OVERLOAD PROTECTION

the downside of macros

context 2021 meeting

Primitives

A TEX engine comes with a whole set of primitive operations for:

- accessing internal variables
- defining macros
- controlling expansion
- constructing boxes
- finalizing pages
- defining characters (text and math)
- inserting kerns, glue and penalties
- defining fonts
- dealing with languages (hyphenation)
- testing properties
- manipulating tokens
- managing inserts
- handling marks
- grouping
- mathematics
- tracing

Macros

- Macros are commands defined by the user and/or a macro package.
- They can overload a primitive which then can confuse the whole machinery.
- A macro package can alias primitives for instance `\relax` can be replaced by `\foo_relax` after `\let \foo_relax \relax`.
- That only when (at definition time) the _ is a letter. By using such a character some protection against overload is provided.
- In ConTEXt we often use(d) aliases like `\normalrelax` but of course these can also be overloaded.
- We only overload a very few primitives, for instance `\language`.
- Users who overload primitives are ‘on their own’ and ‘without support’.
- An easy way to protect yourself is using `\CamelCase` names.

Overload protection

- The LuaMetaTEX engine has overload protection built in for the TEX engine as well as provides means to do that for MetaPost.
- In LMTX all commands have been tagged accordingly (which was quite some work).
- Processing `s-system-macros.mkxl` gives an overview.
- Overload protection is off by default but can be turned on:

```
\enabledirectives[overloadmode=warning]
\enabledirectives[overloadmode=error]
```

- I myself always run with the error variant and make sure that the manuals obey the rules.
- Modules and/or styles (and in a few cases the core code) can cheat and use:

```
\pushoverloadmode
    ........................
    ........................
\popoverloadmode
```

Details

- Traditional TeX has a few so called prefixes: `\global`, `\outer`, `\long`, and type `\immediate`.
- The ε-TeX engine adds `\protected` (because we already had that in ConTeXt we use what we (also already) had: `\unexpanded`).
- In LuaTeX we can force macros to be always long, something that we do in MkIV (as in MkII).
- In LuaMetaTeX the `\outer` and `\long` prefixes have been dropped (they are ignored).
- In LuaMetaTeX the `\protected` prefix acts like in other engines but protection is implemented more naturally.
- In addition LuaMetaTeX has new prefixes: `\frozen`, `\permanent`, `\immutable`, `\mutable`, `\noaligned`, `\instance`, `\untraced`, `\tolerant`, `\overloaded`, `\aliased`, `\immediate` and an experimental `\semiprotected`,
- Some prefixes end up as properties of macros, some influence scanning (for instance in alignments and when calling Lua functions). There is no noticeable runtime overhead.
- The `\meaningfull` primitive is like `\meaning` but also reports properties set by prefixes; there is also `\meaningless`.

Prefixes

Regular definitions:

- `\global`: defines a macro or sets a register value out of scope.
- `\outer`: is used to issue a warning when a macro defined as such was used nested (just ignored in LuaMetaTEX).
- `\long`: triggers a warning when an argument of a macro contains a `\par` equivalent (just ignored in LuaMetaTEX).
- `\protected`: makes a macro unexpandable (being itself) in an `\edef` equivalent situation (where it can get out of hands).
- `\semiprotected`: is like `\protected` but the property is ignored when `\semiexpanded` is applied.

Special case:

- `\immediate`: tells a backend primitive to come into action immediately instead of creating a delayed action (via a whatsit node). In LuaMetaTEX we have no built-in backend so there is signals a Lua interface function this property.

Scanning related:

- `\noaligned`: tags a macro as valid peek ahead macro when scanning alignments; normally that only makes sense for `\protected` macros. This permits cleaner macros in for instance table mechanisms (no unexpected expansion side effects).
- `\untraced`: this flag makes a macro report itself as primitive in traces which is sometimes nicer that presenting a user with some confusing meaning.
- `\tolerant`: a prefix that makes the macro argument parser accept all kind of new argument parsing options and continue when delimited arguments fail. This makes macros with optional arguments produce less noise when they are traced but more important, it makes for cleaner low level interfaces.

Overload protection (primitives are protected against overloads by default):

- `\aliased`: create a reference (using `\let`) that also inherits the properties.
- `\permanent`: sets a flag that makes a macro (or definition) immune for redefinition.
- `\frozen`: prevents overloading but one can bypass that with some more effort.
- `\immutable`: makes a (normally variable definition) fixed, for instance constants.
- `\mutable`: a flag showing that this macro or definition can be used for anything (so the macro package also has to assume that).
- `\instance`: just a flag that can be used to signal that a macro (or definition) is an instance of a more general concept.
- `\overloaded`: bypass a frozen flag (property).

Show some examples in the source code and editor.