
⟨/ctex⟩
```

The ctex option `punct` selects a punctuation style, and `space` has no counterpart here.

```
⟨*ctex⟩
\cs_new_protected:Npn \ctex_set_jfm:n #1
  {
    \tl_set:Nn \l__ctex_jfm_tl {#1}
    \LuaCJKsetup { PunctStyle = {#1} }
  }
\cs_generate_variant:Nn \ctex_set_jfm:n { o }
\tl_new:N \l__ctex_jfm_tl
\ctex_define:n
  {
    space .code:n    =
      { \msg_warning:nn { ctex } { invalid-option } }
  }
\ctex_define:n
  {
    punct .code:n    =
      {
        \tl_set:Ne \l__ctex_punct_tl {#1}
        \ctex_set_jfm:o { \l__ctex_punct_tl }
      },
```

```
    punct .default:n = { quanjiao },
  }
\ctex_define:n
  { experiment .meta:nn = { ctex / experiment } {#1} }
\keys_define:nn { ctex / experiment }
  {
    CJKecglue .code:n           =
      {
        \tl_set:Nn \l__ctex_xkanjiskip_tl {#1}
        \skip_set:Nn \l__ctex_xkanjiskip_skip {#1}
        \ctex_ltj_set_xkanjiskip:N \l__ctex_xkanjiskip_skip
      },
    CJKecglue .value_required:n = true
  }
\ctex_set_jfm:o { \l__ctex_punct_tl }
⟨/ctex⟩
```

A very small part of the luatexja parameter interface is kept, for documents that set the skips directly. Only `kanjiskip` and `xkanjiskip` are understood; any other key is silently ignored, because this is not a reimplementation of the luatexja parameter system.

```
⟨*ctex⟩
\keys_define:nn { LuaCJK / ctex / ltj }
  {
    kanjiskip  .code:n = { \__LuaCJK_set_ccglue:n {#1} },
    xkanjiskip .code:n = { \__LuaCJK_set_ecglue:n {#1} },
    unknown    .code:n = { },
  }
\NewDocumentCommand \ltjsetparameter { m }
  { \keys_set:nn { LuaCJK / ctex / ltj } {#1} }
\cs_new:Npn \ltjgetparameter #1
  {
    \str_case:nnF {#1}
      {
        { kanjiskip  } { \skip_use:N \l__LuaCJK_ccglue_skip }
        { xkanjiskip } { \skip_use:N \l__LuaCJK_ecglue_skip }
      } { 0pt }
  }
⟨/ctex⟩
```

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

Symbols

\/ 6260, 6267
\\ 15, 103, 161, 744, 1127, 1132, 1225, 3765
\␣ 89
\~ 6259, 6266

A

\addCJKfontfeatures *10*, *31*, *76*, *267*, 1412
adjacent commands:
adjacent_marks 3350
ADJUST 1729
adjust 5232
\advance 6184, 6190, 6263, 6274
\allocationnumber 33, 5459
any commands:
any_cjk_math_enabled 3611
apply commands:
apply_check_single 4461
\apply_fixed_widths *179*, *180*
apply_fixed_widths 3484
apply_kinsoku 3319
apply_shift 2406
\AtBeginDocument *85*
attached commands:
attached_box 2582
attribute commands:
attribute_shift 2406

B

\baselineskip *11*, *33*, *91*, 1281, 1295, 1300
\begin 3520, 3531, 3546, 3550, 5196
\begingroup 6240
\bfdefault *10*, *32*, *87*, 1287, 1289
\bfseries *87*
block commands:
block_font 2317
\boldmath *10*, *32*, *87*, *270*
bool commands:
\bool_gset_false:N 657
\bool_gset_true:N 655, 1219, 1665, 1672
\bool_if:NTF 61, 66, 148, 150, 152, 168, 249, 407, 409, 682, 685, 703, 705, 805, 847, 1021, 1162, 1230, 1317, 1351, 1394, 1591, 1673, 1705, 5572, 5583, 5596, 5603, 5605, 5884, 5892, 5950
\bool_lazy_all:nTF 5552
\bool_lazy_and:nnTF 1283, 5529
\bool_lazy_or:nnTF 950, 960
\bool_new:N 44, 45, 115, 116, 117, 156, 228, 387, 389, 457, 458, 461, 464, 477, 478, 736, 737, 738, 909, 910, 911, 912, 992, 1151, 1215, 1666, 5463, 5480, 5481, 5482, 5483, 5499
\bool_not_p:n 5556
\bool_set_eq:NN . . 791, 793, 811, 813
\bool_set_false:N 140, 141, 142, 243, 568, 589, 612, 651, 820, 942, 1043, 1212, 1355, 5810, 5811, 5812, 5813
\bool_set_true:N 52, 57, 124, 129, 134, 170, 247, 398, 400, 572, 593, 609, 647, 800, 867, 878, 932, 933, 934, 935, 1040, 1164, 1353, 5464, 5787, 5793, 5809, 5831, 6370, 6376, 6382, 6414
bound commands:
bound_width 3799
boundary commands:
\boundary_settings *137*
boundary_settings 1858
\boundary_spec *142*, *146*
boundary_spec 2933

box commands:
\box_ht:N 5927
\box_move_up:nn 5855, 5937, 5973, 6024
\box_new:N 740, 5842
\box_set_dp:Nn 5873, 5932, 5988
\box_set_ht:Nn 5872, 5930, 5987
\box_use_drop:N . . . 5874, 5939, 5989
box_visible 2573
\box_wd:N 763, 5918

C

\catcode 6272
\ccwd 6352, 6354, 6356, 6359
\cdp 6352, 6361
cff commands:
cff_box 4034
cff_dict 3963
cff_index 3944
cff_load 4010
cff_string 3959
\changes 3521
char commands:
\char_generate:nn 77
\char_set_catcode_ignore:n 1590, 1594, 1596, 1598
\char_set_catcode_space:n 88
\cht 6352, 6360
\CJKfamily *4*, *6*, *25*, *27*, *83*, *95*, *267*, *271*, 1522, 1554, 1555, 1556, 1557, 1558, 1559
\CJKfamilydefault *7*, *9*, *28*, *30*, *81*, *82*, *91*, *101*, *270*, *273*, 1100, 1106, 1120, 1400, 1690, 1693
CJKfilltwosides (env.) 1, 1, 6148
CJKfilltwosides* (env.) 6170
\CJKfontspec *6*, *28*, *75*, *267*, *270*, *274*, *276*, 1543
\CJKrmdefault *6–9*, *27–30*, *81*, *82*, *91*, *93*, *101*, *270*, *271*, *273*, 1112, 1400, 1452, 1697, 1701
\CJKsfdefault *7*, *13*, *28*, *36*, *95*, *279*, 1400, 1457, 1698
\CJKsout *23*, *46*, 5641
\CJKttdefault . . . *7*, *28*, 1400, 1462, 1699
\CJKunderanyline . *22*, *46*, *267*, *276*, 6013
\CJKunderanysymbol *22*, *23*, *46*, *236*, *267*, *276*, 6113
\CJKunderdblline *21*, *45*, *267*, 5834
\CJKunderdot *23*, *46*, *191*, *193*, *196*, *218*, *267*, *276*, *278*, 6122
\CJKunderline *20*, *43*, *44*, *236*, *276*, *277*, 5472, 5633
\CJKunderwave *21*, *45*, *267*, *276*, 5842
\CJKxout *22*, *45*, *267*, 5965
\clap *129*, *271*
class commands:
class_of_char 2560
class_of_node 2567
clist commands:
\clist_const:Nn 1583, 1585
\clist_gput_right:Nn 196
\clist_if_in:NnTF 195, 210
\clist_map_inline:Nn 244, 1367, 1589, 1593
\clist_map_inline:nn 22, 206, 318, 886, 940, 1012, 1328, 1400, 1560, 5819, 6341, 6352, 6418
\clist_new:N 190, 203
\clist_put_right:Nn 211, 1516
\clist_remove_all:Nn 213
collapse commands:
collapse_earlier_spaces 3085
collect 5158
\colorlet *274*
copy commands:
copy_attributes 4896
copy_glyph 2512
cs commands:
\cs_generate_variant:Nn 489, 490, 918, 1491, 6427
\cs_gset_eq:NN 89, 90, 863
\cs_gset_protected:Npn 864
\cs_gset_protected:Npx . . . 834, 855
\cs_if_exist:NTF 18, 832, 853, 861, 1268, 1718, 6336
\cs_if_exist_p:N 1286
\cs_new:Npn . . 12, 679, 681, 6004, 6466
\cs_new_eq:NN . . 72, 6348, 6350, 6351

\cs_new_protected:Npn 49, 54, 59, 73, 80, 106, 204, 230, 236, 255, 257, 263, 275, 294, 302, 315, 340, 346, 367, 373, 412, 683, 747, 823, 840, 845, 851, 876, 881, 897, 919, 926, 928, 938, 969, 993, 1008, 1037, 1045, 1061, 1080, 1098, 1104, 1135, 1160, 1228, 1253, 1265, 1275, 1305, 1315, 1349, 1358, 1363, 1417, 1481, 1492, 1497, 1554, 1555, 1556, 1557, 1558, 1559, 1570, 1587, 1601, 1603, 1610, 1618, 1667, 5508, 5510, 5516, 5525, 5527, 5600, 5616, 5641, 5709, 5805, 5823, 5828, 5846, 5882, 5890, 5895, 5965, 6036, 6038, 6040, 6088, 6100, 6108, 6134, 6141, 6148, 6156, 6161, 6346, 6349, 6353, 6355, 6357, 6367, 6373, 6379, 6406, 6422
\cs_set_protected:Npn 6393
\cs_set_protected:Npx 1532
\cs_to_str:N 1531, 1662, 1663
\cs_undefine:N 6343

ctex commands:

\ctex_ *242*
\ctex_add_to_selectfont:n 6346, 6419
\ctex_at_end:n 6391
\ctex_define:n 6429, 6434, 6443
\ctex_if_ccglue_touched: . . . 6385
\ctex_if_ccglue_touched:TF . . 6395
\ctex_ltj_set_kanjiskip:N . . . 6367
\ctex_ltj_set_xkanjiskip:N 6373, 6451
\ctex_ltj_zero_globaldefs: . . 6348
\ctex_set_jfm:n 6422, 6427, 6439, 6455
\ctex_update_ccglue: 6379
\ctex_update_ccwd: 6353, 6396
\ctex_update_em_unit: 6355
\ctex_update_kanjisize: 6357, 6418
\ctex_update_xkanjiskip: 6406, 6418

ctex internal commands:

\l__ctex_ccglue_skip 6365, 6366, 6381, 6388
\l__ctex_jfm_tl 6424, 6428
\l__ctex_punct_tl . 6438, 6439, 6455
__ctex_update_stretch_auxii: 6393
__ctex_update_stretch_-auxiii: 6396
\l__ctex_xkanjiskip_skip 6402, 6403, 6409, 6411, 6413, 6450, 6451
\l__ctex_xkanjiskip_tl 6404, 6405, 6411, 6449
\cwd 6352, 6359

D

\def 6181, 6186, 6195, 6204, 6209, 6215, 6231, 6249, 6256, 6265, 6269, 6276, 6295, 6300, 6304
\defaultCJKfontfeatures *10*, *31*, *267*, *274*, 1410

dim commands:

\dim_compare:nNnTF . . 751, 768, 6152
\dim_compare_p:nNn 5554
\dim_eval:n 777, 1057, 1294, 1299, 5665, 5670, 5684, 5689, 5720, 5721, 5736, 5742, 5915, 5923, 6159
\dim_gset:Nn 762
\dim_new:N 459, 462, 731, 732, 733, 5472, 5473, 5476, 5484, 5485, 5494, 5495, 5496, 5497, 5501, 5502, 5843, 5844, 5845, 6014, 6352
\dim_set:Nn 460, 463, 573, 594, 749, 766, 5504, 5505, 5506, 5507, 5534, 5535, 5604, 5607, 5611, 5613, 5757, 5761, 5786, 5792, 5815, 5816, 5817, 5818, 5821, 5897, 5898, 5913, 5921, 6019, 6354, 6356, 6359, 6360, 6361
\dim_set_eq:NN 755
\dim_to_decimal_in_sp:n 177, 186, 682, 1056, 5577, 5580, 5581, 5589, 5593, 5594, 5680, 6056
\dim_use:N 776, 5621, 5622, 5627, 5628,

5646, 5647, 5714, 5715, 5729, 5730, 5731, 5925, 5948, 5949, 6025
\dim_zero:N 569, 590, 5549, 5550, 5760, 5807
\c_zero_dim 5872, 5873, 5931, 5933, 5987, 5988, 6361
\directlua 6302
disabled 2284
DISC 1729
dot commands:
dot_box 4999
draw commands:
\draw_begin: 5859, 5977
\draw_end: 5868, 5983
\draw_path_curveto:nnn . 5863, 5865
\draw_path_lineto:n 5981
\draw_path_moveto:n 5862, 5980
\draw_path_use_clear:n . 5867, 5982
\draw_set_linewidth:n . . 5861, 5979
drawn 5295
drop commands:
drop_orphans 2477

E

\edef 6243, 6301
edge commands:
edge_inked 2606
edge_mark 3382
\edge_punct *140*
edge_punct 2916
\else 6198, 6199, 6220, 6228, 6233, 6236, 6253, 6296, 6305
\emph *272*
\endgroup 6245
\endminipage 6163
environments:
CJKfilltwosides 1, 1, 6148
CJKfilltwosides* 6170
exp commands:
\exp_args:Nc . 32, 942, 1360, 1402, 1403, 5661, 5678, 6068, 6074, 6078
\exp_args:NNV 1236
\exp_args:NV 300, 759, 1004, 1102, 1121, 1144, 1145, 1168, 1196, 1328, 1375, 1414, 1451, 1456, 1461, 1467, 1503, 1525, 1534, 1549, 1551, 1607, 5630, 5724, 5749, 5879, 5953, 5993, 6031, 6104, 6138
\exp_args:Nv 1563
