Liberate TEX: Progress on Building a New TEX-Language Interpreter

Doug McKenna
Mathemaesthetics, Inc.
Boulder, Colorado

TUG — 2014
The \TeX\ Ecosystem Seems Fractured and Forked

- There’s \TeX
- \ldots \text{ or } \epsilon-\TeX
- \ldots \text{ or } \text{pdf}\TeX
- \ldots \text{ or } \text{pdfLaTeX}
- \ldots \text{ or LaTeX or plain } \TeX\text{ or } \text{ConTeXt} \text{ (multiple formats)}
- \ldots \text{ or LaTeX3 or XeLaTeX or pdfXeLaTeX}
- \ldots \text{ or LuaTeX}
- \ldots \text{ or Omega (dead) or } \ldots
- \ldots \text{ or T1 encodings or OpenType vs. TFM or } \ldots

It’s complex, messy, confusing. Can it be unified? Simplified?

Not without a complete re-write of the core \TeX\ engine.
Philip K. Dick’s *The Minority Report*

A “precog” in Philip K. Dick’s short story *The Minority Report* is a human with a special ESP power. From Wikipedia:

“The precogs sit in a room that is perpetually in half-darkness, constantly talking nonsense to themselves that is incoherent until it is analyzed by a computer and converted into predictions of the future. This information is assembled by the computer into the form of symbols before being transcribed onto conventional punch cards that are ejected into various coded slots. … [P]recogs are kept in rigid position by metal bands, clamps and wiring, that keep them attached to special high-backed chairs. Their physical needs are taken care of automatically.”
Replace **predictions of the future** in the foregoing quote with **high-quality automated typesetting**. The engine’s source code

- Is focused on, and fabulously accomplished at, one thing
- Depended upon by an important segment of society
- But in other respects, almost decrepit, foreign, useless
- Lives in rigid stasis, writ in literate stone, topically changed
- Is protected by and strapped in a **WEB**, intubated with tangled shell scripts, barely alive except by the grace of Web2C life-support software, nursed by makefile minions, attended by wizards, and—once in a blue moon—a Grand Wizard
- Like a prehistoric software insect, frozen in amber and time
- Is not a normal piece of modern, living, adaptable software.
- “Being literature” and “being software” have different goals
Rewriting T\(\text{E}X\) from Scratch — JSBox (for now)

\(\text{T\(\text{E}X\)}\)'s source code is what it is: a large set of interconnected algorithms and data structures, relieved of as much redundancy in time and space as possible. It is a platonic creature of its time and its author. Leave it be, but let's liberate its algorithms and services:

- JSBox is a personal project started in 2009 . . . and ongoing
- JSBox is not T\(\text{E}X\): JSBox is a T\(\text{E}X\)-language engine
- Automated translation of T\(\text{E}X\)'s source code doesn't suffice
- Being upwardly compatible with existing T\(\text{E}X\) code is hard
- JSBox wastes some space and time: inherent redundancies reduce code fragility and enhance adaptability
- As simple, understandable, usable, portable as possible
- Tries to solve problems that T\(\text{E}X\)'s source code, its greater ecosystem, and its users (including me) suffer from
Solution:

- **JSBox** is a library for a client program to use
- The library instantiates one or more \TeX \text{ language interpreter} “object”s in the memory space of its client program
- Each interpreter can be client- or job-configurable at run-time: \TeX82, \varepsilon-\TeX, \exists\TeX, JSBox, or other feature levels
- The client program mediates between each interpreter and both the system and the user
- JSBox is 100% system-agnostic: the client performs all system-related services, memory allocation, file I/O, etc.
- Client monitors, suppresses, simulates, or otherwise manages all I/O or memory allocation; interpreters are “sandbox-able”
- Interpreter exists independent of whether a job is done or not
Solution:

- JSBox is written in pedal-to-the-metal, portable C
- Compilable for ILP32 and LP64 architectures (ILP64 soon)
- No dependencies on any other software or libraries
- About 100,000 lines of code, half of it comment(ary)
- Does not use literate programming tools (CWEB, etc.)
- Instead, literate commenting using literac conventions
- Currently implemented as one C file, two header files
- Build time for edit-compile-link-run testing is a few seconds
- Client programs can be written in C, C++, Objective-C, Python, Swift, etc.; whatever can link to and call a C function.
#3 — Formats

- Dumped formats are an unnecessary optimization, due to Problem #1
- They are modes that harm users, and complicate tech support
- The language itself should require/permit a document to declare the format it relies on, just like packages
- `%!TEX TS-program = pdflatex` or similar is an ugly, band-aid comment hack
- Design seems based on 1970s-era core dump hack (see, e.g., Adventure game state restoration on a PDP-20)
- Formats should not incorporate precompiled language hyphenation databases, which should be job- or locale-based
Solution:

- JSBox compiles plain.tex in .008 second (at 2.8GHz)
- And it reads and compiles \LaTeX{}'s 12000 lines of pure \TeX{} code (with over 30 TFM metric files) in .06 second
- A job as an object is divorced from the language interpreter’s existence and initialization level
- As an interpreter initialization level, a format need only be read once (under the hood—the document doesn’t care)
- When a job is done, interpreter state should return to its pre-job state; i.e., format definitions are still there
- Namespaces for formats seem a much better solution
- JSBox will avoid implementing \texttt{\textbackslash dump} unless proven necessary
#4 — 8-bit Character Codes

- JSBox internally traffics in full 21-bit Unicode code points
- \text{T\textit{E}}\textit{X} algorithms, data structures re-implemented for Unicode
- Input can be a mixed stream of 1-, 2-, or 4-byte integers, client-supplied from memory (a text buffer) or from a file
- Input can be UTF-8 (it’s a transport format, not an encoding)
- Client can use fast, native file system calls
- After conversion to internal Unicode, the first 256 8-bit code points can be mapped to any other 21-bit Unicode code points
- Mappings are client- or job-configurable at run-time
- All strings internally stored as UTF-8
- All output in human-readable text is UTF-8
- Client has final say and can convert UTF-8 to anything else
#5 — Too Few Character Categories

Unicode supports over 1,000,000 characters (code points)

- JSBox (very generously) allocates 8 bits for CatCodes (syntactic character categories)
- First 16 are, of course, the usual \TeX{} syntactic code values
- All 240 others, with one exception (16 \texttt{?}), are reserved
- No current \TeX{} code assigns CatCode values above 15
- Therefore, new CatCodes can be upwardly compatible
- And gated by run-time feature level
- New values must be agreed-upon by entire \TeX{} community
Solution:

- CatCode 16: namespace separator character
- For instance, a '.', a '@', or any Unicode code point
- JSBox’s scanner recognizes namespace separator characters as a means of drilling down into nested namespaces to resolve macro names and deliver a single token to higher levels of interpretation
- For example,
  \[\text{plain.obeylines}\]
  or \[\text{latex.fancyvrb.VerbatimFootnotes}\]
  etc.
- Unresolved forward or circular references are handled on the fly
Namespaces can be named and created using, e.g.,
\namespacedef\mydict

Pushed onto or popped from scanner’s current context stack:
\beginnamespace\mydict
...
\endnamespace

Like font names—invoke the name to push and make current:
\latex
\verb"foo"
\endnamespace
\verb"foo" % \verb no longer resolvable

TEX converts each page (as it becomes full) to DVI or PDF, then ships it, so as to recycle precious memory.

But memory is a lot more plentiful 30 years later.

This also works against two- or multi-page optimizations.

Solution:

- JSBox logically ships each page, with all Output nodes executed
- But can also keep all final “shipped” page data structures, with \specials retained, in memory
- Page data structures not recycled until next job begins
- Any (random) page is later exportable to client as needed
- DVI and PDF steps can be skipped to export directly to client
- Client then draws into a scrolling view (an eBook reader)
Tex only traces about 75% of what it's doing. But all hidden state creates invariably confusing modes.

