TUG 2018 program

Friday July 20	8:00 am	*registration*	
	8:55 am	Paulo Ney de Souza, UC Berkeley & BooksInBytes	*Opening*
	9:00 am	Roberto Ierusalimschy, PUC-Rio & Lua team	*The making of Lua*
	9:45 am	Eduardo Ochs, UFF	*Dednat6: An extensible (semi-)preprocessor for LuaLaTeX that understands diagrams in ASCII art*
	10:20 am	*break*	
	10:45 am	Mico Loretan, Zurich, Switzerland	*Selective ligature suppression with the* `selnolig` *package*
	11:15 am	Joseph Wright, LaTeX Project	*Fly me to the moon: (La)TeX testing (and more) using Lua*
	11:50 am	Paulo Cereda, University of São Paulo	*From parrot 1.0 to phoenix 4.0: 6 years of* `arara`*, the beginning of a new era*
	12:25 pm	*lunch*	
	1:45 pm	Will Robertson, University of Adelaide	*Creating teaching material with LaTeXML for the Canvas Learning Management System*
	2:25 pm	Ross Moore, Macquarie University	*Authoring accessible 'Tagged PDF' documents using LaTeX*
	3:00 pm	*break*	
	3:20 pm	Sandro Coriasco, Anna Capietto, Università di Torino	*An automated method based on LaTeX for the realization of accessible PDF documents containing formulae*
	3:55 pm	Doris Behrendt, DANTE e.V.	*The General Data Protection Regulation (GDPR) in the European Union*
	4:30 pm	*TUG meeting*	
	4:30 pm	*Workshop: Accessibility challenges in LaTeX*	
Saturday July 21	8:55 am	*announcements*	
	9:00 am	Frank Mittelbach, LaTeX Project	*A quarter century of* `doc`
	9:35 am	Joseph Wright	*Through the looking glass, and what Joseph found there*
	10:10 am	*break*	
	10:30 am	Boris Veytsman, George Mason Univ. & Chan Zuckerberg Initiative	*Stubborn leaders six years later*
	11:05 am	Joseph Wright	`siunitx`*: Past, present and future*
	11:40 am	Frank Mittelbach	*Compatibility in the world of LaTeX*
	12:40 pm	*lunch*	
	2:00 pm	Joachim Heinze, Springer Verlag	*The unchanged changing world of mathematical publishing*
	2:35 pm	Tom Hejda, Charles University Prague	`yoin` — *Yet another package for automation of journal typesetting*
	3:10 pm	*break*	
	3:30 pm	Paulo Ney de Souza	*Minimizing LaTeX files — First steps on journal automated processing*
	4:05 pm	Boris Veytsman	*R+knitr workshop (*`tug.org/tug2018/workshops.html`*)*
Sunday July 22	8:55 am	*announcements*	
	9:00 am	S.K. Venkatesan and TNQ Lab	*WaTeX (WYSIWYG and LaTeX) and Hegelian contradictions in classical mathematics*
	9:35 am	Susanne Raab, Paulo Cereda*	*A short introduction to the TikZducks package*
	10:10 am	*break*	
	10:30 am	Jaeyoung Choi, Soongsil University	*FreeType_MF_Module: A module for using METAFONT directly inside FreeType's rasterizer*
	11:05 pm	*robertson-fonts*	
	11:40 am	*lunch*	
	1:00 pm	*bus to Sugarloaf*	
	≈ 5:30 pm	*back at hotel*	

* = presenter

Doris Behrendt

The General Data Protection Regulation (GDPR) in the European Union

On 25 May 2018 the GDPR was applied in the EU. In my position as treasurer of the German TEX user group DANTE e.V. I studied this regulation from the DANTE perspective and will talk about some aspects of this regulation, which are concerning us.

As some of you probably know, a lot of Europeans — including myself — are somewhat delicate about data processing and privacy. While the industry complains about the GDPR being a monster of bureaucracy, there are also some quite interesting legal bearings that come with it, e.g. it will also apply "to the processing of personal data of data subjects who are in the Union by a controller or processor not established in the Union, where the processing activities are related to ... the offering of ... services, irrespective of whether a payment of the data subject is required, to such data subjects in the Union ...".

This should be interesting especially to companies that are not based in the EU but are handling data of EU citizens, and by GDPR Article 83 (5) not complying could become expensive: "Infringements ... shall ... be subject to administrative fines up to 20,000,000 EUR, or in the case of an undertaking, up to 4% of the total worldwide annual turnover of the preceding financial year, whichever is higher ...".

You can imagine that this could become very interesting when the next Facebook or similar data scandal comes up.

Paulo Cereda

From parrot 1.0 to phoenix 4.0: 6 years of `arara`*, the beginning of a new era*

For the uninitiated, `arara` is a TEX automation tool based on rules and directives. It determines its actions from metadata in the source code, rather than relying on indirect resources, such as log file analysis. The tool does not employ any guesswork, so you get exactly what you asked for. Historically, the very first version of `arara` was released on 2012. The feedback was very positive. However, it didn't take much time for bug reports and feature proposals to arrive. The author then decided to take a bold step and rewrite the tool entirely from scratch, fixing bugs and including new features on his own spare time. It took a bit longer than expected due to a concurrent never-ending thesis writing process, but the result is now version 4.0 of `arara`, the most exciting release of the tool's entire life cycle.

In this talk, the author will present an overview of `arara`'s greatest new features, including friendly and helpful messages, improved command line layouts, better logging, multiline directives, dry-run mode, file hashing, UI elements and conditionals. However, there is a lot more to the tool than meets the eye, so be prepared for a mind-blowing live demonstration (beware, there might be sound involved as well!). Expect lots of use case scenarios, and of course, bird-related jokes.

Jaeyoung Choi

FreeType_MF_Module: A module for using METAFONT directly inside FreeType's rasterizer

METAFONT is a font description language which generates bitmap fonts for use by the TEX system, printer drivers, and related programs. One advantage of METAFONT over outline fonts is its capability to produce various font styles by changing parameter values defined in its font specification file. Another major advantage of using METAFONT is that it produces various font styles like bold, italic, and bold-italic from one source file, unlike outline fonts, which require development of a separate font file for each style in one font family. These advantages can be applied to design CJK (Chinese-Japanese-Korean) fonts, which require significant time and cost because of the large number of characters used in these scripts. However, in order to use METAFONT in current font systems, users need to convert its bitmaps into a corresponding outline font. Furthermore, font rendering engines like FreeType do not support METAFONT.

In this paper, an MF Helping Module for FreeType rasterizer is proposed. This 'FreeType_MF_Module' enables direct usage of METAFONT just like any other outline and bitmap font support in FreeType rasterizer. Users of METAFONT don't need to pre-convert METAFONT into its corresponding outline font as the module automatically does this. Furthermore, the module allows users to easily and directly generate various font styles from METAFONT by changing parameter values.

Joachim Heinze

The unchanged changing world of mathematical publishing

1. A very short overview of the history of mathematical publishing with some Springer examples is given. *Numerische Mathematik* was the first of all Springer-Nature journals ever over all disciplines to go online in 1994.

2. The change of the world of publishing: generating (scientists), composing (publishers and scientists) and disseminating (librarians and publishers) mathematical content in electronically form. TEX and "online visibility" are the buzzwords here.

3. Open access for all mathematical content? "New" initiatives like "Overlay Journals", based on arXiv, are briefly discussed, as well as the more recent Sci-Hub and ResearchGate initiatives.

4. Keep track of what has been published and cited. MathSciNet and zbMATH, the two big math review journals, in comparison to other initiatives, like Google Scholar/xs.glgoo.com, Scopus, and Web of Science. A new initative from China? MathSciDoc.

5. Recent developments in the dissemination of scientific information are discussed. Social media (Scholarly Collaboration Networks (SCN)) in scientific communication and some new initiatives such as "Sharedit" and "SciGraph" are briefly reflected upon. Artificial intelligence and some hope for the future will close the presentation.