\exp_not:N 837, 857, 1533, 1697, 1698, 1699, 1701, 5620, 5719, 5735, 5909, 5911, 5913, 5915, 5918, 5921, 5923, 5926, 5927, 5930, 5931, 5932, 5933, 5934, 5935, 5936, 5937, 5938, 5939, 5940, 5941, 5942, 6022, 6024, 6026, 6111, 6159
\exp_not:n 310, 313, 836, 858, 955, 965, 1423, 1424, 1533, 5912, 6026, 6055
\expandafter 1359, 6244, 6254, 6298, 6303
\ExplFileVersion 11
explicit commands:
explicit_space 3080

F

fallback commands:
fallback_font 2362
\familydefault *7*, *28*, *273*, 1695
\fangsong *7*, *28*, *270*, 1554, 6342
\fbox *13*, *17*, *36*, *40*
\fcolorbox *17*, *40*
\fi 6199, 6201, 6207, 6212, 6222, 6223, 6224, 6225, 6230, 6238, 6247, 6255, 6299, 6308, 6314, 6318
file commands:
\file_if_exist:nTF 1612
\file_input:n 1613
find commands:
find_cjk_math 3732
first commands:
first_punct 2628
\font 1290, 1297, 1302
font commands:
font_advance 3926
font_box 4211
font_cmap 3857
font_file 3818
font_files 3807

font_glyph 3894
\fontencoding 898
\fontfamily 898
\fontseries 1279, 1289
\fontshape 1280
\fontsize 1281, 1293, 1298
\fontspec *6*, *28*, *124*, *279*
fontspec commands:
\fontspec_gset_family:Nnn 281, 918, 978, 1431, 1504
\footnote *12*, *35*, *237*, *276*
forbid commands:
forbid_marked_disc 3344
forbids commands:
forbids_break 3360
fp commands:
\fp_new:N 388
\fp_set:Nn 130, 399
\fp_use:N 151, 408
\framebox *17*, *40*
free commands:
free_node 1729

G

\gdef 6261, 6268
\global 1290, 1297, 1302, 6307
GLUE 1729
glue commands:
glue_width 5282
GLYPH 1729
group commands:
\group_begin: 75, 82, 87, 279, 722, 756, 866, 977, 1010, 1167, 1195, 1277, 1374, 1430, 1502, 1692, 5539, 5635, 5703, 5836, 5958, 5998, 6017, 6093, 6115, 6124, 6143, 6150
\group_end: 78, 85, 91, 286, 729, 765, 869, 987, 1035, 1174, 1202, 1303, 1379, 1440, 1519, 1703, 5544, 5639, 5707, 5840, 5963, 6002, 6034, 6096, 6120, 6132, 6146, 6164

H

half commands:
half_mark 2076
handle commands:
handle_boundary 2849
handle_source_space 3107
hang commands:
hang_amount 4297
hang_bounds 4369
hang_rule 4310
has commands:
has_attribute 1729
has_char 2348
has_char_cache 2348
hbox commands:
\hbox:n 5857, 5909, 5969, 5975, 6022, 6026, 6136
\hbox_set:Nn . . 761, 5853, 5911, 5971
\hbox_to_wd:nn 5850
head commands:
head_lead 4310
\heiti *7*, *28*, *270*, 1554, 6342
\hfil *12*, *34*, *128*, *137*
\hfill *70*, *273*
hide commands:
hide_list 4904
hide_text 4941
HLIST 1729
hook commands:
\hook_gput_code:nnn . . . 20, 717, 873, 892, 894, 1217, 1391, 1392, 1562, 1565, 1688, 1716, 6338, 6347
\hspace *17*, *41*, *70*, *152*, *275*
hyphenate commands:
hyphenate_runs 4606

I

i16 3809
\IfBooleanT 166
\iffalse 6242
\IfNoValueTF 1531
\ifnum 6218, 6233, 6262, 6273
\ifx 6234, 6253, 6296, 6312, 6316
\ignorespacesafterend 6165

inert commands:
inert_mark 2080
inherit commands:
inherit_attributes 2779
inherit_common_attributes . . . 2785
ink commands:
ink_bounds 4748
ink_of 4748
ink_span 4748
inked commands:
inked_rule 2606
INS 1729
insert commands:
insert_boundary_glue 2985
insert_dots 5025
int commands:
\int_compare:nNnTF 723, 785, 817, 995, 1258, 1365
\int_const:Nn 33, 1158, 5459
\int_eval:n 1016
\int_from_hex:n 300
\int_gincr:N 725, 1257, 1546, 5656, 6063
\int_gset:Nn 719, 1157, 1170, 1198, 1291, 1377, 5542
\int_gzero:N 1370, 5814
\int_new:N 467, 718, 739, 907, 1152, 1155, 1156, 1274, 1362, 1543, 5487, 5503
\int_set:Nn 468, 622, 795, 1308, 1331
\int_step_inline:nnn . . . 1595, 1597
\int_use:N 33, 702, 726, 997, 1028, 1323, 1324, 1336, 1337, 1344, 1384, 1386, 1548, 1645, 1646, 1647, 1648, 1649, 1651, 1652, 1653, 1654, 1655, 1656, 1658, 1659, 1660, 1661, 5459, 5462, 5565, 5582, 5595, 5658, 5903, 6009, 6065
\c_zero_int 785, 1147, 1149, 1179, 1180, 1188, 1189, 1207, 1208
is commands:
is_break_point 3417
is_inserted 2774
ITALICCORR 1729
item commands:
item_label 4310
\itshape 272

K

\kaishu 7, 28, 270, 1554, 6342
KERN 1729
keys commands:
\keys_define:nn 120, 390, 493, 5465, 5752, 6445, 6458
\keys_if_exist:nnTF 5512
\l_keys_key_str 669
\keys_set:nn 143, 401, 676, 5471, 5509, 5513, 5514, 6465
keyval commands:
\keyval_parse:NNn . . . 261, 943, 5521
kind 4733

L

\large 11, 33, 275
last commands:
last_punct 2650
\lccode 6259, 6260, 6266, 6267
\let 6202, 6203, 6214, 6235, 6237, 6241, 6242, 6248
LIGATURE 1729
\linewidth 6154
\lishu 7, 28, 270, 1554, 6342
literal 5316
\llap 129, 271
LOCAL commands:
LOCAL_PAR 1729
\loop 6258, 6271
\lowercase 6261, 6268
\ltjgetparameter 6466
\ltjsetparameter 6464
lua commands:
\lua_escape:n 147, 177, 185, 200, 406, 691, 707, 1385, 1575
\lua_load_module:n 1642, 5457
\lua_now:n 144, 174, 182, 197, 343, 349, 370, 376, 384, 403, 687, 1025, 1073, 1341, 1381, 1572, 1602, 1643, 1670, 5461, 5562, 5585, 5900, 6006
LuaCJK 1721

LuaCJK internal commands:
__LuaCJK_add_fallback:nn 1476, 1477, 1479, 1497, 1497
__LuaCJK_add_family_features:n 1414, 1417, 1417
\g__LuaCJK_attached_attr 22
\c__LuaCJK_attached_attr_int 22, 1656
\c__LuaCJK_attached_chars_clist 1585, 1593
\l__LuaCJK_auto_fallback_bool 447, 647, 651, 705
\l__LuaCJK_autofake_bold_given_bool 909, 932, 951
\g__LuaCJK_autofake_bold_tl . . 420, 431, 439, 447, 641, 642, 948, 955
\l__LuaCJK_autofake_slant_given_bool 909, 933, 961
\g__LuaCJK_autofake_slant_tl . 424, 432, 443, 447, 663, 664, 958, 965
\g__LuaCJK_block_fontid_int 1362, 1370, 1377, 1386
\l__LuaCJK_block_key_tl 222, 1369, 1372
\g__LuaCJK_block_names_clist 189, 244, 1367
\g__LuaCJK_block_prop 189
\l__LuaCJK_blocks_off_clist 203, 210, 211, 213, 707
\l__LuaCJK_bold_font_given_bool 909, 934, 952
__LuaCJK_bool_js:N 679, 692, 693, 694, 695, 701
\l__LuaCJK_bound_width_warned_bool 156, 168, 170
\l__LuaCJK_break_between_puncts_bool . . . 447, 609, 612, 703
__LuaCJK_build_features:n 280, 921, 926, 938, 1503
__LuaCJK_build_ranges:n 193, 315, 342, 348, 369, 375
\l__LuaCJK_ccglue_skip 42, 51, 63, 788, 807, 1662, 6366, 6369, 6381, 6388, 6470
\l__LuaCJK_ccglue_touched_bool 44, 52, 61, 792, 811, 6370, 6382
\l__LuaCJK_check_single_tl 447, 617, 619, 701
\l__LuaCJK_cj_strict_tl 447, 517, 520, 693
\l__LuaCJK_cjk_math_tl 447, 526, 528, 695, 1137
\g__LuaCJK_cjk_space_attr 22, 498, 499
\c__LuaCJK_cjk_space_attr_int 22, 1646
\l__LuaCJK_default_feature_tl 909, 945, 1411
__LuaCJK_detect_shaping: 1666, 1667, 1686
\g__LuaCJK_disabled_attr 22, 713, 715, 796
\c__LuaCJK_disabled_attr_int 22, 1655
\g__LuaCJK_document_started_bool 1215, 1230
\l__LuaCJK_ecglue_skip 42, 56, 68, 790, 809, 1663, 6375, 6401, 6403, 6409, 6413, 6471
\l__LuaCJK_ecglue_touched_bool 45, 57, 66, 794, 813, 6376, 6414
\g__LuaCJK_embolden_factor_tl 421, 438, 447, 640
__LuaCJK_ensure_family_default: 99, 1104, 1104, 1710
__LuaCJK_explicit_nobreak_space: 72, 80, 90
__LuaCJK_explicit_space: 72, 73, 89
\g__LuaCJK_explicit_space_attr 22, 76, 83
\c__LuaCJK_explicit_space_attr_int 22, 1658
__LuaCJK_extract_font_keys:nn 222, 263, 1494
\l__LuaCJK_fallback_family_tl 447, 1505, 1509, 1516
\l__LuaCJK_fallback_ids_tl 447, 1011, 1017, 1018, 1023, 1029

\l__LuaCJK_fallback_key_tl 480, 1472, 1509, 1514, 1518
\g__LuaCJK_fallback_prop 447, 998, 1326, 1513, 1517
\g__LuaCJK_fallback_pushed_prop 991, 1072
\g__LuaCJK_family_block_prop 222, 1371
\l__LuaCJK_family_default_init_tl 1400, 1690
__LuaCJK_family_font:n . . 759, 897, 897, 1014, 1168, 1196, 1278, 1375
\l__LuaCJK_family_name_tl 902, 1165, 1168, 1173, 1181
\g__LuaCJK_family_prop 757, 899, 980, 988, 1082, 1101, 1107, 1109, 1113, 1115, 1142, 1165, 1192, 1234, 1441, 1707
\l__LuaCJK_family_tl 757, 899, 988, 998, 1003, 1019, 1032, 1084, 1140, 1165, 1173, 1369, 1415
\l__LuaCJK_feature_tl 282, 917, 922, 945, 947, 954, 964, 973, 974, 979, 1419, 1421, 1423, 1432, 1444, 1505
\l__LuaCJK_features_tl *56*, 222, 1495, 1501, 1503
\l__LuaCJK_fntef_any_code_tl 5618, 5631, 5717, 5725, 5733, 5750, 5848, 5880, 5907, 5954, 5967, 5994, 6013, 6020, 6032, 6136, 6139
\l__LuaCJK_fntef_any_depth_dim 5897, 5926, 5949, 6013, 6019, 6025
__LuaCJK_fntef_body:n 5525, 5525, 5638, 5706, 5839, 5962, 6001, 6033, 6119, 6127, 6130
\l__LuaCJK_fntef_bottom_dim 5472, 5607, 5614, 5792
\l__LuaCJK_fntef_bottom_given_bool 5472, 5605, 5793, 5812
\l__LuaCJK_fntef_cache_key_tl 5488, 5624, 5631, 5643, 5654, 5697, 5711, 5725, 5726, 5750, 5878, 5880, 5945, 5954, 5992, 5994, 6029, 6032, 6053, 6061, 6084, 6137, 6139
\l__LuaCJK_fntef_cache_name_tl 5489, 5654, 5657, 5662, 5679, 5697, 5699, 6061, 6064, 6069, 6075, 6079, 6084, 6086
__LuaCJK_fntef_capture_color: 6052, 6088, 6088
\l__LuaCJK_fntef_color_key_tl 5472, 6044, 6049, 6057
\g__LuaCJK_fntef_color_ops_tl 5472, 6095, 6097
\l__LuaCJK_fntef_color_ops_tl 5472, 6091, 6097, 6102, 6105
\l__LuaCJK_fntef_commute_bool *216*, 5463, 5464, 5467, 5558, 5583, 5596
__LuaCJK_fntef_decoration_setup:n 5704, 5752, 5823, 5837, 5959, 5999, 6018, 6116
\l__LuaCJK_fntef_depth_dim 5472, 5536, 5604, 5608, 5714, 5720, 5722, 5729, 5738, 5744, 5776, 5815
__LuaCJK_fntef_dots: 6134, 6134, 6144
\l__LuaCJK_fntef_dots_bool 5472, 5530, 5813, 5831
\g__LuaCJK_fntef_dots_color_int . . . 5472, 5542, 5582, 5595, 5814
\l__LuaCJK_fntef_dots_js_tl 5472, 5533, 5548, 5557, 5579, 5592
__LuaCJK_fntef_dots_setup:n 5828, 5828, 6125
\l__LuaCJK_fntef_dots_size_dim . . . 5472, 5534, 5549, 5580, 5593
\l__LuaCJK_fntef_dots_top_dim 5472, 5535, 5550, 5581, 5594
\l__LuaCJK_fntef_em_descent_dim 5845, 5898, 5925, 5948
__LuaCJK_fntef_fill_begin:nn 6148, 6168, 6171
__LuaCJK_fntef_fill_box:nn 6153, 6154, 6156

__LuaCJK_fntef_fill_end: 6161, 6169, 6172
\g__LuaCJK_fntef_format_int .. 5487, 5656, 5658, 6063, 6065
\g__LuaCJK_fntef_format_prop . 5486, 5653, 5696, 6060, 6083
\l__LuaCJK_fntef_format_tl ... 5475, 5537, 5540, 5648, 5651, 5652, 5659, 5796, 5797, 6042, 6048, 6072, 6090
\l__LuaCJK_fntef_gap_dim 5496, 5506, 5731, 5740, 5746, 5780, 5817
