Hyphenation languages in LuaTEX 0.90

Hans Hagen

In LuaTEX you can define up to 16,383 separate languages, and words can be up to 256 characters long. The language is stored with each character. You can set `\firstvalidlanguage` (a new variable) to, for instance, 1 and thereby make language 0 an ignored hyphenation language. Because the language is stored in the glyph nodes this is an efficient way to disable hyphenation locally.

The new primitive `\hyphenationmin` can be set to specify the minimum length of a word considered for hyphenation. This value is stored with the (current) language and applies to the whole paragraph. Because `\lefthyphenmin` and `\righthyphenmin` are stored with the glyphs you can temporarily change them. The `\uchyph` value is also saved in the actual nodes, therefore its handling is different from TEX82: changes to `\uchyph` become effective immediately, not at the end of the current partial paragraph.

LuaTEX now uses the new language-specific variables `\prehyphenchar` and `\posthyphenchar` when creating implicit discretionaries, instead of TEX82's `\hyphenchar`, and new variables `\preexhyphenchar` and `\postexhyphenchar` (also language-specific) for explicit discretionaries, instead of TEX82's empty discretionary.

Typeset boxes now always have their language information embedded in the nodes themselves, so there is no longer a dependency on the surrounding language settings. In TEX82, a mid-paragraph statement like `\unhbox0` would process the box using the current paragraph language unless there was a `\setlanguage` issued inside the box. In LuaTEX all language variables are already frozen.

In traditional TEX, hyphenation is driven by the so-called `\lccode` table. In LuaTEX we made this dependency less strong. Several strategies are possible. When you do nothing, the currently-used `\lccode`'s are still the default when loading patterns, setting exceptions or hyphenating a list.

When you set `\savinghyphcodes` to a value larger than zero the current set of `\lccode`s will be saved with the language but in a dedicated namespace reflecting hyphenation-justification codes. In this case changing a `\lccode` afterwards has no effect. Instead of `\lccode` you can use `\hjcode`. These are per-language and when set take precedence over the shared `\lccode`s. So, LuaTEX doesn't store `\lccode`s per language but has a dedicated hyphenation-justification code instead. You can change these values at any time with `\hjcode‘a=‘a`.

This change is global which makes sense if you keep in mind that the moment when hyphenation happens is (normally) when the paragraph or a horizontal box is constructed. If `\savinghyphcodes` was zero when the language was initialized you start out with nothing, otherwise you already have a set. Beware: the `\hjcode` values are always saved in the format, independent of the value of `\savinghyphcodes` when the format is dumped.

The value of the two counters related to hyphenation, `\hyphenpenalty` and `\exhyphenpenalty`, are now stored in the discretionary nodes. This permits a local overload when explicit `\discretionary` commands are used. The implementation is downward compatible but permits control for special situations.

MetaPost arrowhead variants

Alan Braslau, Hans Hagen

Some colleagues have complained that the arrowheads in MetaPost graphics are too blunt, and that they would like to see a more stylish arrowhead. Perhaps they do not appreciate how MetaPost produces arrowheads that actually follow a curved path? Still, we realized that one can easily modify the arrowhead macro to produce variants to satisfy everyone while remaining simple and elegant.

We settled on a backwards-compatible solution of adding two new global, internal variables to the existing `ahlength` and `ahangle`: `ahvariant` and `ahdimple`. With the default `ahvariant:=0`, one gets the traditional MetaPost arrowhead. A nonzero value will give a more stylized arrowhead that uses the value of `ahdimple`, a unitless fraction of `ahlength`, by default 0.2, to create a dimple at the base. The `variant:=1` uses `...` to give a rounded dimple or "ear" and `variant:=2` uses `--` to create a barb. Finally, a sort of "broadhead" can be produced by making `ahdimple` negative. (We also made an efficiency change in PDF drawing that leads to an improvement when drawing arrowheads.)

This change is now part of MetaFun and ConTEXt and can be easily included as MetaPost macros. Examples follow (scaled for *TUGboat*).

```
pickup pencircle scaled 2mm;
ahlength := 8mm;drawarrow fullcircle scaled 2cm;
ahvariant := 1; drawarrow fullcircle scaled 4cm;
ahdimple := .5; drawarrow fullcircle scaled 6cm;
ahangle := 60;  drawarrow fullcircle scaled 8cm;
```