Tom Hejda

`yoin` — *Yet another package for automation of journal typesetting*

A new LaTeX package will be presented that allows combining journal, conference and similar papers into issues. The most important premises the package is built upon are (1) the papers themselves are independent documents to the extent that even different compilers can be used for different papers, and (2) the papers' page numbering is automated and there are tools for communicating metadata between the whole issue and the papers.

Please note that a preliminary version of the package will be presented and help from the community will very likely be sought at the conference.

Mico Loretan

Selective ligature suppression with the `selnolig` *package*

TeX has long provided straightforward methods for creating typographic ligatures. Until recently, though, suppressing inappropriate ligatures selectively could only be achieved by applying mark-up by hand to a document. `selnolig`, a LuaLaTeX package, provides the machinery to perform selective ligature suppression in an automated way that requires minimal user involvement. The package also provides sets of ligature suppression rules for English and German language documents. The talk provides an overview of the package's design philosophy and main features, discusses some of its current limitations, and gives the outlook for further developments.

Frank Mittelbach

A quarter century of `doc`

In this talk I will re-examine my poor attempts at Literate Programming and how they have shaped (for the better or worse) the LaTeX world in the past decades.

It's about time to rethink some of the concepts invented back then — but can we still evolve?

Frank Mittelbach

Compatibility in the world of LaTeX

In this talk I take a look at the major disruptions that have rocked the LaTeX world in the past decades and how we handled them, covering some of the resulting consequences.

In the latest part of this saga a rollback concept for the LaTeX kernel was introduced (around 2015). Providing this feature allowed us to make corrections to the software (which more or less didn't happen for nearly two decades) while continuing to maintain backward compatibility to the highest degree.

I will give some explanation on how we have now extended this concept to the world of packages and classes which was not covered initially. As the classes and the extension packages have different requirements compared to the kernel, the approach is different (and simplified). This should make it easy for package developers to apply it to their packages and authors to use when necessary.

Ross Moore

Authoring accessible 'Tagged PDF' documents using LaTeX

Several ISO standards have emerged for what should be contained in PDF documents, to support applications such as 'archivability' (PDF/A) and 'accessibility' (PDF/UA). These involve the concept of 'tagging', both of content and structure, so that smart reader/browser-like software can adjust the view presented to a human reader, perhaps afflicted with some physical disability. In this talk we will look at a range of documents which are fully conformant with these modern standards, mostly containing at least some mathematical content, created directly in LaTeX. The examples are available on the author's website (`http://web.science.mq.edu.au/ ross/TaggedPDF/`).

The desirability of producing documents this way will discussed, along with aspects of how much extra work is required of the author. Also on the above website is a 'five-year plan' proposal how to modify the production of LaTeX-based scientific publications to adopt such methods. This will involve cooperation between academic publishers and a TUG working group. The proposal PDF (`http://web.science.mq.edu.au/ ross/TaggedPDF/ PDF-standards-v2.pdf`) is itself produced to be fully accessible, complying with both PDF/UA-1 and PDF/A-2a standards.

Eduardo Ochs

Dednat6: An extensible (semi-)preprocessor for LuaLaTeX that understands diagrams in ASCII art

(La)TeX treats lines starting with "`%`" as comments, and ignores them. This means that we can put anything we want in these "`%`" lines, even code to be processed by other programs besides TeX.

In this talk we will describe a "semi-preprocessor", called `dednat6`, that makes blocks of lines starting with "`%L`" be executed as Lua code, treats blocks of lines starting with "`%:`" as 2D representations of derivation trees, and treats blocks of lines starting with "`%D`" as diagrams in which a 2D representation specifies where the nodes are to be placed and a stack-based language inspired by Forth is used to connect these nodes with arrows.

A predecessor of `dednat6`, called `dednat4`, was a preprocessor in the usual sense: running "`dednat4.lua foo.tex`" on a shell would convert the trees and diagrams in "`%:`" and "`%D`"-blocks in `foo.tex` to "`\def`"s that LaTeX can understand, and would put these "`\def`"s in a file `foo.dnt`; we had to put in `foo.tex` an "`\input "foo.dnt"`" that would load those definitions. `Dednat6` does something almost equivalent to that, but it uses LuaLaTeX to avoid the need for an external preprocessor and for an auxiliary "`.dnt`" file. Here is how; the workflow is unusual, so let's see it in detail.

Put a line
```
\directlua{dofile("loaddednat6.lua")}
```
in a file `bar.tex`. When we run "`lualatex bar.tex`" that line loads the `dednat6` library, initializes the global variable `tf` in the Lua interpreter with a

`TexFile` object, and sets `tf.nline=1` to indicate that nothing in `bar.tex` has been processed with dednat6 yet. A (low-level) command like `\directlua{processlines(200, 300)}` in `bar.tex` would "process the lines 200 to 300 in `bar.tex` with dednat6", which means to take all the blocks of "`%L`"-lines, "`%:`"-lines, and "`%D`"-lines between the lines 200 to 300 in `bar.tex`, run them in the adequate interpreters, and then send the resulting LaTeX code — usually "`\def`"s — to the `latex` interpreter. The high-level macro "`\pu`" runs "`\directlua(processuntil{tex.inputlineno})`", that runs `processlines` on the lines between `tf.nline=1` and the line where the current "`\pu`" is, and advances `tf.nline` — i.e., it processes with `dednat6` the lines in the current file between the previous "`\pu`" and the current one.

The strings "`%L`", "`%:`", and "`%D`" are called "heads" in `dednat6`, and it's easy to add support for new heads; this can even be done in a "`%L`" block.

Note that with `dednat4` all the "`\def`"s had to be loaded at once; in `dednat6` idioms like "`{\pu ...}`", "`$\pu ...$`", and "`$$\pu ...$$`" can be used to make the "`\def`"s between the last "`\pu`" and the current one be local.

Susanne Raab, Paulo Cereda (presenter)

A short introduction to the TikZducks package

The Ti*k*Zducks package is a little package (and Ti*k*Z library) that allows one to easily add rubber duck images to LaTeX documents. As implied by the name, the package uses the graphics language Ti*k*Z to create the ducks based on simple geometrical shapes and paths.

Besides the ducks themselves, the package also provides a large (and ever growing) assortment of accessories which can be added to the ducks. Under the hood of the Ti*k*Zducks package a simple key–value interface is used to access and customise all the available options.

After a short introduction to the package and its options, some examples will be presented in this talk, including how the ducks can be personalised. One of the examples will show a step-by-step guide how a duck can be designed based on a photo of a person.

Will Robertson

Creating teaching material with LaTeXML for the Canvas Learning Management System

In this presentation I will outline the system by which I produce PDF and HTML versions of course material for the honours project students in the School of Mechanical Engineering, The University of Adelaide.

This course material is broad and relatively dynamic in that it needs both frequent and periodic updates, and there is a soft need to have it available in a single document PDF and a hyperlinked HTML version. There are a number of tools to perform such a task, and LaTeXML was chosen for its robustness and relative simplicity. Nonetheless, the processing phase does involve some regexps to clean up the resulting HTML, which is not ideal from a maintenance perspective.

On the back end, this project could not have been accomplished without the API provided by the Learning Management System that we use, Canvas by Instructure. The web API allows HTML pages to be updated from the command line as well as PDF files to be automatically uploaded.

This system allows me to have a single source for the documentation for the course and makes updates almost entirely friction-free. While still cobbled together from a number of technologies (largely `curl` and shell scripts), it provides an interface that could be expanded for more general use.

In the future, as well as re-writing the code in Lua for cross-platform functionality, I also plan to overcome the problems involving use of embedded graphics with text, and mathematical content in general.

Will Robertson

Unicode fonts with `fontspec` *and* `unicode-math`

While the fundamentals of both the `fontspec` and `unicode-math` packages have stayed the same, these packages have undergone a significant amount of development behind the scenes. While many users won't be interested in the technical details, there are a number of feature additions that deserve broader discussion.

In this presentation I will cover the basics of these packages and best practices for using them, specifically including more recent features that users may not yet have seen. I will also try to give an overview of some technical details to focus on `expl3` package development and lessons learned.