\l__LuaCJK_fntef_height_dim .. 5497, 5507, 5646, 5667, 5672, 5686, 5691, 5782, 5818
\l__LuaCJK_fntef_hidden_bool . 5481, 5556, 5572, 5772, 5810
\l__LuaCJK_fntef_hidden_js_tl 5479, 5573, 5574, 5578, 5591
__LuaCJK_fntef_literal_-direct:n 6088, 6104, 6108
__LuaCJK_fntef_option_bare:n 5508, 5510, 5522
__LuaCJK_fntef_option_-pair:nn 5508, 5508, 5522
__LuaCJK_fntef_options:n 5508, 5516, 5602, 5825, 5830
\l__LuaCJK_fntef_pattern_box . 5842, 5853, 5872, 5873, 5874, 5911, 5918, 5927, 5930, 5932, 5939, 5971, 5987, 5988, 5989
__LuaCJK_fntef_pattern_-color: 5860, 5978, 6088, 6100
__LuaCJK_fntef_prepare_-underline:n 5600, 5636
__LuaCJK_fntef_push_settings: 5527, 5527, 5610, 5826, 5832
__LuaCJK_fntef_reset_decoration: 5518, 5805
\l__LuaCJK_fntef_rule_depth_-dim 5485, 5613, 5622, 5628
\l__LuaCJK_fntef_rule_height_-dim 5484, 5611, 5621, 5627
\g__LuaCJK_fntef_settings_-attr 5458, 5460
\c__LuaCJK_fntef_settings_-attr_int 5459, 5462, 5565
\l__LuaCJK_fntef_settings_-key_tl *218*, 5472, 5561, 5575, 5588, 5649, 6058
__LuaCJK_fntef_silent_begin: 5882, 5936
\l__LuaCJK_fntef_silent_bool . 5472, 5774, 5809, 5884, 5892, 5950
__LuaCJK_fntef_silent_end: .. 5890, 5940
\l__LuaCJK_fntef_skip_js_tl .. 5478, 5570, 5571, 5578, 5590
\l__LuaCJK_fntef_skip_tl 5477, 5555, 5569, 5768, 5769, 5808
__LuaCJK_fntef_slash: 5965, 6000
\l__LuaCJK_fntef_subtract_dim 5476, 5554, 5577, 5589, 5757, 5760, 5761, 5807
\l__LuaCJK_fntef_symbol_kern_-dim 5843, 5913, 5935, 5942
\l__LuaCJK_fntef_symbol_-raise_dim 5844, 5921, 5938
\l__LuaCJK_fntef_symbol_sep_-dim 5800, 5821
\l__LuaCJK_fntef_symbol_tl ... 5498, 5531, 5799, 5912, 5951, 5960, 6117, 6126
\l__LuaCJK_fntef_textformat_-tl 5472, 5526, 5802
\l__LuaCJK_fntef_thickness_-dim 5472, 5608, 5647, 5668, 5673, 5687, 5692, 5715, 5722, 5730, 5739, 5745, 5778, 5816
\l__LuaCJK_fntef_top_dim 5472, 5604, 5612, 5786
\l__LuaCJK_fntef_top_given_-bool 5472, 5603, 5787, 5811
__LuaCJK_fntef_underdot:n ... 6127, 6134, 6141
__LuaCJK_fntef_underline: ... 5616, 5616, 5637
__LuaCJK_fntef_use_double: .. 5709, 5709, 5838

__LuaCJK_fntef_use_drawn_-type:nn 226, 230, 5879, 5993, 6038
__LuaCJK_fntef_use_leader_-type:nn . . 220, 226, 5630, 5724, 5749, 5953, 6031, 6036, 6036, 6138
__LuaCJK_fntef_use_sout: 5641, 5705
__LuaCJK_fntef_use_symbol:n 5895, 5961, 6118, 6129
__LuaCJK_fntef_use_type:nnn 6037, 6039, 6040
__LuaCJK_fntef_wave: . . 5846, 5961
\g__LuaCJK_font_attr 22, 1175, 1179, 1713
\c__LuaCJK_font_attr_int 22, 1645, 5903, 6009
\g__LuaCJK_font_feature_prop 909, 922, 973, 1419, 1443
\g__LuaCJK_font_name_prop 909, 923, 971, 1426
__LuaCJK_font_option:n . . 255, 261
__LuaCJK_font_option:nn . 236, 261
\g__LuaCJK_font_shift_prop 447, 984, 1045, 1437, 1510
\g__LuaCJK_fontid_int 995, 997, 1028, 1151, 1170, 1175, 1365, 1384, 1713
\l__LuaCJK_fontid_tl 447, 997, 1001, 1032
\g__LuaCJK_fontspec_int 1543, 1546, 1548
\l__LuaCJK_fontspec_tl 902, 978, 980, 983, 1431, 1436, 1442
__LuaCJK_fv_patch: 840, 851, 874, 875
__LuaCJK_fv_prep_hook: 74, 840, 845, 857
__LuaCJK_fv_use_verbatim: 863, 868
\l__LuaCJK_hex_int_tl 289, 299, 310
__LuaCJK_hex_store:n 294, 308
\l__LuaCJK_hex_tl 289, 296, 297, 298, 300
\g__LuaCJK_inserted_attr 22
\c__LuaCJK_inserted_attr_int 22, 1648
\l__LuaCJK_is_block_bool 228, 243, 247, 249
\l__LuaCJK_italic_font_given_-bool 909, 935, 962
\l__LuaCJK_latin_punct_tl 453, 454, 600, 602, 694
__LuaCJK_load_cj: . 521, 1601, 1601
__LuaCJK_load_config_file:n 1607, 1610
__LuaCJK_load_local_config: 1603, 1603, 1715
__LuaCJK_load_lxgw: 1618, 1618, 1708
\g__LuaCJK_load_lxgw_bool 447, 655, 657, 1705
\l__LuaCJK_local_config_tl 447, 627, 628, 629, 1605, 1607
\g__LuaCJK_lua_loaded_bool 685, 991, 1317, 1665
\g__LuaCJK_math_bracket_attr . . 22
\c__LuaCJK_math_bracket_attr_-int 22, 1651
\l__LuaCJK_math_fallback_ids_-tl 1314, 1320, 1321, 1334, 1345
__LuaCJK_math_fallbacks_-push:n 1145, 1314, 1315
__LuaCJK_math_family:n 1248, 1253, 1307, 1330
__LuaCJK_math_family_fonts:nn 1274, 1275, 1311, 1333
\g__LuaCJK_math_family_int 1156, 1257, 1258, 1261
\c__LuaCJK_math_family_max_-int 1156, 1258
\g__LuaCJK_math_family_prop 1156, 1255, 1260, 1309, 1332
\l__LuaCJK_math_family_tl 1139, 1141, 1153, 1540
\g__LuaCJK_math_font_attr 22, 1147, 1149, 1312, 1324, 1344
\c__LuaCJK_math_font_attr_int 22, 1649

__LuaCJK_math_font_update: 1042, 1135, 1135, 1183, 1354
__LuaCJK_math_size_set: 1264, 1265, 1310, 1319
\l__LuaCJK_math_size_tl 1264, 1281, 1294, 1299
\g__LuaCJK_math_textfont_int 87, 1274, 1291, 1323, 1336
__LuaCJK_math_version_-refresh: 83, 1349, 1349, 1361
\l__LuaCJK_new_family_tl 96, 902, 1547, 1550, 1551
\g__LuaCJK_nobreak_attr 22, 723, 726, 799, 816
\c__LuaCJK_nobreak_attr_int 22, 1653
\g__LuaCJK_nobreak_id_int 718, 725, 726
__LuaCJK_nobreak_space: 72, 72, 84
\g__LuaCJK_normal_spaced_-chars_tl 1567, 1575, 1580
__LuaCJK_note_feature_key:n 927, 928, 944
__LuaCJK_note_feature_key:nn 926, 944
\l__LuaCJK_option_key_tl 222
\l__LuaCJK_option_seq 222
__LuaCJK_parse_font_options:n 222, 257, 266, 1499
\g__LuaCJK_pending_block_prop 222
\l__LuaCJK_punct_bound_width_-dim 447, 590, 594, 700
\l__LuaCJK_punct_bound_width_-given_bool 447, 589, 593, 699
\l__LuaCJK_punct_compress_-bool 387, 392, 398, 407
\l__LuaCJK_punct_compress_-given_bool 115, 124, 140, 148
\l__LuaCJK_punct_compress_-value_tl 118, 125, 149
\l__LuaCJK_punct_family_tl 580, 581, 1153, 1186, 1192, 1201, 1232, 1234, 1237, 1240, 1242
\g__LuaCJK_punct_family_-unknown_seq 1215, 1237, 1239
\g__LuaCJK_punct_font_attr 22, 1188, 1203, 1207
\c__LuaCJK_punct_font_attr_-int 22, 1652
\g__LuaCJK_punct_fontid_int 1153, 1198, 1203
\l__LuaCJK_punct_keep_bool 387, 394, 400, 409
\l__LuaCJK_punct_keep_given_-bool 115, 134, 142, 152
\l__LuaCJK_punct_keep_value_-tl 119, 135, 153
\l__LuaCJK_punct_kinsoku_tl 447, 511, 513, 692
\l__LuaCJK_punct_ratio_fp 130, 151, 387, 393, 399, 408
\l__LuaCJK_punct_ratio_given_-bool 115, 129, 141, 150
\g__LuaCJK_punct_shift_attr 22, 1189, 1204, 1208
\c__LuaCJK_punct_shift_attr_-int 22, 1660
\g__LuaCJK_punct_style_prop 92, 108, 402
\l__LuaCJK_punct_style_tl 99, 109, 112, 691
\l__LuaCJK_punct_width_dim 447, 569, 573, 697
\l__LuaCJK_punct_width_given_-bool 447, 568, 572, 697
__LuaCJK_push_block_fonts: 1185, 1362, 1363
__LuaCJK_push_class:nn 328, 329, 335, 336, 337, 346
__LuaCJK_push_fallback: 991, 993, 1041, 1184
__LuaCJK_push_fallback_aux:n 1004, 1008
__LuaCJK_push_normal_space:n 334, 340
__LuaCJK_push_punct:nn 330, 331, 332, 333, 356, 357, 358, 359, 360, 361, 362, 363, 364, 367

__LuaCJK_push_shift:nnn *80*, 1019, 1045, 1061
__LuaCJK_put_feature:n 230, 252, 256
__LuaCJK_range_entry:n . . 302, 318
\l__LuaCJK_range_pair_tl 289, 305, 309, 313
\l__LuaCJK_range_seq . 289, 304, 306
\l__LuaCJK_ranges_tl 194, 200, 289, 312, 317, 344, 350, 371, 379
__LuaCJK_refresh_fallbacks: 648, 991, 1037, 1520
__LuaCJK_refresh_glues: 59, 1396
__LuaCJK_register_block_-fonts:n 273, 275
__LuaCJK_register_family:n 924, 969, 969
__LuaCJK_register_math_-family:n 1144, 1305, 1305
\l__LuaCJK_registered_name_tl 908, 971, 972, 979, 1426, 1427, 1432
\c__LuaCJK_removed_chars_-clist 1583, 1589
__LuaCJK_reset_family: 1098, 1098, 1566, 1711
\l__LuaCJK_selectfont_bool 1040, 1043, 1151, 1162, 1164, 1212, 1351, 1353, 1355, 1394
__LuaCJK_set_autofake:nn 412, 412, 632, 635
__LuaCJK_set_block_enabled:nn 204, 219, 221
__LuaCJK_set_ccglue:n 49, 49, 495, 6460
__LuaCJK_set_charset:nnn 373, 373, 541, 543, 545, 547, 549, 551, 553, 555, 557, 559, 561, 563
__LuaCJK_set_ecglue:n 49, 54, 496, 6461
__LuaCJK_set_family:nnn 899, 919, 1491, 1495
__LuaCJK_set_family_with_-options:nnnn *93*, 1451, 1456, 1461, 1467, 1481, 1481, 1534, 1539, 1549
__LuaCJK_set_family_with_-options_aux:nnn 1486, 1487, 1489, 1492
__LuaCJK_set_normal_spaced_-chars: 385, 1570, 1581, 1685
__LuaCJK_set_punct_style:n 99, 106, 507
\g__LuaCJK_settings_attr 22
\c__LuaCJK_settings_attr_int 22, 1654
__LuaCJK_settings_push: 215, 677, 679, 683, 815, 1397, 1684, 6371, 6377, 6383, 6415
__LuaCJK_setup_format_chars: 1583, 1587, 1687
\g__LuaCJK_shaping_bool 1591, 1666, 1672, 1673
\g__LuaCJK_shift_attr 22, 1176, 1180
\c__LuaCJK_shift_attr_int 22, 1659
\g__LuaCJK_shift_decl_prop 267, 447, 975, 1428
\l__LuaCJK_shift_dim_tl 447, 975, 976, 981, 985, 1045, 1428, 1429, 1433, 1438
\l__LuaCJK_shift_fid_tl . 447, 1045
\l__LuaCJK_shift_key_tl . . 447, 983, 985, 1045, 1435, 1438, 1508, 1511
\l__LuaCJK_shift_old_tl . 447, 1045
\g__LuaCJK_shift_pushed_prop 447, 1045
\l__LuaCJK_shift_tl *56*, 222, 1500, 1506, 1511
__LuaCJK_shift_value:nn *79*, 1045, 1045, 1064, 1172, 1200
\g__LuaCJK_shift_value_tl *79*, 447, 1045, 1176, 1204
\g__LuaCJK_shifted_attr 22
\c__LuaCJK_shifted_attr_int 22, 1661
\g__LuaCJK_slant_factor_tl 425, 442, 447, 662
__LuaCJK_switch_family:n *75*, 1080, 1080, 1102, 1121, 1525, 1533, 1551, 1563

\l__LuaCJK_tmpa_int 907, 1308, 1311, 1312, 1331, 1333, 1337
\l__LuaCJK_tmpa_tl 281, 285, 757, 759, 902, 998, 1004, 1100, 1101, 1102, 1106, 1107, 1119, 1140, 1141, 1142, 1145, 1193, 1196, 1201, 1326, 1328, 1372, 1375, 1466, 1468, 1514, 1515, 1516, 1518, 1524, 1525, 1530, 1533, 1535
\l__LuaCJK_tmpb_tl 906, 1001, 1002, 1003, 1015, 1017, 1019, 1112, 1113, 1116, 1119, 1120, 1121, 1142, 1144
__LuaCJK_toks_put_right:Nn 1358, 1358, 1360
\g__LuaCJK_unknown_family_seq 1088, 1090, 1095
__LuaCJK_update_font: *75*, *78*, *83*, *85*, *91*, *96*, 216, 526, 528, 582, 1151, 1160, 1391, 1541, 1712
\l__LuaCJK_verb_addon_bool 731, 800, 805, 820
__LuaCJK_verb_addon_compute: 747, 798
\l__LuaCJK_verb_box . . 731, 761, 763
\l__LuaCJK_verb_cjk_dim 731, 755, 762, 767
\l__LuaCJK_verb_gap_dim 731, 766, 768, 776, 777
\l__LuaCJK_verb_mode_tl . . 447, 532, 534, 536, 538, 825, 828, 842, 883
\l__LuaCJK_verb_saved_ccglue_skip 731, 787, 808
