PROGRAMMING

the way ConTEXt is set up

context 2021 meeting

Levels

When you look at ConTEXt bottom–up (engine–interface) you will notice:

1. **primitives:** this is what the engine comes with
2. **infrastructure:** basic management of data structures
3. **helpers:** macros that hide complexity
4. **subsystems:** collections of macros that implement functionality
5. **mechanisms:** these combine various subsystems
6. **modules:** extra functionality (uses 1–5)
7. **styles:** handling sources and layout (uses 4–6)

Users normally see ConTEXt top–down (usage–hacking).

Styles

- These are prebuilt solutions for common as well as rare situations.
- The system comes with some styles: the `s-*` files.
- Right from the start the idea was that you get some reasonable default.
- And if you want more you stepwise define your style as you go.
- It really is part of the game: exploration.
- Solving your problem is a nice challenge.
- If you want a completely predefined setup, shop somewhere else.

Modules

- A lot of functionality is built in.
- This helps to keep the system consistent.
- We could cheat and ship thousands of few–liner styles but don't do that.
- There are a few mechanisms that don't really fit into the core, so these are implemented as modules that do fit into the interface: the `m-*` and `x-*` files.
- Users can build and share their solutions which has resulted in some third party modules: the `t-*` files.
- For (a few, often old) private files I use `p-*` name scheme.

Mechanisms

- These are combinations of subsystems but often they cannot really be distinguished.
- Examples are notes, that combine notations, lists, references, descriptions etc.

Subsystems

- This is what users see and can configure
- Most are (conceptually) rather old but evolved over time. There are no fundamental differences between MkIV and LMTX, but the later is hopefully a bit cleaner.
- Examples are fonts, languages, color, structure (sectioning, lists, constructions, itemgroups, references), spacing, graphics, bibliographies, positioning, numbering and layout.
- More hidden are the backend, export and xml interfaces.
- Some have subsystems themselves, like widgets that relate to a specific backend.
- There are often dependencies between subsystems which makes that it's not really a hierarchy. A more strict separation would demand much more overhead.

Helpers

- These provide basic programming help.
- Examples are macros for comparing things, loops, list processing, argument handling.
- But more abstract box manipulations also fits in here.
- Some subsystems, like xml and bibliographies provide more specific low level helpers.

Infrastructure

- The engine provides counters, dimension and other registers that need to be managed in order to avoid clashes in usage.
- Many of the helpers, subsystems and mechanisms fall back on common rather low level functions (Lua) and macros (using primitives).

Primitives

- This is what the engine provides: the built-in commands and features.
- In addition to the visible primitives there are Lua interfaces and these permit adding extra primitives.
- In LuaMetaTEX we have the core TEX set but a few were dropped because we don't have a backend and a different io subsystem (so they have to be emulated).
- We also have some of the ε-TEX primitives and very few of the pdfTEX ones but I now consider for instance expansion and protrusion extensions to be kind of ε-TEX.
- There are additional LuaTEX primitives but some were dropped, again because of the backend, so we emulate some, and also because some were experimental.
- There are quite some new primitives and existing mechanisms have been extended, cleaned up and (hopefully) improved.

The shift

- There have always been complaints about TEX as a language (what makes me wonder why those who complain use it.)
- Although there are some extensions to the language in ε-TEX, follow-ups have not really succeeded in this area.
- At some point I decided that code in the categories 1–4 cold benefit from extensions.
- That also meant that we use less of the low helpers. It makes the code look a bit more TEX.
- It also means less clutter, in code as well in tracing. Often the code becomes simpler too.
- The idea is that TEX becomes a bit more a programming language.
- Of course it takes away the “Watch me, I can do real dirty TEX hacking!” brawling.
- It also can take away some of the complaints.
- And it definitely adds some fun.

During the week we show some of the implementation (in Visual Studio) and examples of applications. We also write a small extension (the dk *unit)*