Boris Veytsman

Stubborn leaders six years later

After six years the journal *Res Philosophica* changed the style of its table of contents. The new design requires the dotted line with the page number to follow the last line of the article title rather than the first one. The old design was described in a *TUGboat* article (33:3, pp. 316–318, 2012, `tug.org/TUGboat/tb33-3/tb105veytsman-leaders.pdf`). We use this occasion to revisit the old code, discuss the new one and the fact that deceptively similar designs require completely different code.

S.K. Venkatesan and TNQ Lab

WaTeX (WYSIWYG and LaTeX) and Hegelian contradictions in classical mathematics

Contradictions exist in life and are the mill through which the world evolves. We will demonstrate a strange hybrid that has evolved from the contradiction between LaTeX and WYSIWYG, WaTeX, a new LaTeX editor based on KaTeX. However, we will also be presenting many other common varieties of Hegelian contradictions that exist in classical mathematics which don't lead to blow-ups in classical logic and which don't require a new paraconsistency logic proposed by the Peruvian philosopher Francisco Miró

Quesada, which quantum computing (QPU) may demand.

Joseph Wright
Through the looking glass, and what Joseph found there

The LaTeX3 programming language, `expl3`, has grown over the past decade to form a strong and stable environment for solving problems in TeX. A key aim is to grow this work to cover a wider range of areas. In recent work, the team have been building on the existing code, and in particular the expandable FPU, to develop approaches to color, drawing and image support. In this talk, I will look at why this work is useful, what models we can work from and where the work has taken us so far.

Joseph Wright
`siunitx`*: Past, present and future*

Over the past decade, `siunitx` has become established as the major package for typesetting physical quantities in LaTeX. Here, I will look at the background to the package, and how it's developed over the years. I'll also lay out plans for the future: where are we going for version 3, and why is that important for users.

Joseph Wright
Fly me to the moon: (LA)TeX testing (and more) using Lua

Testing has been important to the LaTeX team since its inception, and over the years a sophisticated set of test files have been created for the kernel. Methods for running the tests have varied over the past quarter-century, following changes in the way the team work.

In recent years, the availability of Lua as a scripting language in all TeX systems has meant it has become the natural choice to support this work. With this as a driver, the team have developed `l3build` for running tests automatically. Building on the core work, `l3build` has grown to provide a powerful approach to releasing packages (and the LaTeX kernel) reliably.

Here, I'll look at the background of our testing approach, before showing how and why Lua works for us here.

From parrot 1.0 to phoenix 4.0

Paulo Roberto Massa Cereda

Abstract

This article covers a bit of history behind arara, the cool TeX automation tool, from the earlier stages of development to the new 4.0 series. We also highlight some noteworthy features of our tool.

1 Introduction

Writing software is easy. Writing *good* software is extremely difficult. I was working on a Catholic songbook with 1200+ songs and several indices and cross-references. The compilation steps required to achieve the final result were getting out of hand.

At some point, I realized I *knew* all the steps I had to reproduce beforehand, I only had to find a way to automate them! Inspired by the way compilers work (i.e., read a source file, ignore all comments and process the rest), I could exploit TeX comments to include special indications on what to do on the document. Since engines do ignore comments, no side effects would arise, at least document-wise.

It was a cold afternoon. I sat in front of my computer and decided to work on this new tool. It was a matter of time to reach preliminary yet promising results. I mentioned this effort in the chat room of the TeX community at StackExchange and some friends asked me to make a public release out of it, as other users could benefit from this new tool.

However, a name was needed for the tool. In the chat room, we used to have a lot of fun with palindromes (specially palindromic reputations in arbitrary bases), so I took that aspect as inspiration. Then I thought of a very beautiful, colourful bird of the Brazilian fauna: the macaw, or as we like to call it, the arara. The name was immediately adopted!

Once the name was chosen, I needed a logo. Since I am a Fedora Linux user, I was always a fan of their default typeface, which is quite round! The choice was made: the humble arara tool became arara. My life was about to change.

2 A bit of history

A lot of things happened since version 1.0, released in 2012, to the new version 4.0, released in 2018. This section presents a bit of history of arara, including challenges in each version.

2.1 The first version

There is a famous quote along the lines of “if at first you do not succeed, call it version 1.0”. The first version of arara was also the first public release, dated April 2012. Nothing much was there, besides the core concepts that exist until this very date:

- *Rules:* a rule is a formal description of how arara handles a certain task. It tells the tool how to do something.
- *Directives:* A directive is a special comment inserted in the source file in which you indicate how arara should behave.

Back then, we could write directives in our document and have the tool process them as expected, like the following example:

```
% arara: pdftex
Hello world!
\bye
```

Amusingly, the first version only offered a log output as an additional feature. There was no verbose mode at all. The log file was a gathering of streams (error and output) from the sequence of commands specified through directives. And that was it.

2.2 The second version

The first version had a serious drawback: compilation feedback was not in real time and, consequently, no user input was allowed. For the second version, real time feedback was introduced when the tool was executed in verbose mode.

```
$ arara -v mydoc.tex
...
[real time feedback]
...
```

Two other features were included in this version: a flag to set the execution time out, in milliseconds, as a means to prevent a potentially infinite execution, and a special variable in the rule context for handling cross-platform operations.

2.3 The third version

So far, arara was only a tiny project with a very restricted user base. However, for version 3.0, a qualitative goal was reached: the tool became international, with localised messages in English, Brazilian Portuguese, German, Italian, Spanish, French, Turkish and Russian. Besides, new features such as configuration file support and rule methods brought arara to new heights. As a direct consequence, the lines of code virtually doubled from previous releases.

```
$ arara --help -L es
...
 -h,--help     imprime el mensaje de ayuda
 -l,--log      genera el registro de la salida
...
```

When the counter stopped at version 3.0, Brent Longborough, Marco Daniel and I decided it was time for arara to graduate and finally be released in

TEX Live. Then things really changed in my life. The tool was a success! Given the worldwide coverage of that specific TEX distribution, arara silently became part of the daily typographic tool belt of many users. But then, the inevitable happened: a lot of bugs emerged from the dark depths of my humble code.

2.4 Critical and blocker bugs

Suddenly, several questions about arara were posted in the TEX community at StackExchange and I was not able to provide a consistent, definitive answer for many of them! It was very tricky to track the bugs to their sources, and some of them were really nasty. For instance, a simple scenario of a file with spaces in the name was more than enough to make the poor tool cry for help for apparently no reason:

```
$ arara "My PhD thesis.tex"
```

Likewise, the issues page of the project repository hosted at GitHub had a plethora of reports, and little I could do about them. I delved into the code of third party libraries, but the root of all evil seemed to lie in my own sources.

2.5 Nightingale

In all seriousness, I was about to give up. My code was not awful, but there were a couple of critical and blocking bugs. Something very drastic had to be done in order to put arara back on track. Then, proceeding on faith, I decided to rewrite the tool entirely from scratch. In order to achieve this goal, I created a sandbox and started working on the new code. And this new project got a proper name: nightingale.

It was the right thing to do. Nicola Talbot helped me with the new version, writing code, fixing bugs and suggesting new features. She was writing a book about LATEX for administrative work at the time and was extensively using arara in the code examples. Her writing indirectly became my writing as well, as I progressively improved the code and added new features to match her suggestions.

2.6 The fourth version

At some point, nightingale had to say farewell and gave most of its features to the bigger, older bird in the nest. It is worth mentioning that nightingale still lives in my repository at GitHub for those who are bold to try it. From 1500+ lines of code in version 3.0, arara 4.0 tripled that number: a whooping 4500+ lines of code! And the most important: all critical and blocking bugs were completely fixed.

However, although the code was ready for production, the user manual was far from being finished. In fact, the documentation had to be written entirely from scratch. Then another saga had started: find proper time and effort to document a great yet complex tool in all details, from user to developer perspectives.