\l__LuaCJK_verb_saved_ccglue_touched_bool 731, 791, 812
\l__LuaCJK_verb_saved_disabled_int 731, 795, 817
\l__LuaCJK_verb_saved_ecglue_skip 731, 789, 810
\l__LuaCJK_verb_saved_ecglue_touched_bool 731, 793, 814
\l__LuaCJK_verb_space_dim 731, 749, 755, 767
__LuaCJK_verbatim@font_hook: 823, 823, 837, 849
\l__LuaCJK_verbatim_block_bool 447, 847, 867, 878
__LuaCJK_verbatim_block_disable: 840, 840, 848, 879
__LuaCJK_verbatim_env_begin: 876, 876, 893
__LuaCJK_verbatim_env_end: 876, 881, 895
\c__LuaCJK_version_tl 11, 12
__LuaCJK_warn_punct_family: 1209, 1215, 1220, 1228
\l__LuaCJK_widow_penalty_int 447, 622, 702
__LuaCJK_width_js:Nn 681, 696, 698
\g__LuaCJK_xecglue_attr 22, 503, 504
\c__LuaCJK_xecglue_attr_int 22, 1647

LuaCJK.attr commands:
LuaCJK.attr_cjk_space 1756
LuaCJK.attr_disabled 2282
LuaCJK.attr_explicit_space . . 1756
LuaCJK.attr_font 1756
LuaCJK.attr_inserted 1756
LuaCJK.attr_math_font 1756
LuaCJK.attr_settings 1765
LuaCJK.attr_xecglue 1756
LuaCJK.cj 1867

LuaCJK.cj commands:
LuaCJK.cj_count 1867
LuaCJK.cj_loaded 1867

LuaCJK.declare commands:
LuaCJK.declare_ranges 2101

LuaCJK.define commands:
LuaCJK.define_punct_style . . . 2211
LuaCJK.hpack 4598
LuaCJK.hyphenate 4630

LuaCJK.is commands:
LuaCJK.is_attached 2009
LuaCJK.is_cjk 1909
\LuaCJK.is_cjk . . . *108*, *113*, *128*, *239*

LuaCJK.new commands:
LuaCJK.new_settings 1785

LuaCJK.pre commands:
LuaCJK.pre_linebreak 4594

LuaCJK.pre_mlist_to_hlist . . . 4602
LuaCJK.process 4576
LuaCJK.punct commands:
LuaCJK.punct_codes 2191
LuaCJK.punct_styles 2172
LuaCJK.set commands:
LuaCJK.set_charset 2150
LuaCJK.settings 1765
LuaCJK.settings commands:
LuaCJK.settings_by_signature 1765
LuaCJK.shaping 1754
LuaCJK.warned commands:
LuaCJK.warned_cjk_math 3730
\LuaCJKCancelSubCJKBlock 4, 8, 26, 29, 269, 203
\LuaCJKDeclareCharClass 5, 7, 14, 15, 26, 28, 37, 38, 60, 108, 268, 289
\LuaCJKDeclarePunctClass 5, 15, 26, 38, 268, 352
\LuaCJKDeclarePunctStyle 16, 24, 39, 47, 52, 117, 268, 387
\LuaCJKDeclareSubCJKBlock 5, 7, 26, 28, 269, 189
\LuaCJKdisable 18, 41, 69, 70, 269, 279, 712, 818, 826, 829, 843, 6317
\LuaCJKEditPunctStyle . 15, 38, 269, 115
\LuaCJKemdescent 22, 45, 46, 269, 5536, 6004
\LuaCJKenable 18, 41, 269, 279, 712, 797, 819, 884
LuaCJKfntef commands:
\LuaCJKfntef_use_symbol:n 235
LuaCJKfntef.attr 4658
LuaCJKfntef.color commands:
LuaCJKfntef.color_attr 4658
LuaCJKfntef.dot commands:
LuaCJKfntef.dot_attr 4658
LuaCJKfntef.em commands:
LuaCJKfntef.em_descent 4681
LuaCJKfntef.hpack 5434
LuaCJKfntef.new commands:
LuaCJKfntef.new_settings 4713
LuaCJKfntef.settings 4658
LuaCJKfntef.settings commands:
LuaCJKfntef.settings_by_key 4658
\LuaCJKfntefsetup 20, 43, 279, 5463
\LuaCJKknobreak 17, 24, 41, 48, 69, 152, 269, 270, 275, 277, 279, 720
\LuaCJKOffVerbAddon 18, 42, 70, 269, 273, 731
\LuaCJKResetCharClass . 15, 38, 268, 382
\LuaCJKRestoreSubCJKBlock 4, 8, 26, 29, 269, 203
\LuaCJKsetkern 5, 16, 23, 26, 39, 47, 138, 146, 269, 277, 278, 180
\LuaCJKsetup 4, 25, 62, 68, 73, 447, 771, 774, 781, 6151, 6425
\LuaCJKsetwidth 5, 13, 16, 26, 35, 39, 157, 269, 156
\LuaCJKShipoutHook 18, 41, 269, 716
\LuaCJKVerbAddon 18, 19, 42, 69, 269, 731
\LuaCJKversion 4, 25, 269, 11
luaul commands:
\luaul_apply_color: 5541, 5682, 6080, 6094
\luaul_context_color: . . 5680, 6050
\luaul_set_color:n 5540, 5652, 6048

M

M 1729
M.block commands:
M.block_fonts 2292
M.block_order 2292
M.blocks 2292
M.charsets 2125
M.fallback 2336
M.math commands:
M.math_fallback 3664
M.yoffset 2342
make commands:
make_glue 2797
make_kern 2835
\makeatletter 6176
\makeatother 6319
MARK 1729
mark commands:
mark_blank 4248
mark_blanks 4279
mark_math_brackets 3624
MATH 1729

MATH commands:
MATH_CHAR 3608
MATH_TEXT_CHAR 3608
math commands:
math_bracket 2886
math_family_for 3670
\mbox 13, 36, 181, 183, 274, 276, 278
\mddefault 10, 32, 87, 1279
\meta 276
\minipage 6159
msg commands:
\msg_critical:nn 7, 19, 21, 5453, 6327, 6337, 6339
\msg_error:nn 1259
\msg_new:nnn . 5, 13, 101, 157, 320, 322, 672, 741, 1096, 1125, 1130, 1222, 1248, 1616, 1676, 5451, 6325
\msg_new:nnnn 6330
\msg_warning:nn 171, 770, 1110, 1674, 6432
\msg_warning:nnn 111, 338, 365, 668, 1091, 1241, 1447, 1614
\msg_warning:nnnn 1118

N

nested commands:
nested_pair 4798
new commands:
new_node 1729
\newattribute 32, 5458
\newCJKfontfamily . 6, 27, 267, 270, 1528
\NewDocumentCommand 138, 164, 180, 191, 218, 220, 324, 352, 382, 396, 674, 712, 714, 716, 720, 783, 803, 1410, 1412, 1449, 1454, 1459, 1464, 1470, 1522, 1528, 1537, 1544, 1578, 5470, 5633, 5701, 5834, 5956, 5996, 6015, 6113, 6122, 6464
\NewDocumentEnvironment . . . 6167, 6170
\newfontfamily 6, 27
\newif 6177, 6178, 6179
\newunderlinetype 5661, 5678, 6068, 6074, 6078
next commands:
next_glyph 3334
next_visible 3400
no commands:
no_break_mark 3319
\nobreak 272
\nobreakspace 72, 90
node commands:
node_width 4741
\node.subtypes 161
\normalfont 270
\normalspacedchars 11, 15, 33, 38, 60, 269, 1567
\numberline 12, 34, 128, 137

O

open commands:
open_hang 4310

P

pair commands:
\pair_kern 142
pair_kern 2849
\parbox 12, 34, 128, 237, 271, 274
\parfillskip 12, 34, 128
PENALTY 1729
\pingfang 6342
place 5158
placeholder 2438
pre commands:
pre_shaping 2442
prg commands:
\prg_do_nothing: . . 6348, 6350, 6351
\prg_new_conditional:Npnn . . . 6385
\prg_return_false: 6389
\prg_return_true: 6389
\ProcessKeyOptions 1683
prop commands:
\prop_gclear:N 265, 1039, 1072
\prop_get:NnNTF . . . 489, 757, 971, 973, 975, 998, 1000, 1048, 1065, 1142, 1165, 1192, 1326, 1371, 1419, 1426, 1428, 1513, 5653, 6060
\prop_gput:Nnn . 93, 94, 95, 96, 97, 98, 194, 250, 267, 284, 402, 490,

922, 923, 980, 984, 1031, 1070, 1260, 1437, 1510, 1517, 5696, 6083
\prop_if_empty:NTF . 268, 1109, 1707
\prop_if_in:NnTF 108, 988, 1082, 1101, 1107, 1113, 1234, 1255
\prop_item:Nn 1309, 1332
\prop_map_break: 1116
\prop_map_inline:Nn 277, 1115
\prop_new:N 92, 189, 222, 223, 476, 481, 482, 483, 899, 913, 914, 991, 1159, 5486
\prop_put:Nnn 1441, 1443
PUNCT commands:
PUNCT_CLASS 2021
punct commands:
punct_class 2084
punct_style 2219

Q

\quad *17*, *41*, *70*, *152*, *275*

R

\raisebox *129*, *137*
\relax . . 6234, 6248, 6257, 6262, 6266, 6267, 6270, 6272, 6273, 6296, 6303
\repeat 6264, 6275
replace commands:
replace_pair 5061
\RequirePackage 10, 1717, 1719, 5456, 6175, 6340
restore commands:
restore_source_spaces 3065
rewrite commands:
rewrite_fonts 2526
rewrite_math 3662
rewrite_math_list 3662
rewrite_noad 3686
\rlap *129*, *271*
\rmdefault 1697
\rule *22*, *46*
run commands:
run_mark 3356

S

same commands:
same_break_span 3327
scale commands:
scale_ink 5323
scan commands:
\scan_stop: 76, 83, 713, 715, 726
\scriptfont 1297
\scriptscriptfont 1302
\scriptsize *11*, *33*, *275*
segments 4805
\selectfont 760, 989, 1014, 1085, 1169, 1197, 1282, 1289, 1296, 1301, 1376, 1445
seq commands:
\seq_gput_right:Nn 1090, 1239
\seq_if_in:NnTF 1088, 1236
\seq_map_inline:Nn 306
\seq_new:N 227, 290, 1095, 1216
\seq_set_split:Nnn 304
set commands:
set_attribute 1729
set_glue 2819
\setCJKfallbackfamilyfont *8*, *29*, *268*, 1470
\setCJKfamilyfont *4*, *6*, *13*, *25*, *27*, *36*, *93*, *267*, *273*, 1449, 1628, 1632, 1634, 1639
\setCJKmainfont *4*, *6*, *25*, *27*, *92*, *93*, *267*, 1128, 1449, 1620
\setCJKmathfont *10*, *32*, *83*, *163*, *269*, 1537
\setCJKmonofont *4*, *6*, *25*, *27*, *267*, 1449, 1626
\setCJKsansfont *4*, *6*, *25*, *27*, *267*, 1449, 1624
settings commands:
settings_at 4839
settings_of 1846
\sfdefault 1698
\shapedefault *10*, *32*, *87*, 1280, 1287
shift commands:
shift_of 2406
single commands:
single_rule 4850
skip commands:
\skip_horizontal:n 5934, 5941
\skip_if_eq:nnTF 6387, 6408

\skip_new:N 42, 43, 734, 735, 6365, 6402
\skip_set:Nn 46, 47, 51, 56, 63, 68, 6401, 6411, 6450
\skip_set_eq:NN 787, 789, 807, 809, 6366, 6369, 6375, 6381, 6403, 6412
\skip_use:N 6470, 6471
skippable 2577
\songti 7, 28, 270, 1554, 6342
SPACESKIP 1729
spacing 2849
str commands:
\str_case:nn 930
\str_case:nnTF 326, 354, 414, 625, 1695, 6468
\str_if_eq:nnTF 208, 240, 246, 418, 430, 436, 566, 579, 587, 680, 1669, 5569, 5756, 5759, 5767, 6052, 6066
\str_if_eq_p:nn . . . 1284, 5555, 5557
SUB commands:
SUB_MLIST 3622
suspended commands:
suspended_break 3147
sys commands:
\sys_if_engine_luatex:TF 3, 5449, 6323

T

target commands:
target_font 2377
TEX and LATEX 2ε commands:
\@empty 6244, 6253, 6312, 6316
\@makefnmark 129
\@ne 6184, 6190, 6263, 6274
\@nil 6244, 6249, 6254
\@tempcnta 6257, 6259, 6260, 6262, 6263, 6270, 6272, 6273, 6274
\addCJKfontfeatures 10, 31
\begin 12, 35
\char 12, 34, 97
\CJKfamily 6, 27
\CJKfontspec 6, 28
\CJKsout 23, 46
\CJKunderanyline 22, 46
\CJKunderanysymbol 22, 46
\CJKunderdblline 21, 45
\CJKunderdot 23, 46
\CJKunderline 20, 43
\CJKunderwave 21, 45
\CJKxout 22, 45
\cleaders 191, 226
\color 220
\colorbox 142, 277
\CTEX@alchar 6349
\CTEX@beginallalchar 6350
\CTEX@endallalchar 6351
\current@color 233, 234
\defaultCJKfontfeatures 10, 31
\edef 51
\emph 9, 30
\end 12, 35
\every@math@size 90
\f@size . . . 87, 1267, 6354, 6356, 6360
\fbox 129, 181, 183, 275, 278
\fcolorbox 129, 275
\fi 238
\footnote 275
\framebox 129, 183, 275, 278
\FV@FormattingPrep . . . 853, 855, 858
\FV@UseVerbatim 861, 863, 864
\FVC@Verb 73
\glb@currsize . . 87, 1268, 1270, 1271
\glb@settings 89
\hbox 226, 227
\hfil 145
\hfill 145, 146
\hpack 188
\hskip 144, 146
\hss 226
\ifLuaCJK@listings@breakrun 6179, 6226
\ifLuaCJK@listings@cjkrun 6177, 6196, 6205, 6210, 6217
\ifLuaCJK@listings@defined 6178, 6305
\itshape 9, 30
\kern 271
\lccode 238

\leaders 22, 46, 191, 198, 212, 220, 226, 227, 231, 232, 235
\leftskip 182, 183
\linewidth 237