- At least 1/3 of the code in JSBox is devoted to full tracing
- No generic tracing; primitives trace themselves
- Indented execution contexts; lines are assumed arbitrarily long
- Indentation for subordinate lines of tracing information
- Vertical whitespace between classes of log file output
- Commands that are interrupted (to recursively expand or collect arguments, by an error message) are marked as such and re-trace themselves when done
- Alignment stages when constructing tables are traced
- Conditional tests shown more clearly
- File positions where files are not found can be traced.
Other Debugging Aids

- Ability to trace exactly one invocation of one macro
- Character data presented in multiple value formats
- Original names and types when restoring group context values
- Better skip glue origination labeling
- Many design decisions made with log searchability in mind
- For example, all box nodes given unique (per job) IDs
- Integral \showfont OpenType or TFM font metric dumps
- JSBox \debugger primitive enables \TeX source to create a breakpoint in interpreter’s execution loop
- Data structure examination with IDE debugger now possible
#9 — Error Reporting

\TeX’s error messages are hard to understand, formatted in a way that violates the user’s view of the world, two-level, and sometimes unnecessarily confusing.

Solution:

- No generic error reporters (e.g., misleading \texttt{\textbackslash badness} error)
- All error messages in JSBox have been completely rewritten
- All errors provide as much information as possible up front; no “failure-to-communicate” secondary reports
- Token being executed, from a compiled token list, or from file, is highlighted on a line user will recognize
- Structured error/warning messages can be packaged for client’s GUI use outside of log file
- Optional compatibility warnings for run-time feature levels
Solution:

- JSBox parses OpenType font metrics, tables, features, and whatever else is needed to measure glyphs (very fast, too)
- 'maxp', 'head', 'name', 'cmap', 'hhea', 'O/2' 'htmx' 'post', 'GPOS', 'GSUB', 'kern', 'TeX ', 'MATH' tables
- Font data structures designed to be union of TFM and OpenType information
- Subroutines to handle, e.g., ligatures or extensions, can be made font-type-specific, within one job
- Many sub-problems left to solve; \TeX primitives to incorporate; font feature support; etc.
#11 — Hyphenation Databases

- U.S. English database is pre-compiled into JSBox
- Hyphenation data should not be part of a format, pre-compiled or not; usually locale-dependent
- Nor job- nor interpreter-specific
- Multiple languages in one job are not very common
- Databases should be dynamically loaded by library as needed, and shared among instantiated interpreters
- With interpreter- or job-specific overrides/updates as needed
- JSBox keeps separate “tries” for separate language codes
- Some time-optimization for tries, but (currently) not space
- Therefore . . . no artificial limit on number of languages
TEX uses an artificially halved fixed-point arithmetic dynamic range, so that any two scaled integers can be added without worrying about overflow. But multiple sums can still overflow, with wraparound garbage results.

Solution:

- All fixed-point measures in JSBox are 32-bit [16:16] format
- When recompiled for ILP64 architecture, [48:16] format
- No hacks that use fixed-point bits as special flag values
- Calculations check for overflow or boundary conditions, including most-negative twos-complement number
- Overflows don’t wrap; they saturate to most positive, or most negative, fixed-point number
- Box content summations in the average case need no overflow checking, but are checked again in the exceptionally large case
Current State of JSBox

- JSBox functionally conforms with Knuth’s "trip.tex" test
- All measurements the same, all data structures “the same”
- Does not produce the same log file, so a diff won’t work
- This 200+ page log file shows what “trip.tex” does
- But … JSBox is not yet ready for prime-time
- Need to get it to typeset my own LaTeX documents first
- Need to understand what kpathsea does, and how to avoid the messes it enables
- Some remaining $\varepsilon$-TeX primitives are still unimplemented
- Plenty of OpenType layout work to do
- Giant balance between simplicity and generality

“Congratulations on a massive achievement” — Don Knuth
Demo