It took me a lot of dedication to write the user manual and try to cover as much detail as possible for every feature, old and new, and the tool itself. Some of the internals had to be changed, so more explanations were needed. Documenting a tool is almost as difficult as writing code for it!

3 New features

This section highlights some noteworthy features found in the new version 4.0 of arara. For additional information, please refer to our user manual.

3.1 REPL work flow

In version 4.0, arara employs a REPL work flow for rules and directives. In previous versions, directives were extracted, their corresponding rules were analyzed, commands were built and added to a queue before any proper execution or evaluation. I decided to change this work flow, so now arara evaluates each rule on demand, i.e., there is no *a priori* checking. A rule will always reflect the current state, including potential side effects from previously executed rules.

3.2 Multiline directives

Sometimes, directives can span several columns of a line, particularly the ones with several parameters. From arara 4.0 on, we can split a directive into multiple lines by using the `arara: -->` mark on each line which should compose the directive. We call it a multiline directive. Let us see an example:

```
% arara: pdflatex: {
% arara: --> shell: yes,
% arara: --> synctex: yes
% arara: --> }
```

It is important to observe that there is no need for them to be in contiguous lines, i.e., provided that the syntax for parametrized directives hold for the line composition, lines can be distributed all over the code. In fact, the log file (when enabled) will contain a list of all line numbers that compose a directive.

3.3 Directive conditionals

arara 4.0 provides logical expressions, written in the MVEL language, and special operators processed at runtime in order to determine whether and how a directive should be processed. This feature is named *directive conditional*, or simply *conditional* as an abbreviation. The following list describes all conditional operators available in the directive context.

- `if`: The associated MVEL expression is evaluated beforehand, and the directive is interpreted if, and only if, the result of such evaluation is true. This directive, when the conditional holds true, is executed at most once.

```
% arara: pdflatex if missing('pdf')
% arara: --> || changed('tex')
```

- `until`: The directive is interpreted the first time, then the associated MVEL expression evaluation is done. While the result holds false, the directive is interpreted again and again. There is no guarantee of proper halting.

```
% arara: pdflatex until !found('log',
% arara: --> 'undefined references')
```

- `unless`: Technically an inverted `if` conditional, the associated MVEL expression is evaluated beforehand, and the directive is interpreted if, and only if, the result is false. This directive, when the conditional holds false, is executed at most once.

```
% arara: pdflatex unless unchanged('tex')
% arara: --> && exists('pdf')
```

- `while`: The associated MVEL expression is evaluated beforehand, the directive is interpreted if, and only if, the result is true, and the process is repeated while the result still holds true. There is no guarantee of proper halting.

```
% arara: pdflatex while missing('pdf')
% arara: --> || found('log', 'undefined
% arara: --> references')
```

Although there are no conceptual guarantees for proper halting of unbounded loops, we have provided a technical solution for potentially infinite iterations: arara has a predefined maximum number of loops. The default value is set to 10, but it can be overridden either in the configuration file or with a command line flag.

3.4 Directive extraction only in the header

The `--header` command line option changes the mechanics of how arara extracts the directives from the code. The tool always reads the entire file and extracts every single directive found throughout the code. However, by activating this switch, arara will extract all directives from the beginning of the file until it reaches a line that is not empty and is not a comment (hence the option name). Consider the following example:

```
% arara: pdftex
Hello world.
\bye
% arara: pdftex
```

When running arara without the `--header` option, two directives will be extracted (the ones found in lines 1 and 4). However, if executed with this switch, the tool will only extract one directive (from line 1), as it will stop the extraction process as soon as it reaches line 2.

3.5 Dry-run execution

The `--dry-run` command line option makes arara go through all the motions of running tasks and subtasks, but with no actual calls. It is a very useful feature for testing the sequence of underlying system commands to be performed on a file.

```
[DR] (PDFLaTeX) PDFLaTeX engine
----------------------------------------
Authors: Marco Daniel, Paulo Cereda
About to run: [ pdflatex, hello.tex ]
```

Note that by the rule, authors are displayed (so they can be blamed in case anything goes wrong), as well as the system command to be executed. It is an interesting approach to see everything that will happen to your document and in which order. It is important to observe, though, that conditionals are not evaluated in this mode.

3.6 Local configuration files

From version 4.0 on, arara provides support for local configuration files. In this approach, the configuration file should be located in the working directory associated with the current execution. This directory can also be interpreted as the one relative to the processed file. This approach offers a project-based solution for complex work flows, e.g., a thesis or a book. However, arara must be executed within the working directory, or the local configuration file lookup will fail. Observe that this approach has the highest lookup priority, which means that it will always supersede a global configuration.

3.7 File hashing

arara 4.0 features four methods for file hashing in the rule and directive scopes, presented as follows. The file base name refers to the file name without the associated extension.

- `changed(extension)`: checks if the file base name concatenated with the provided extension has changed its checksum from last verification.
- `changed(file)`: the very same idea as the previous method, but with a proper Java `File` object instead.
- `unchanged(extension)`: checks if the file base name concatenated with the provided extension is unchanged from last verification. It is the opposite of the `changed(...)` method.
- `unchanged(file)`: the very same idea as the previous method, but with a proper Java `File` object instead.

The value is stored in a database file named `arara.xml` as a pair containing the full path of the provided file and its corresponding CRC-32 hash (the database is created if missing). If the entry already exists, the value is updated, or created otherwise.

3.8 Dialog boxes

A *dialog box* is a graphical control element, typically a small window, that communicates information to the user and prompts them for a response. arara 4.0 provides UI methods related to such interactions. As good practice, make sure to provide descriptive messages to be placed in dialog boxes in order to ease and enhance the user experience.

3.9 Session

Rules are designed under the *encapsulation* notion, such that direct access to the internal workings of such structures is restricted. However, as a means of supporting framework awareness, arara provides a mechanism for data sharing across rule contexts, implemented as a `Session` object. In practical terms, this particular object is a global, persistent map composed of keys and values available throughout the entire execution.

3.10 Redesigned user interface

For arara 4.0, we redesigned the interface in order to look more pleasant to the eye; after all, we work with TEX and friends. Please note that the output was deli‘berately truncated to respect the column width.

```
  __ _ _ __ __ _ _ __ __ _
 / _‘ | ’__/ _‘ | ’__/ _‘ |
| (_| | | | (_| | | | (_| |
 \__,_|_|  \__,_|_|  \__,_|

Processing ’doc.tex’ (size: 307 bytes, last
modified: 05/29/2018 08:57:30), please wait.

(PDFLaTeX) PDFLaTeX engine ............ SUCCESS
(PDFLaTeX) PDFLaTeX engine ............ SUCCESS

Total: 1.45 seconds
```

First of all, we have the nice application logo, displayed using ASCII art. The entire layout is based on monospaced font spacing, usually used in terminal prompts. Hopefully you follow the conventional use of a monospaced font in your terminal, otherwise the visual effect will not be so pleasant. First and foremost, arara displays details about the file being processed, including size and modification status:

```
Processing ’doc.tex’ (size: 307 bytes, last
modified: 05/29/2018 08:57:30), please wait.
```

The list of tasks was also redesigned to be fully justified, and each entry displays both task and subtask names (the former being displayed enclosed in parentheses), besides of course the usual execution result:

```
(PDFLaTeX) PDFLaTeX engine ............ SUCCESS
(PDFLaTeX) PDFLaTeX engine ............ SUCCESS
```

If a task fails, arara will halt the entire execution at once and immediately report back to the user. This is an example of how a failed task looks like:

```
(PDFLaTeX) PDFLaTeX engine ............ FAILURE
```

Also, observe that our tool displays the execution time before terminating, in seconds. The execution time has a very simple precision, as it is meant to be easily readable, and should not be considered for command profiling.

```
Total: 1.45 seconds
```

The tool has two execution modes: *silent*, which is the default, and *verbose*, which prints as much information about tasks as possible:

- When in silent mode, arara will simply display the task and subtask names, as well as the execution result. Nothing more is added to the output.
- When executed in verbose mode, arara will display the underlying system command output as well, when applied. In version 4.0 of our tool, this mode was also entirely redesigned in order to avoid unnecessary clutter, so it would be easier to spot each task.