\lpcode 166
\lst@AddToHook 6180, 6311, 6315
\lst@Append 238, 6185, 6191
\lst@AppendLetter 6202, 6204
\lst@AppendOther 6203, 6209
\lst@CheckDelay 6232
\lst@currstyle 6234, 6237, 6241
\lst@identifierstyle 6235
\lst@ifbreakatwhitespace 6220
\lst@ifbreaklines 6219
\lst@ifkeywords 6242
\lst@ifletter 6199
\lst@inlinechars 6312, 6316
\lst@lastother 6248
\lst@length 6184, 6190, 6218, 6233, 6251
\lst@letterfalse 6193, 6211
\lst@lettertrue . . . 6197, 6199, 6206
\lst@Output 238, 6192, 6199, 6206, 6211, 6214, 6215
\lst@OutputOther 6199
\lst@OutputToken 6252
\lst@PrintToken 238
\lst@ResetToken 6246
\lst@thestyle 6235, 6237, 6241
\lst@token 6243, 6250
\lst@whitespacefalse . . . 6182, 6187
\lsthk@Output 6239
\lstinline 19, 43, 242
\LuaCJK@listings@activate 6304, 6313
\LuaCJK@listings@breakrunfalse 6216
\LuaCJK@listings@breakruntrue 6221
\LuaCJK@listings@catcodes 6269, 6309
\LuaCJK@listings@cjkrunfalse 6180, 6188, 6194, 6206, 6211
\LuaCJK@listings@cjkruntrue 6200
\LuaCJK@listings@codes . 6301, 6303
\LuaCJK@listings@define 6256, 6306
\LuaCJK@listings@definedtrue 6307
\LuaCJK@listings@definepunct 6265, 6297
\LuaCJK@listings@emit 6244, 6249, 6254
\LuaCJK@listings@letter 6202, 6208
\LuaCJK@listings@other . 6203, 6213
\LuaCJK@listings@output 6214, 6229
\LuaCJK@listings@process 6181, 6261
\LuaCJK@listings@processpunct 6186, 6268
\LuaCJK@listings@punctuate 6295, 6298, 6303
\LuaCJK@listings@punctuation 6300, 6310
\LuaCJK@listings@ranges 6276, 6306, 6309
\LuaCJK@listings@run . . . 6243, 6244
\LuaCJK@listings@setcjkflag 6183, 6189, 6195
\LuaCJK@listings@splitrun 6227, 6231
\LuaCJKCancelSubCJKBlock 8, 29
\LuaCJKDeclareCharClass 15, 38
\LuaCJKDeclarePunctClass . . . 15, 38
\LuaCJKDeclarePunctStyle . . . 16, 39
\LuaCJKDeclareSubCJKBlock . . . 7, 28
\LuaCJKdisable 18, 41
\LuaCJKEditPunctStyle 15, 38
\LuaCJKenable 18, 41
\LuaCJKfntefsetup 20, 43
\LuaCJKnobreak 17, 41
\LuaCJKOffVerbAddon 18, 42
\LuaCJKRestoreSubCJKBlock . . . 8, 29
\LuaCJKsetkern 16, 39
\LuaCJKsetup 4, 25
\LuaCJKsetwidth 16, 39
\LuaCJKShipoutHook 18, 41
\math@version 1284
\mathchoice 87
\mathversion 10, 32, 89
\mbox 185, 267, 276, 278
\newCJKfontfamily 6, 27

\nobreak 17, 40
\nobreakspace 11, 33, 50, 51, 145, 146
\normalfont 7, 28, 81, 91, 101
\normalspacedchars 11, 33
\p@ 6354, 6356, 6360
\parbox 276
\parfillskip 276
\parindent 181
\phantom 271
\protrudechars 166
\raise 230
\raisebox 129, 275
\rightmarginkern 166
\rlap 142
\rmfamily 7, 28, 91
\rpcode 166
\scan_stop 51, 69, 70
\selectfont 75, 83
\setCJKfallbackfamilyfont . . . 8, 29
\setCJKfamilyfont 6, 27
\setCJKmainfont 6, 27
\setCJKmathfont 10, 32
\sffamily 7, 28, 95
\space 51
\text 11, 32, 87, 272
\textit 276
\textsuperscript 275
\texttt 19, 42
\the 90
\ttfamily 7, 28
\tw@ 6218, 6251
\underline 185, 276, 278
\UseVerb 19, 42, 272
\UseVerbatim 19, 42, 272
\Verb 19, 42, 73, 272
\verb 19, 42, 73, 74, 272
\verbatim@font 73, 74, 272, 832, 834, 836
\VerbatimInput 19, 42, 272
\write 51
\XeTeXglyphbounds 167
\xleaders 191, 212, 227
\xout 22, 45, 230
\z@ 6233
tex commands:
\tex_attribute:D 5903, 6009
\tex_baselineskip:D 46, 64
\tex_currentgrouplevel:D 785
\tex_endinput:D 8, 5454, 6328
\tex_expanded:D 6110
\tex_font:D 750, 752, 753, 1016, 1171, 1199, 1292, 1378
\tex_fontcharwd:D 753
\tex_fontdimen:D 750, 752
\tex_fontid:D 1016, 1171, 1199, 1292, 1378
\tex_hss:D 5852, 5875
\tex_ignorespaces:D 1526, 1552
\tex_leaders:D 5620, 5664, 5683, 5719, 5735, 5850, 5909, 5969, 6022, 6136
\tex_pdfextension:D 5886, 5893, 6111
\tex_the:D 1359, 5543, 6045
\tex_vrule:D 5620, 5664, 5683, 5719, 5735
\textbf 144, 270, 273
\textcolor 21, 44, 274
\textfont 1290
\textnormal 270
\textsubscript 17, 40, 129, 137, 273
\textsuperscript . . 17, 40, 129, 137, 273
\the 6243
tile 5341
tl commands:
\tl_clear:N 259, 260, 305, 317, 580, 628, 972, 974, 976, 1002, 1011, 1050, 1181, 1320, 1427, 1429, 1515, 5561, 5820
\tl_const:Nn 11
\tl_gclear:N 431, 432
\tl_gset:Nn 438, 439, 442, 443, 473, 475, 640, 642, 662, 664, 1052, 1054, 1568, 1580, 1693
\tl_gset_eq:NN . 420, 424, 1120, 6095
\tl_if_blank:nTF 946, 1424
\tl_if_empty:NTF 232, 270, 641, 663, 948, 958, 981, 1023, 1051, 1139, 1186, 1232, 1270, 1433, 1500,

1506, 1605, 5537, 5651, 5659, 5960, 6042, 6072, 6090, 6102, 6126
\tl_if_empty_p:N 5531
\tl_if_eq:NNTF 1003, 1068, 1690
\tl_if_eq:NnTF 1137
\tl_if_eq:nnTF . . . 825, 828, 842, 883
\tl_if_novalue:nTF 1473, 1475, 1483, 1485, 5519
\tl_new:N 99, 118, 119, 224, 225, 226, 229, 289, 291, 292, 293, 447, 449, 451, 453, 455, 465, 469, 470, 471, 472, 474, 479, 480, 484, 485, 486, 487, 488, 491, 492, 900, 902, 903, 904, 905, 906, 908, 915, 917, 1153, 1154, 1264, 1314, 1402, 1406, 1408, 1567, 5474, 5475, 5477, 5478, 5479, 5488, 5489, 5490, 5491, 5492, 5493, 5498, 5500, 6013, 6404, 6428
\tl_put_right:Nn *90*, 234, 309, 312, 947, 954, 964, 1017, 1018, 1321, 1334
\tl_replace_all:Nnn 297
\tl_set:Nn 100, 109, 112, 125, 135, 233, 238, 241, 271, 283, 296, 299, 448, 450, 452, 454, 456, 466, 511, 513, 517, 520, 526, 528, 532, 534, 536, 538, 581, 600, 602, 617, 619, 627, 629, 901, 916, 983, 997, 1015, 1047, 1063, 1067, 1084, 1100, 1106, 1112, 1116, 1369, 1403, 1407, 1411, 1421, 1435, 1466, 1472, 1501, 1508, 1524, 1530, 1540, 1547, 5533, 5548, 5570, 5571, 5573, 5574, 5575, 5618, 5624, 5643, 5657, 5711, 5717, 5726, 5733, 5768, 5769, 5808, 5848, 5878, 5907, 5945, 5967, 5992, 6020, 6029, 6044, 6049, 6053, 6064, 6117, 6136, 6137, 6405, 6424, 6438, 6449
\tl_set_eq:NN 945, 1140, 1141, 1267, 1271, 1409, 6091, 6097
\tl_to_str:n 5951, 6030
\tl_trim_spaces:N 239, 298
\tl_trim_spaces:n 930
\tl_use:N 12
token commands:
\token_to_str:N 159, 745, 1128
transparent commands:
transparent_node 2567
\ttdefault 1699
\typeout 6322

U

u16 3809
u32 3809
use commands:
\use:N . . . 1287, 5543, 5699, 6045, 6086
\use:n 6158
\use_i:nn 1402, 1403
\use_ii:nn 1404
\usepackage *4*, *25*

V

\verb *268*, *274*, *277*
VLIST 1729

W

warn commands:
warn_disabled_cjk_math 3732
WHATSIT 1729
write commands:
write_copies 5341

Y

\yahei 6342
\youyuan *7*, *28*, *270*, 1554, 6342

Change History

v0.0.1

General: Provide LuaCJK-listings, which keeps CJK characters in place inside a listings block and counts each of them as two columns; LuaCJK loads it automatically with listings. 1

Provide the CJK font family interface (\setCJKmainfont, \setCJKsansfont, \setCJKmonofont, \setCJKfamilyfont, \CJKfamily, \newCJKfontfamily, \CJKfontspec); an undeclared family warns once. 1

Refuse to load with luatexja. 1

apply_check_single: Provide CheckSingle and WidowPenalty, which stop a single CJK character from ending up alone on the last line of a paragraph. 185

apply_kinsoku: Provide PunctKinsoku line-breaking prohibitions: closing punctuation cannot start a line and opening punctuation cannot end a line. 152

bound_width: Provide PunctBoundWidth, which leaves only the given advance of a mark inside the line and hangs the rest into the margin, at the end of a line, at its start, and when the mark ends a box such as \mbox. 167

CJKfilltwosides: Provide the CJKfilltwosides environment, which justifies every line to a given width and places the box on the baseline as requested. It is tagged: a text block in vertical mode, part of the surrounding paragraph in horizontal mode. 237

\CJKunderanyline: Provide \CJKunderanyline and \CJKunderanysymbol for arbitrary rules and repeated symbols; \CJKunderanysymbol puts one symbol per em, centered in the cell, its depth measured from the baseline to the top of the symbol. 231

\CJKunderdblline: Provide \CJKunderdblline, \CJKunderwave, and \CJKxout; the wave and the slash stay in phase along a line, their patterns scale with the font size, a decoration does not open up its line, and it fills its span exactly. 226

copy_glyph: A character the font lacks leaves no placeholder: the .notdef or private-use glyph a font substitutes for it is removed. 126

\defaultCJKfontfeatures: Provide \defaultCJKfontfeatures and \addCJKfontfeatures. 92

handle_boundary: A mark keeps its break opportunity beside suspended text: the boundary glue at a mark has no width. . . 138

inherit_attributes: Glue, kerns, and penalties inserted by the package carry the attributes of the neighbouring node, so a decoration such as \CJKunderdot covers CJK/CJK

and CJK/Latin boundaries without a gap. 134
LuaCJK.cj_loaded: Provide CJLineBreak, which forbids a line break before the characters whose Unicode Line_Break property is CJ. 107
LuaCJK.define_punct_style: Provide KaiMingPunct, MiddlePunct, LongPunct, and NoBreakLongPunct, each with + and - forms, to edit the punctuation character sets. . . . 117
__LuaCJK_add_fallback:nn: Provide \setCJKfallbackfamilyfont and AutoFallBack for characters that the selected CJK font does not have. 94
__LuaCJK_load_lxgw:: Provide the LoadFandol option, which supplies Fandol fonts when the document declares none. 99
__LuaCJK_set_autofake:nn: Provide AutoFakeBold, AutoFakeSlant, EmboldenFactor, and SlantFactor; a family that sets BoldFont or ItalicFont is not emboldened or slanted. 61
__LuaCJK_set_punct_style:n: Provide the PunctStyle key with the quanjiao, plain, banjiao, kaiming, CCT, and hangmobanjiao styles. Under a style with a fixed width the advance of an opening or middle mark moves its ink with it. 52
__LuaCJK_settings_push:: Every setting travels with the characters: a snapshot of the options is recorded on the glyphs, so a group that ends before its paragraph decides how its own text is set. 68
__LuaCJK_setup_format_chars:: Ignore the zero-width format characters (U+200B, U+200C, U+200D, U+2060, U+180E) and the variation selectors at the input, so that they neither take glue nor separate two CJK characters. 97
__LuaCJK_update_font:: Provide PunctFamily, which sets a separate CJK family for the punctuation marks. 83
__LuaCJK_verbatim_env_end:: Suspend the adjustments inside \verb and the verbatim-like environments (verbatim, fancyvrb, listings); the Verb option (true, env, env+, false) controls what is suspended there. 74
\LuaCJKDeclareCharClass: Provide \LuaCJKDeclareCharClass, \LuaCJKDeclarePunctClass, and \LuaCJKResetCharClass to override the class of a code-point range. The classes carry the names of xeCJK (FullLeft, FullRight, HalfLeft, HalfRight, NormalSpace, CJStarter, CM, Boundary), the reset also restores the normal-spaced set, and the built-in table covers the CJK blocks of Unicode 15 and 16. 58
\LuaCJKDeclarePunctStyle: Provide \LuaCJKDeclarePunctStyle for a punctuation style of your own, with the keys compress-adjacent, adjacent-ratio, and keep-open-close. 61

\LuaCJKDeclareSubCJKBlock: Provide \LuaCJKDeclareSubCJKBlock with \LuaCJKCancelSubCJKBlock and \LuaCJKRestoreSubCJKBlock, so a range of characters can use its own font inside a family. . . . 55
\LuaCJKEditPunctStyle: Provide \LuaCJKEditPunctStyle to edit the punctuation sets of the current style, and AllowBreakBetweenPuncts. . . . 53
\LuaCJKemdescent: Provide \LuaCJKemdescent, the depth of the em box of the current font, so that an object of height 1em placed at that depth covers the one-em square of a CJK character in any font. 231
\LuaCJKenable: Provide \LuaCJKdisable and \LuaCJKenable to suspend the typographic adjustments while keeping CJK font selection; the state is recorded per node. 69
\LuaCJKnobreak: Provide \LuaCJKnobreak and the PunctWidth option; neither this prohibition nor that of \LuaCJKVerbAddon depends on PunctKinsoku, which controls only the punctuation rules. 69
\LuaCJKOffVerbAddon: Provide \LuaCJKVerbAddon and \LuaCJKOffVerbAddon, which make one CJK character as wide as two spaces of the current monospaced font and forbid breaks between CJK characters, so columns in code listings stay aligned. 70
\LuaCJKsetwidth: Provide \LuaCJKsetwidth and \LuaCJKsetkern for per-mark widths and per-pair gaps that override the selected punctuation style. 54
\LuaCJKShipoutHook: Provide \LuaCJKShipoutHook, which runs code through the standard shipout/before hook. 69
\LuaCJKversion: Provide \LuaCJKversion, which prints the version of the package. 49
\normalspacedchars: Provide \normalspacedchars so that slashes, backslashes, and hyphens take no glue on either side. 97
rewrite_fonts: Provide LatinPunct, which keeps the shared quotation marks, the middle dot, and the ellipses in the Latin font instead of the CJK font. 127
rewrite_math: Provide CJKmath and \setCJKmathfont for CJK characters in math mode, including nested math lists such as fractions. The family is registered as a math family, so a character in a script or a fraction keeps the size of that math style; with CJKmath off, a CJK character in math mode warns once. 162
spacing: Provide CJK/CJK and CJK/Latin spacing with CJKglue, CJKecglue, CJKspace, and xCJKecglue; a value keeps its stretch and shrink orders. A source space between two CJK characters becomes the inter-character glue, so the boundary keeps its break opportunity and its stretch, and no glue is inserted between two long punctuation marks. 137

\youyuan: Provide \songti, \heiti, \fangsong, \kaishu, \lishu, and \youyuan as shorthands for the families zhsong, zhhei, zhfs, zhkai, zhli, and zhyou. 96
v0.0.2
apply_kinsoku: AllowBreakBetweenPuncts no longer lets a closing mark start a line or an opening mark end one, and a mark inside a box takes part in the prohibitions. 152
\CJKfontspec: \CJKfontspec and \newCJKfontfamily take their options after the font name as well. 96
\CJKunderdblline: The double rule takes format like the other decorations, and a rule or a symbol that carries no format follows the color of the surrounding text. 226
hang_bounds: The hanging of a mark takes part in line breaking, so a line that ends at a closing mark or begins with an opening mark holds the mark the margin allows; the ink of that mark stands outside the line, and PunctBoundWidth decides how much of the advance stays inside. 182
__LuaCJK_add_fallback:nn: The fallback list is resolved again at every font selection, so the fallback follows the series, shape, and size of the text. 94
__LuaCJK_parse_font_options:n: Baseline shifts the glyphs of the font that declares it, so two CJK fonts whose baselines do not match can be set side by side. 56
__LuaCJK_reset_family:: Restore \CJKfamilydefault when \normalfont or \textnormal runs, so that the default font command overrides a family chosen earlier in the group; \CJKfamilydefault follows \CJKrmdefault until the document sets it. 81
\LuaCJKnobreak: \LuaCJKnobreak also forbids the break at a discretionary, so an explicit hyphen or an automatic hyphenation point inside the argument no longer breaks the line. 69
mark_blank: The hang of a mark is the blank its font carries, read from the outline, so the ink of a line-final or line-initial mark stands on the measure instead of half an em inside it; a file that cannot be read falls back to half the advance. 178
restore_source_spaces: The space in front of \textbf and its siblings is still the inter-character glue: LaTeX moves a space across the italic correction of a text font command, and the move leaves the glue with the subtype of a skip the document wrote. 144
\setCJKmathfont: A CJK character in a formula takes the face of the math version: the bold face of the family under \boldmath when the family has one, and the upright shape instead of the shape of the text. 95
v0.0.3
General: A mark that the CJK class shares with Latin, such as a curly quotation mark or the em dash, keeps its place inside an lstlisting block and counts as

two columns: the character table covers the CJK ranges of the Basic Multilingual Plane, both ends of each included. 1
The punctuation the two scripts share is a CJK character: the middle dot (U+00B7), the horizontal bar (U+2015), the two dot leader (U+2025), the hyphenation point (U+2027), and the midline ellipsis (U+22EF) are set in the CJK font, and LatinPunct keeps the ones it lists in the Latin font; the two dot leader joins a run of dashes and ellipses that may not be broken. 1
apply_check_single: A whatsit between two CJK characters no longer hides the break point from CheckSingle: a color change, a mark, and the other nodes that carry no text are passed over, so they no longer leave a single character on the last line. 185
apply_kinsoku: A dash or an ellipsis cannot start a line either: a break in front of a run of them is forbidden, so the run travels to the next line whole, while the break between the marks of a run stays the business of the NoBreakLongPunct set. 152
box_visible: A box that occupies no advance is passed over: \rlap, \llap and \clap overlay ink on the text around them, so they no longer take a CJKecglue on either side. 129
cff_box: A curve of four operands is read as one curve in both axes, so the blank of a mark whose charstring draws it with the alternating or the vertical curve operator is the blank the glyph has: a double opening quotation mark no longer stands in the margin by more than a tenth of a point. 173
The blank of a mark is measured from the charstring the glyph uses, so a glyph whose charstring calls a subroutine is measured in the subroutine rather than in the charstring the measurement began with. 173
\CJKfamily: \CJKfamily expands its argument, so a family can be given as \CJKrmdefault and friends. 95
CJKfilltwosides: The t and c positions of CJKfilltwosides place the box as \parbox does: t puts the baseline of its first line on the surrounding baseline, and c centres the box on the text axis instead of on the baseline. 237
\CJKunderdot: The emphasis dots sit under each CJK character, centred in the one-em cell of the character; Latin letters, digits, and spaces carry none, and the dots follow the characters when the inter-character glue stretches. 236
\CJKunderline: hidden takes the ink out of the text instead of putting it in a \phantom: the text keeps its line breaks, so a hidden paragraph breaks like the visible one instead of overflowing in one line. 216
A decoration keeps the width, the stretch, and the shrink of the text it covers, so a rule no longer changes the spacing of the line it sits in, skip no longer tightens a justified line, and a \kern inside the text no longer moves

the gap skip leaves at a punctuation mark. 216
\CJKunderwave: The wave is a smooth arch of one period per em, a quarter em above and below the underline; a decoration is divided into whole patterns once its final width is known and the pattern is drawn at that width, so the periods meet without a gap, the wave reaches the last character, and its period follows the stretch of the line. 226
The wave takes format like the other decorations: the pattern is stroked in that color, in the color of its text when the key is absent, and neither color reaches the text outside it. . . . 226
\CJKxout: The slash is divided into whole cells once its width is final, so the slashes cover the span evenly and the last one reaches its end. 230
The slash takes format like the wave, and follows the color of its text when the key is absent. 230
drop_orphans: A joiner or a variation selector that the font cannot use leaves nothing behind, and neither does a CJK character the CJK font does not have. 125