It is important to observe that, when in verbose mode, arara can offer proper interaction if the system command requires user intervention. However, when in silent mode, the tool will simply discard this requirement and the command will almost surely fail.

4 The future

Now that arara 4.0 is officially released and already available in CTAN and TEX Live, it is time to plan the future. Our repository already has suggestions for new features and improvements. The work on arara 5.0 has begun! If you have any feedback about our tool, please drop us a note.

Also, if you believe your custom rule is comprehensive enough and deserves to be in the official pack, please contact us. We will be more than happy to discuss the inclusion of your rule in forthcoming updates. Happy TEXing with arara!

◇ Paulo Roberto Massa Cereda
Analândia, São Paulo, Brazil
`cereda.paulo (at) gmail.com`

FreeType_MF_Module: A module for using METAFONT directly inside the FreeType rasterizer

Jaeyoung Choi, Ammar Ul Hassan and Geunho Jeong

Abstract

METAFONT is a font description language which generates bitmap fonts for the use by TEX system, printer drivers, and related programs. One advantage of METAFONT over outline font is its capability to produce different font styles by changing various parameter values defined in its font specification file. Other major advantage of using METAFONT is that it produces various font styles like bold, italic, and bold-italic from one source file unlike outline fonts, which require development of a separate font file for each style in one font family. These advantages can be applied to design CJK (Chinese-Japanese-Korean) fonts which require significant time and cost because of the large number of characters used in Hangeul (Korean character) and Hanja (Chinese character). However, to use METAFONT in current font systems, users need to convert it into its corresponding outline font. Furthermore, font rendering engines like FreeType doesn't support METAFONT.

In this paper, *FreeType_MF_Module* for FreeType rasterizer is proposed. The proposed module enables a direct usage of METAFONT just like any other outline and bitmap font support in FreeType rasterizer. Users of METAFONT don't need to pre-convert METAFONT into its corresponding outline font as FreeType_MF_Module automatically performs this. Furthermore, FreeType_MF_Module allows the user to easily generate various font styles from one METAFONT source file by changing parameter values.

1 Introduction

As of today's, information society, the traditional pen and paper usage for communication between people are swiftly replaced by computers and mobile devices. Text has become an effective source for gathering information and means of communication between people. Although people use smart devices commonly these days with effective resources like media and sound, text plays the key role of interaction between user and device. Text is composed of characters, and these characters are physically build from specific font files in digital environment system.

Fonts are the graphical representation of text in a specific style and size. These fonts are mainly categorized in two types; outline fonts and bitmap fonts. Outline fonts are the most popular fonts for producing high-quality output used in digital environment system. However, for creating a new font style for outline font, font designers have to design a new font with extensive cost and time. This recreation of font files for each variant of font can be very hectic for font designers especially in case of CJK fonts which require designing of thousands individual letters one by one. Since CJK fonts have a lot more letters and complex shapes compared to Roman fonts, it often takes more than a year to design CJK fonts.

To overcome this disadvantage of outline fonts mentioned above a programmable font, METAFONT was developed. METAFONT is a programming language introduced by D. E. Knuth [1] that generates TeX-oriented bitmap fonts. It allows users to generate various font styles easily. METAFONT file is different from outline font file. It consists of functions for drawing characters and has parameters for different font styles. By changing the parameter values defined in its font specification file, various font styles can be easily generated. Therefore, a variety of font variants can be generated from one METAFONT source file.

However, users are unable to use METAFONT on their PCs because current font engines like FreeType [2] doesn't provide any direct support of METAFONT. Unlike standard bitmap and outline fonts, METAFONT is expressed as a source code that is compiled to generate fonts. To use METAFONT in a general font engine like FreeType rasterizer users have to convert METAFONT into its corresponding outline font. As it was developed in 1980s, the PC hardware was not fast enough to provide a real-time conversion of METAFONT into its corresponding outline font.

Current PC hardware, however, is fast enough to provide a real-time conversion of METAFONT into its corresponding outline font. If METAFONT is used directly in font engines like FreeType, then users can easily generate variety of font styles. METAFONT can also be used to produce various font styles like bold, italic, and bold-italic with one source file unlike outline fonts, which require development of a separate font file for each style in a font family. These advantages can also be applied to design CJK fonts to produce various high-quality font styles because, compared to alphabetic scripts, CJK characters are both complicated in shape and expressed by combinations of radicals [3].

In this paper, a FreeType_MF_Module for FreeType rasterizer is proposed. The proposed module enables a direct use of METAFONT in FreeType rasterizer just like any other default outline and bitmap font modules in it. With FreeType_MF_Module, users don't need to pre-convert METAFONT into its corresponding outline font before using it with FreeType rasterizer as FreeType_MF_Module automatically performs this. It allows users to easily generate variants of font styles by applying different parameter values. This module is directly installed in FreeType rasterizer just like its default font modules so, there is no dependability and performance issues. We have tested our proposed module by generating different font styles with METAFONT and compared its performance with default modules of FreeType and other researches.

This paper is organized as follows. In Section 2, the related research regarding font modules and libraries are explained. The architecture of FreeType_MF_Module is explained in Section 3. The authors have tested the FreeType_MF_Module by demonstrating how FreeType can provide support for METAFONT directly in Section 4. FreeType_MF_Module's performance is also compared with FreeType default modules and other researches in this section. Section 5 gives concluding remarks.

2 Related research and their problems

MFCONFIG [4] is a plug-in module for Fontconfig [5]. It enables the use of METAFONT on Linux font system. Figure 1 shows the architecture of MFCONFIG module linked with Fontconfig.

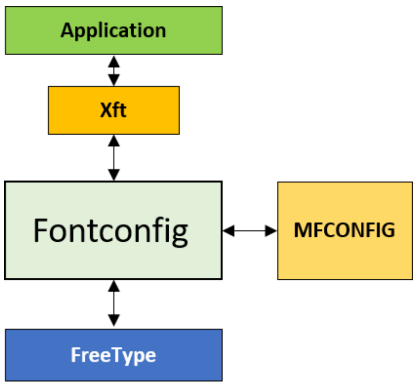


Figure 1: Basic architecture of MFCONFIG module

Although MFCONFIG supports METAFONT in Linux font system, it has performance and dependability problems. Since MFCONFIG is plugged into the high-level of font system i.e. Fontconfig, and not directly plugged inside FreeType, it makes its performance very slow compared to the font-specific driver modules supported by FreeType. Whenever the client application sends a METAFONT file request, Fontconfig communicates with MFCONFIG, performs operations, and then sends input to FreeType for rendering text. This whole process becomes slow because of the high-level operations before FreeType receives its input.

Other than the performance problem, MFCONFIG also has a dependability problem. As it is dependent on Fontconfig library, this means that if there is a font environment where there is no Fontconfig, then this module cannot be used. Fontconfig is mainly used in Linux font system for locating fonts and no other operating systems, so MFCONFIG cannot be supported in them.

VFlib [6], a virtual font library, is a font rasterizer developed for supporting multilingual fonts. VFlib can process fonts which are represented in different font formats and outputs glyphs as bitmap images from various font files. VFlib supports many font formats like TrueType, Type 1, GF, and PK fonts [7] etc. It provides a unified API for accessing different font formats. A new module can be added in this font library for adding support for METAFONT but this library has its own drawbacks; as it supports many different font formats and with a database support it can be very heavy font library for embedded systems. It is also dependent on additional font libraries like FreeType engine for TrueType font support, T1lib [8] for Type 1 font support so it has its dependency problems as well. Therefore, VFlib is not suitable to add support for METAFONT.

FreeType is a font rasterizer to render fonts and it can produce high quality output for mainly two kinds of font formats, vector and some bitmap font formats. FreeType mainly supports font formats such as TrueType, Type1, Windows, and OpenType fonts using same API irrespective of their font formats. Although FreeType supports many different font formats but it doesn't provide any support for METAFONT directly. If there is a module for FreeType that directly supports METAFONT then users can take the advantages of METAFONT by generating variants of font styles by just changing parameter values. MFCONFIG module problems can also be resolved using this module. FreeType can also be used for supporting TEX oriented fonts and not only outline or bitmap fonts.