handle_boundary: A penalty that prohibits the break no longer removes the glue: the CJK–CJK and CJK–Latin spacing survives \nobreak, because the glue goes after the penalty, where the prohibition still covers it. A source space in front of the penalty is converted and the penalty moves in front of it. . . 138
hang_bounds: An opening mark that begins a paragraph, or a line the document forces a break at, hangs as well: its blank stands in the indentation or in the margin, and PunctBoundWidth decides how much of the advance stays inside the line there too. 182
__LuaCJK_ensure_family_default:: A document whose default CJK family is not declared is reported, and its text takes a family it did declare instead of none at all. 82
__LuaCJK_fv_patch:: The Verb option also reaches the inline \Verb and \UseVerb of fancyvrb, which never select \verbatim@font and are set as \verb sets them; the block commands \VerbatimInput and \UseVerbatim follow the environments instead. 73
__LuaCJK_load_lxgw:: LoadFandol gives the roman family FandolKai-Regular.otf as its italic face, so \emph and \itshape set the text in Kai instead of leaving it upright. . . . 99
LoadFandol is on by default, so a document that declares no CJK font is set in the four Fandol faces without asking for them. . 99
__LuaCJK_math_fallbacks_push:n: A CJK character in math mode takes a fallback font when the CJK font of the formula does not have it. 88
__LuaCJK_register_math_family:n: The three sizes of a CJK math family come from the text size of the formula, so a \text inside it no longer sets the other characters of the formula in a script size. 88

__LuaCJK_setup_format_chars:: The joiners (U+200C, U+200D) and the variation selectors are no longer removed at the input: they reach the font, so an emoji sequence is not split and a CJK variation sequence selects its glyph, and what the font leaves of them is dropped before the spacing pass, so two CJK characters still take no glue around them. 97
__LuaCJK_update_font:: PunctFamily shifts its marks by the Baseline of the family it names. 83
\LuaCJKOffVerbAddon: true and env as well; \LuaCJKOffVerbAddon restores the suspension that was in force. 70
\LuaCJKsetup: PunctWidth and PunctBoundWidth take 0pt as a length: the former gives every mark no advance and the latter hangs the whole of it, where a zero length used to act as false and return the mark to its style. 62
Process the package options after the helpers their key codes call, so that AutoFallBack, CJKmath, PunctFamily, and CJLineBreak act as package options as well. . 62
pre_shaping: The CJK font is applied to the characters before the text is shaped instead of to the glyph nodes after it: a combining mark attaches to its base, a variation sequence selects its glyph, and the font's own ligatures and locl forms appear. An older luaotfload keeps the previous behavior. . . 124
restore_source_spaces: \hfill after \textbf keeps its fill: only a glue of normal order counts as a source space, so the fill of the line is no longer turned into the inter-character glue. 144
\setCJKfamilyfont: \setCJKfamilyfont expands the name of the family it declares, so a family can be given as \CJKrmdefault and friends. 93
spacing: A superscript or subscript set with \textsuperscript or \textsubscript stays with the character in front of it: the shifted box takes no glue on either side, so no break is offered before it. 137
v0.0.4
General: A run of CJK characters inside an lstlisting block is one token of listings, so a emph or keywords entry that names a CJK word highlights it; a Latin character and a CJK mark both end the run. 1
apply_fixed_widths: A punctuation mark is given its fixed advance once: a box that holds the list is packed again, and the mark no longer takes a second compensating kern, so a fixed-width style no longer squeezes a table cell or a parbox. 157
\CJKfamilydefault: An untouched \CJKfamilydefault follows \familydefault at the start of the document, so a preamble that redefines \familydefault sets the main text in the matching CJK family. 91
The family \CJKfamilydefault names is selected when the document starts, so a preamble that redefines it sets the main

text in that family from the first line. 91
CJKfilltwosides: CJKfilltwosides leaves vertical mode as \parbox does, so the text after it joins the box on the same line instead of beginning a paragraph, and a space after the environment is ignored as xeCJK ignores it. . . 237
\CJKfontspec: \CJKfontspec no longer reads an option list after the font name, so a bracket that the text starts with is no longer swallowed. 96
\CJKunderdblline: format is resolved at every call, so a color that \colorlet redefines between two decorations is followed by the second one instead of keeping the color of the first. 226
\CJKunderdot: A CJK character inside a box gets its dot wherever the box sits in the span, so \mbox no longer drops the dot of a character in the middle of the text. 236
\CJKunderline: hidden hides a formula as it hides text: a character or a rule inside a box of the span carries no ink, so a fraction is hidden like a plain number and keeps its width, height, and depth. 216
\CJKunderwave: The wave is written one period at a time, so it covers the start and the end of a span that is not a whole number of periods, on a stretched line as well as on the last line of a paragraph. 226
\CJKxout: The slashes are written one cell at a time, so they cover the whole span. 230
\defaultCJKfontfeatures: \defaultCJKfontfeatures reaches the font of a sub-block and of a fallback font as well. . . 92
handle_boundary: A space in front of a color change keeps its advance: the whatsit a \textcolor inserts no longer hides the space from the boundary rules, so no CJKecglue or second CJKglue joins it. 138
A suspended run keeps the break opportunity beside it: the zero-width glue at its edge lets \verb end a line next to a CJK character while the spacing stays removed. 138
hang_bounds: A centered or flush-right paragraph does not hang the mark it begins with: the line does not start at the edge of the measure, so the shift would move the whole line instead of leaving the blank in a margin that is not there. 182
The first line of a list item hangs as well: the label box that the list writes in front of the item's text is passed over, so a mark that begins the item leaves its blank in the label's advance. . . 182
LuaCJK.hyphenate: A CJK character ends a word for hyphenation, so the line breaker no longer puts a hyphen beside a CJK character, and the boundary keeps the CJKecglue that the two sides give it. 190
LuaCJK.is_cjk: A code point below U+0100 in a font that is not Unicode-encoded is a position in that font's encoding, not a character, so a symbol font's slot is no longer read as a

CJK character: pifont's slot 0xB7, the circled 2, keeps its own font and prints. 108
With LatinPunct on, the midline ellipsis U+22EF stays in the Latin font like the other shared marks that key lists. 108
__LuaCJK_family_font:n: The font of a CJK family is resolved in the Unicode encoding: a document that sets its Latin text in an 8-bit encoding with fontenc keeps its CJK text. 75
__LuaCJK_parse_font_options:n: Baseline belongs to the declaration that writes it, so a fallback font and a family that name the same font keep their own offsets. 56
\LuaCJKDeclareCharClass: HalfLeft and HalfRight declare a half-width ASCII bracket: it keeps its advance, does not hang, and hugs the text inside the pair, where they used to declare a full-width opening or closing mark. 58
\LuaCJKnobreak: \LuaCJKnobreak forbids the break at a glue the document writes inside its argument, so a \quad or an \hspace there no longer starts a line. 69
PUNCT_CLASS: The ASCII brackets are punctuation marks of their own: a half-width opening bracket no longer ends a line and a half-width closing bracket no longer starts one, the pair hugs the text between it, and a compression style leaves the bracket's own advance alone. . 111
The punctuation table covers the marks xeCJK classes as opening or closing, so the percent sign U+FF05 no longer starts a line and the currency symbols no longer end one; the halfwidth and compatibility forms and the three hyphen-like marks follow the same class. 111
v0.1.0
apply_fixed_widths: A ratio style never cuts into the ink: a mark keeps its ink and the smaller of its two blanks, so the fullwidth percent sign and the fullwidth currency signs keep their advance under banjiao and no longer overlap the character after them. 157
attached_box: A box that holds nothing on its own baseline, only raised or lowered boxes, is attached like a shifted box: the mark of \footnote and \raisebox therefore take no glue on either side, as the box \textsuperscript builds already did. 129
A framed box is an object of the line and not a mark: \fbox, \framebox and \fcolorbox lower a vertical box around their content, so they take the CJKecglue on both sides like any other box, while the mark of \raisebox or a \footnote still takes none. 129
boundary_settings: A CJK–Latin boundary takes the settings of the CJK character it touches, so CJKecglue follows the size of the CJK text: a Latin run set with \scriptsize or \large keeps the space of the surrounding text on both of its sides. 106
boundary_spec: A formula that sets one opening or closing

delimiter alone is that bracket: the CJK–Latin glue stands outside the pair and not between the delimiter and the text it encloses, so the angle brackets of a \meta argument hug their Chinese characters. . 140
cff_box: The ink of a glyph is measured at the extremum of its curves rather than at their control points, and the alternating curve operators read the abscissa of their third point from the operand that carries it, so the blank a mark hangs is the blank its outline paints. 173
CJKfilltwosides: A \footnote inside CJKfilltwosides is set at the bottom of the box and numbered with the box's own counter, instead of being lost with the insert that a plain vertical box cannot hold. 237
The body of CJKfilltwosides is set as a minipage, so a paragraph after a paragraph break starts at the left edge of the box and is filled like the first one, and the leading between the lines is the leading of the document instead of whatever an enclosing alignment left behind. 237
\CJKfontspec: \CJKfontspec switches the text to its font even when PunctFamily names a family, which used to replace the family name the command was about to select. 96
\CJKunderanysymbol: The cell of a repeated symbol starts at the left edge of the decorated span, so \CJKunderanysymbol keeps the symbol of a single character whose span begins in the middle of a cell; \CJKunderanyline and the symbol of \CJKunderdot and \CJKunderwave follow the same rule. 235
class_of_node: A box takes the class of the text nearest its edge, so a kern, a spacing glue, or a rule inside it no longer hides that text: \mbox around an italic \textit keeps the CJKecglue beside its last letter, a box that holds no text takes the glue on both sides, and a filling glue still leaves the edge empty, which keeps the titles of a table of contents aligned. 128
A box the edge holds does not hide the text behind it when that box has no class of its own, and a paragraph line is read past its local_par marker: \parbox takes the CJKecglue at the edge its first character reaches, while its last edge stays empty, because \parfillskip does not let the text reach it. . . 128
A vertical box reads its edges by the same rule, so the rule that underlines a \underline no longer hides the text above it: the underlined Latin run takes the CJKecglue on both of its sides, like the run a \mbox or \CJKunderline holds. 128
handle_boundary: A source space beside a box that occupies no advance belongs to the boundary of the two characters the box stands between: the space in front of the box becomes the inter-character glue, so an \mbox between two CJK characters takes no CJKglue of its own, and two spaces on

either side of it are one space. 138
A suspended run takes the glue its two classes give at its edge: \verb beside a CJK character is spaced as the same characters in a text font are, while the material itself is left alone. . . . 138
hang_bounds: The last line of a centered or flush-right paragraph hangs the mark it ends with as well, so every line of it stops at the same edge instead of the last one standing the width of that blank further in. 182
inherit_common_attributes: A node inserted at a CJK–Latin boundary keeps only the attributes both characters carry, so a decoration ends at the last character it covers: the CJKecglue behind \CJKunderline's argument no longer takes the rule past it. . . 135
insert_boundary_glue: A half-width closing bracket keeps the CJKecglue between itself and the CJK character that follows it, so the outside of a pair of brackets is spaced on both sides while the pair still hugs the text it encloses. 142
The glue stands in front of the background a colour box draws: the highlight of \colorbox covers the box it was asked to colour and not the space between the two characters, so it stops the same distance from the text on both sides. 142
__LuaCJK_load_lxgw:: LoadLXGW replaces LoadFandol and supplies the LXGW family when the document declares no CJK font. 99
__LuaCJK_parse_font_options:n: Baseline moves the characters of the family in force where they are written, and the marks of the family PunctFamily names, so a family selected inside a group moves only that group and two declarations of one font keep their values in the same paragraph. 56
\LuaCJKDeclarePunctClass: The punctuation group latin-close can be declared and removed like the others, so the spacing of the ASCII closing marks is a setting of the document and not only a default. 59
The two- and three-em dashes U+2E3A and U+2E3B are CJK characters of the dash group, so a font that joins a pair of em dashes into one glyph keeps the prohibition of NoBreakLongPunct. 59
\LuaCJKnobreak: \LuaCJKnobreak forbids the breaks inside its argument alone: a break at either edge of the argument is the business of the text around it, and the glue the package inserts behind the last character no longer carries the prohibition. 69
mark_blanks: The blank of a glyph on each side is read from its outline, so the gap \LuaCJKsetkern gives and the amount a compression takes are the blanks the font carries. . . . 179
\normalspacedchars: The initial list holds the two characters it documents, the slash and the hyphen: the list is read as a sequence of characters, so the

commas that separated them used to be normal-spaced as well, and so did the space between two of them. A backslash is not in it, because beside a CJK character it usually begins a printed command name, which takes the glue on both of its sides like any other Latin character. 97
pair_kern: The gap between two adjacent marks is the blank between their ink: the length \LuaCJKsetkern gives is that blank, so 0pt makes the two marks meet instead of widening the gap, and a compression takes at most the blank the compressed mark carries on its right. 138
v0.1.1
General: With breaklines, a run of CJK characters inside an lstlisting block is set one character at a time, so a line breaks between two of them; the run is still one token, so emph and keywords still highlight the whole word. 1
apply_check_single: A box at the end of a paragraph no longer hides its characters from CheckSingle: the character a \underline or an \mbox ends with is the last character of the paragraph, and the characters a box holds count as the one unit they travel as. 185
apply_kinsoku: CJLineBreak reads the settings of the character that would start the line, so a group that selects strict no longer forbids a break in front of a small kana written outside it. 152
A word space between two punctuation marks is a break point again: the two are not adjacent, so the rule that keeps two adjacent marks together does not read them as a pair. . . 152
\CJKunderdot: A decoration nested inside \CJKunderdot keeps the dots: the span of the dots is measured with the pair of an inner decoration stepped over, and the dots of a call no longer depend on which decoration the characters inside it belong to. . 236
\CJKunderline: A decoration draws with the settings of its own call, so a skip, a subtract, or a hidden reaches a span that another decoration of the same kind built the type for, a decoration nested inside another one keeps its own inset, and a key that the command does not use is accepted and ignored instead of being handed to lua-ul. 216
handle_source_space: A source space between two long punctuation marks adds no advance and no stretch, so justification cannot pull a run of dashes or ellipses apart. 146
hang_bounds: A mark inside a box hangs only when nothing inked stands between it and the edge of that box, so \fbox and \framebox stop at the alignment edge instead of moving their whole frame past it, while \mbox around a mark keeps its blank in the margin. 182
A paragraph that begins with a box hangs the opening mark inside it as well, so \mbox around the mark leaves its

blank in the indentation, as a line of plain text does. 182
A space in front of a mark that begins a line the document broke keeps the discard of the line head: the rule that carries the hang stands where the line begins and not at the break, so the line is no longer set the width of that space further in. 182
__LuaCJK_warn_punct_family:: A family that PunctFamily names but no declaration provides warns once, so the fallback of the punctuation to the text font is no longer silent. 85
\LuaCJKDeclareSubCJKBlock: A declared sub-block adds its ranges to the CJK character class, so a block declared outside the built-in CJK ranges (the circled numbers U+2460 to U+2473, say) reaches the font the family gives it instead of staying in the Latin font. 55
\LuaCJKenable: \LuaCJKdisable and \LuaCJKenable end the value of the attribute they set, so a digit that follows either command stays in the text instead of joining the value. . . . 69
\LuaCJKfntefsetup: Provide \LuaCJKfntefsetup with the key commute, which packs the inks of a group of nested decorations into the band their own positions make, so the nesting order decides which ink is below which instead of every ink keeping the height its keys give. 216
\LuaCJKnobreak: Every \LuaCJKnobreak marks its argument as a span of its own, so two arguments beside each other keep the line break between them, while an argument written inside another one still belongs to the span around it. 69
The span identifier of \LuaCJKnobreak ends its value, so an argument that begins with a digit keeps it. 69
\LuaCJKsetup: PunctFamily expands the family name it is given, so a family can be named by \CJKsfdefault or another macro that holds the name. . . . 62
pre_shaping: The Baseline of a family lifts a CJK character that is already set in the family's own font, which is what a document that selects that font itself with \fontspec gets. . . . 124