The proposed FreeType_MF_Module in this paper uses the process for printing METAFONT from MFCONFIG module. FreeType_MF_Module intends to solve the two problems of MFCONFIG module. METAFONT can be used with any system having FreeType using the proposed module. As it is implemented just like any other default FreeType module, it can be easily installed or uninstalled.

3 Implementation of FreeType_MF_Module in FreeType rasterizer

3.1 FreeType_MF_Module as an internal module of FreeType

FreeType can support various font formats. Processing a font file corresponding to its format is done by an internal module in FreeType. This internal module is called a font driver. FreeType contains a configuration list of all the driver modules installed in a specific order. When FreeType receives a request of font file from an application, it passes this request to the driver module mentioned on the top of the list for processing. This module performs some internal operations to check if this font format can be processed or not. If this driver module supports this request, it performs all other operations to process this font file request. Otherwise this font file request is sent to the second driver module mentioned in the list. This process continues until a font driver is selected for processing the font file request. If no font driver can process the request, an error message is sent to the client application. Similarly, FreeType_MF_Module is directly installed inside FreeType just like its other internal modules. When the client application sends a request of METAFONT file, FreeType_MF_Module receives this request and processes it. Figure 2 shows how FreeType will select a driver module for processing a METAFONT file request.

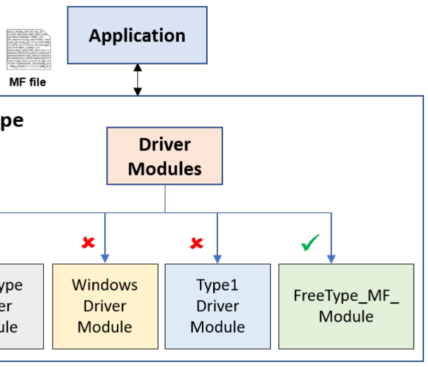


Figure 2: Process of selecting a module in FreeType

FreeType_MF_Module consists of three sub-modules: Linker module, Administrator module, and Transformation module.

3.2 Linker Module

Linker module is the starting point of FreeType_MF_Module. It is mainly responsible for linking FreeType internal modules with FreeType_MF_Module. It is divided into two parts: inner meta interface and outer meta interface. Inner meta interface receives font file request from internal modules and delivers it to Administrator module for processing. After processing by Administrator module, outer meta interface delivers the response to internal modules for further operations. The process of Linker module is shown in Figure 3.

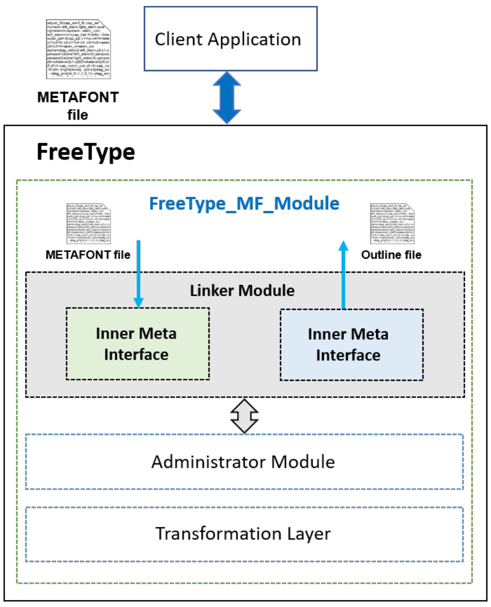


Figure 3: Linker module

3.3 Administrator Module

The core functionality of FreeType_MF_Module is performed in Administrator module. This module is divided into two layers; Search layer and Management layer.

Search layer is responsible for finding all the installed METAFONT fonts in a table. This table contains a list of all the METAFONT fonts installed and for fetching information related to them. Search layer is implemented in Meta scanner and Meta table.

Management layer mainly performs following tasks. (1) Checking whether the requested font file is METAFONT or not, (2) Checking the cache if the corresponding outline font for the METAFONT request is already stored. If yes, it directly sends the response from the cache. This functionality is implemented to achieve better performance and re-usability. (3) If the outline font is not prepared in the cache this request is sent to Transformation layer. (4) The outline font prepared by Transformation layer is stored in the cache (5) The response is sent back to FreeType internal modules by management layer. Management layer is implemented in Meta analyzer, Meta request, and Meta cache. Figure 4 shows the Administrator module and its sub layers.

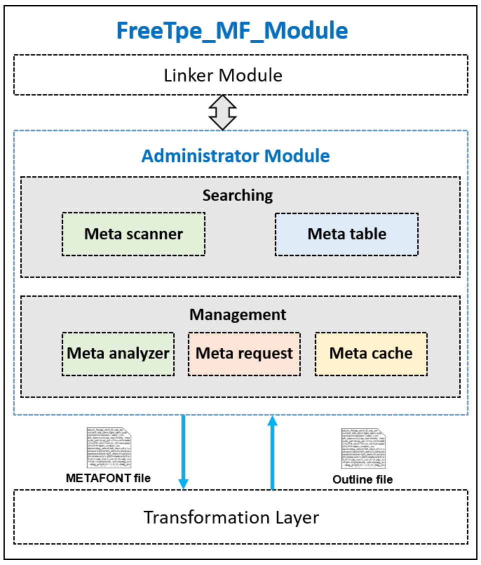


Figure 4: Administrator module

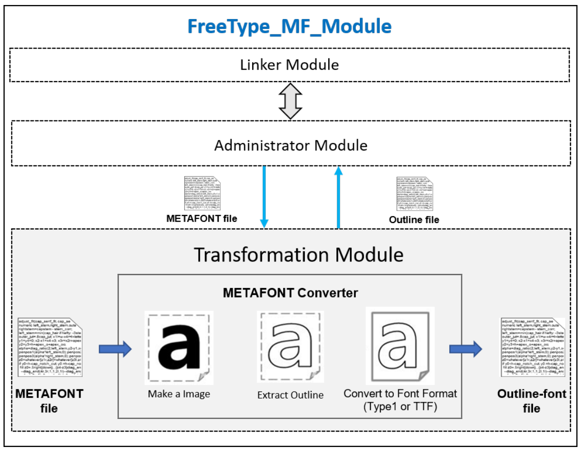


Figure 5: Transformation module

3.4 Transformation Module

Transformation module is mainly responsible for converting the METAFONT file into its corresponding outline font file. If the outline font file for a requested METAFONT file doesn't exist in the table then Administrator module sends the request to Transformation module. This module processes the request and returns the corresponding outline font file to Administrator module. Figure 5 shows how Transformation module converts METAFONT files into corresponding outline files.

3.5 METAFONT support in FreeType using FreeType_MF_Module

As shown in the Figure 6, FreeType_MF_Module is an internal module of FreeType which is responsible for processing METAFONT file request. First an application sends a font file to FreeType (step 1). If all other driver modules fail to process this font file request, this request is sent to FreeType_MF_Module through Linker module. Inner meta interface delivers this request to Administrator module (step 2). Meta request in Administrator module receives all information of this font file request and sends it Meta Analyzer to check if this font file is METAFONT or not (step 3). If this font file is not METAFONT this request is sent back to FreeType (step 3a). If this request is METAFONT file, Meta analyzer checks if this METAFONT file is installed or not by scanning Meta table. If this METAFONT information is not found in Meta table an error is sent back to FreeType internal modules(step 3b). There can be a scenario in which METAFONT is installed but its corresponding outline font is not stored in the cache. In this case, Meta cache is scanned to check if the corresponding outline file is stored in it (step 4). If it is already stored in the Meta cache with the same style parameters as requested, it is directly sent to FreeType (step 4a). If it is not stored in the Meta cache, the request is sent to Transformation layer (step 4b). Transformation layer converts the METAFONT file into its corresponding outline font by applying requested style parameters (step 5). Outline font is returned from Transformation module to administrator module where the Meta cache is updated for future re-usability (step 6). Outer Meta interface returns this outline font to core FreeType module for further processing (step 7). Lastly, FreeType renders this outline font that was made from the requested METAFONT with the styled parameter values.

The FreeType_MF_Module is perfectly compatible with the standard FreeType rasterizer. FreeType_MF_Module provides direct support of METAFONT in FreeType rasterizer just like its default Type1 driver module, TrueType driver module etc. The module manages METAFONT and its conversion to corresponding outline font. Client applications can request for any style parameters of METAFONT , FreeType_MF_Module processes them and the result can be easily displayed on the screen. As it is directly implemented inside FreeType rasterizer, it has no dependability problems as discussed in Section 2. FreeType_MF_Module can easily generate multiple font families like bold, italic, and bold-italic depending on the style parameter values passed to it.

4 Experiment and performance evaluation of FreeType_MF_Module

For the experiment, that FreeType_MF_Module can

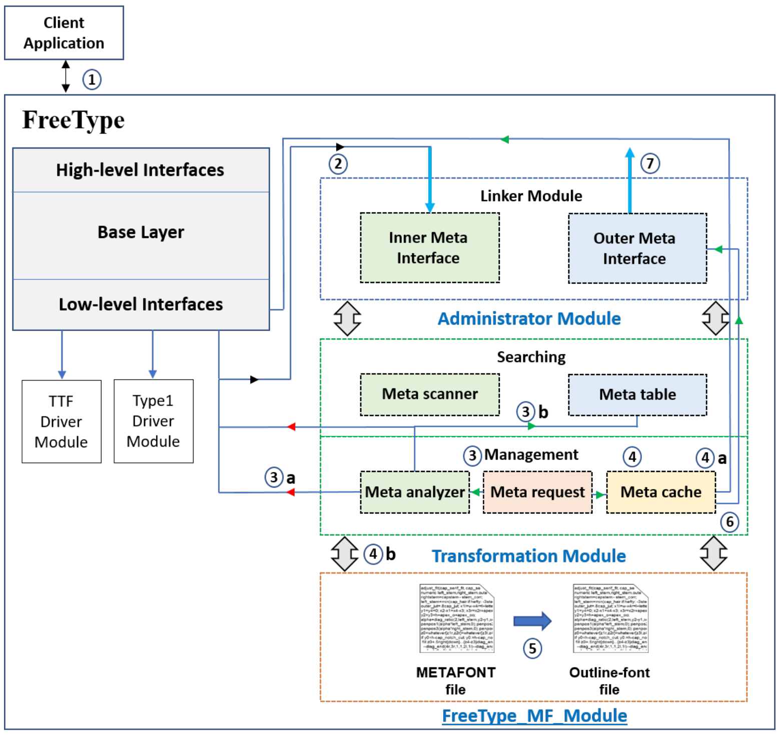


Figure 6: FreeType_MF_Module

Table 1: FreeSerif font family

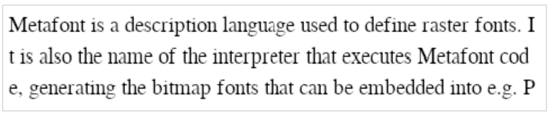
Style	Styled Output	Font files
Normal	FreeSerif	FreeSerif.ttf
Bold	**FreeSerifBold**	FreeSerifBold.ttf
Italic	*FreeSerifItalic*	FreeSerifItalic.ttf
Bold + Italic	***FreeSerifBoldItalic***	FreeSerifBoldItalic.ttf

Table 2: Various font styles with Computer Modern

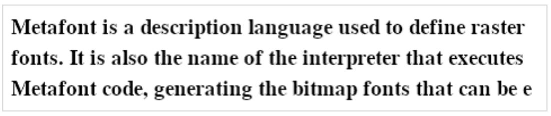
Style	Styled Output	Parameter values
Normal	ComputerModern	Default values of stem, hair, curve, slant
Bold	**ComputerModern**	stem+20, hair+20, curve+20, slant default
Italic	*ComputerModern*	Default values of stem, hair, curve, slant= 0.4
Bold + Italic	***ComputerModern***	stem+20, hair+20, curve+20, slant = 0.4

be used to generate different font styles from METAFONT the authors have used a font viewer application in Linux. This application directly uses FreeType to render fonts. It takes a font file and text as input and displays the styled text on the screen using X windows system. For testing, the authors have used all four styles of FreeSerif font family of TrueType font i.e. normal, bold, italic, bold-italic, and Computer Modern font of METAFONT.

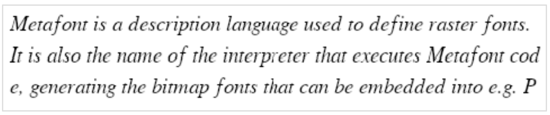
Table 1 shows the FreeSerif font family in four different styles. These styles are generated by using four different font files. Table 2 shows Computer Modern font in same four styles using different parameter values. These styles are made from one single METAFONT source file. The parameter values which are modified for generating these font styles are hair, stem, curve, and slant. The three parameters, hair, curve, and stem are related with the bold style. Incrementing their value, increases the boldness of text. These parameter values are different for lower-case and upper-case characters. The slant parameter is related with italic style. As shown in the Table 2, for normal style the default values of all these four parameters are used. For bold style, the values used are stem+20, hair+20, curve+20, and slant parameter default value. Default values of stem, hair, curve, and slant= 0.4 are used for italic style. Whereas, stem+20, hair+20, curve+20, slant = 0.4 values are used for bold-italic style. Similarly, many other font styles can be generated with this


Metafont is a description language used to define raster fonts. I
t is also the name of the interpreter that executes Metafont cod
e, generating the bitmap fonts that can be embedded into e.g. P

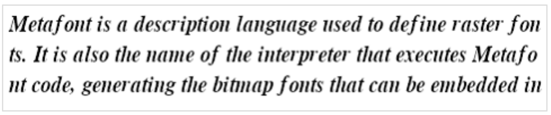
(a) FreeSerif normal style


Metafont is a description language used to define raster
fonts. It is also the name of the interpreter that executes
Metafont code, generating the bitmap fonts that can be e

(b) FreeSerif bold style

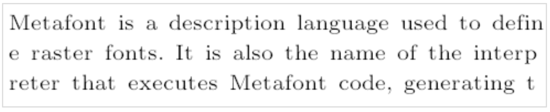

Metafont is a description language used to define raster fonts.
It is also the name of the interpreter that executes Metafont cod
e, generating the bitmap fonts that can be embedded into e.g. P

(c) FreeSerif italic style

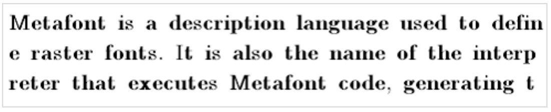

Metafont is a description language used to define raster fon
ts. It is also the name of the interpreter that executes Metafo
nt code, generating the bitmap fonts that can be embedded in

(d) FreeSerif bold-italic style

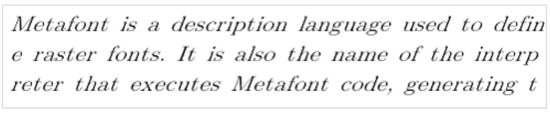
Figure 7: Dataset printed with FreeSerif font family


Metafont is a description language used to defin
e raster fonts. It is also the name of the interp
reter that executes Metafont code, generating t

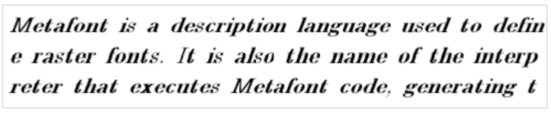
(a) default values of stem, hair, curve, slant


Metafont is a description language used to defin
e raster fonts. It is also the name of the interp
reter that executes Metafont code, generating t

(b) stem+20, hair+20, curve+20, slant default


Metafont is a description language used to defin
e raster fonts. It is also the name of the interp
reter that executes Metafont code, generating t

(c) default values of stem, hair, curve, slant= 0.4


Metafont is a description language used to defin
e raster fonts. It is also the name of the interp
reter that executes Metafont code, generating t

(d) stem+20, hair+20, curve+20, slant = 0.4

Figure 8: Dataset printed with Computer Modern

single METAFONT source file by changing parameter values.

For the performance testing of FreeType_MF_Module compared with FreeType default driver modules and MFCONFIG module, an experiment was performed using the same font viewer application. All four font files of FreeSerif in Table 1 were used for testing TrueType driver module of FreeType, Computer Modern source file was used with four different parameter values to generate four different styles in Table 2. For the text input, a sample dataset was used, which comprised of 2,000 words and over 8000 characters, including space character. The average time per millisecond between the font style request from application and the successful display of styled text on the screen was computed and compared.

Figure 7 shows the result of printing four FreeSerif fonts and Figure 8 shows the result of four Computer Modern METAFONT fonts. Table 3 shows the average time to print the dataset with FreeSerif font by TrueType driver module, Computer Modern font by FreeType_MF_Module, and Computer Modern font with MFCONFIG module. The default TrueType driver module of FreeType takes 3 ms to 7 ms to print dataset with all four families of FreeSerif font. FreeType_MF_Module takes 4 ms to 10 ms to print this dataset with Computer Modern font. Whereas, MFCONFIG module took 50 ms to 120 ms to display similar size dataset.

The performance of FreeType_MF_Module is comparatively slower than default FreeType driver module, because it takes an extra time to convert a METAFONT into its corresponding outline font by applying styled parameters. Whereas, FreeType_MF_Module has a very good performance than MFCONFIG module, because it is directly implemented inside FreeType rasterizer just like any other internal module of FreeType and it is not dependent on any other font libraries like Fontconfig and Xft [9] etc. Hence, we can conclude that FreeType_MF_Module can be used in FreeType rasterizer for providing direct support of METAFONT in almost real time on a modern Linux PC.

FreeType_MF_Module is a suitable module to provide users with parameterized font support on the screen by applying style parameters directly to the METAFONT font. Users don't need to pre-convert METAFONT into its corresponding outline font before using with FreeType rasterizer, as FreeType_MF_Module automatically performs this. The users can use METAFONT easily just like TrueType fonts using FreeType. FreeType_MF_Module has also overcome the problems of MFCONFIG module. The performance can be further improved by optimizing the METAFONT converter in Transformation layer. Currently, the METAFONT converter works with mftrace and autotrace programs. Future work will consider

Style	TrueType driver Avg. Time	FreeType_MF_Module Avg. Time	MFCONFIG Avg. Time
Normal	4.5 ms (3-6)	6 ms (5-8)	70 ms (50-80)
Bold	4 ms (4-6)	7 ms (6-9)	85 ms (70-100)
Italic	4 ms (4-6)	6 ms (4-7)	105 ms (70-110)
Bold + Italic	5 ms (5-7)	8 ms (6-10)	100 ms (90-120)

Table 3: Average time to display dataset with TrueType driver, FreeType_MF_Module, and MFCONFIG

about proposed module optimization and direct usage of TEX bitmap fonts like GF and PK which are not supported by FreeType rasterizer.

5 Conclusion

In this paper, we have proposed a module named FreeType_MF_Module, which enables the direct support of METAFONT in FreeType rasterizer. Outline fonts such as TrueType and Type1 don't allow users to easily change font styles. For every different font style in outline fonts, a new font file is created which can be very time consuming and costly process for CJK fonts which consists of large number of characters. To overcome this disadvantage of the outline font production method, METAFONT font can be used.

FreeType supports many different font formats like TrueType, Type1, windows font etc. but doesn't provide any support for METAFONT . The proposed module, FreeType_MF_Module is installed directly inside FreeType and can be used just like other internal modules to support METAFONT font. The authors have demonstrated experiments which shows that variety of styled fonts can be generated from METAFONT by adjusting parameter values in single METAFONT source file with FreeType_MF_Module in almost real time.

Acknowledgement

This work was supported by Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korean government(MSIT) (No. R0117-17-0001, Technology Development Project for Information, Communication, and Broadcast).

References

[1] Donald E. Knuth, *Computers and Typesetting,* Volume C: The METAFONTbook. Addison-Wesley, 1996.

[2] David Turner, Robert Wilhelm, Werner Lemberg, *FreeType,* www.freetype.org.

[3] Kim Jinpyung, *et al. Basic Study of Hangeul Font.* Seoul: Korea Publishing, Research Institute, 1988.

[4] Choi Jaeyoung, *MFCONFIG: METAFONT plug-in module for Freetype rasterizer* TUG 2016 (TUGboat, 2016): 163170.

[5] K. Packard, Keith Packard, Behdad Esfahbod, *et al. Fontconfig.* www.fontconfig.org.

[6] H. Kakugawa, *VFlib - a general font library that supports multiple font formants,* EuroTEX conference, March 1998.

[7] Rokicki T, *GFtoPK version 2.3,17 April 2001* https://www.tug.org/svn/texlive/trunk/Master/texmf-dist/doc/generic/knuth/mfware-/gftopk.pdf?view=co.

[8] Menzner R, *A library for generating character bitmaps from Adobe Type 1 fonts.* www.fifi.org/doc/t1lib-dev/t1lib_doc.pdf.gz.

[9] Keith Packard, *The Xft font library:* Architecture and users guide. Proceedings of the 5th annual conference on Linux Showcase Conference, 2001. https://keithp.com/keithp/talks/xtc2001/xft.-pdf.

A rollback concept for packages and classes

Frank Mittelbach

Abstract

In 2015 a rollback concept for the LaTeX kernel was introduced. Providing this feature allowed us to make corrections to the software (which more or less didn't happen for nearly two decades) while continuing to maintain backward compatibility to the highest degree.

In this paper we explain how we have now extended this concept to the world of packages and classes which was not covered initially. As the classes and the extension packages have different requirements compared to the kernel, the approach is different (and simplified). This should make it easy for package developers to apply it to their packages and authors to use when necessary.

Contents

1 Introduction ?1

2 Typical scenarios ?2

3 The document interface ?2
3.1 Global rollback . . . . ?2
3.2 Individual rollback . . . . ?2
3.3 Specifying a version instead of a date ?3
3.4 Erroneous input . . . . ?3
3.5 Advice for early adopters . . . . . . ?3

4 The package/class interface ?4

5 Special considerations for developers ?4
5.1 Early adopters . . . . ?5
5.2 New major release in beta . . . . . . ?5
5.3 Two major releases in use . . . . . . ?5
5.4 Fine grained control (if needed) . . . ?6
5.5 Using `l3build` for source management ?6

6 Command summary ?6
6.1 Document interface . . . . ?6
6.2 Package and class interface . . . . . ?6

1 Introduction

In 2015 we introduced a rollback concept for the LaTeX kernel that enables a user to request a kernel rollback to its state at a given date by using the latexrelease package [1]. For example,

```
\RequirePackage[2016-01-01]{latexrelease}
```

would result in undoing all kernel modifications (corrections or extensions) released between the first of January 2016 and the current date.[1] Undoing means reinstalling the definitions current at the requested date and normally also removing new commands from TeX's memory so that `\newcommand` and similar declarations do not fall over because a name is already declared.

This mechanism helps in correctly processing older documents that contain workarounds for issues with an older kernel, issues that have since been fixed in a way that would make the old document fail, or produce different output, when processed with the newer, fixed kernel.

If necessary, the latexrelease package also allows for rolling the kernel forward without installing a new format. For example, if the current installation is dated 2016-04-01 but you have a document that requires a kernel with date 2018-01-01, then this can be achieved by starting it with

```
\RequirePackage[2018-01-01]{latexrelease}
```

provided you have of a version of the latexrelease package that knows about the kernel changes between the date of your kernel and the requested date. Getting this version of the package is simple as the latest version can always be downloaded from CTAN. Thus you will be able to process your document correctly even when updating your complete installation is not advisable or impossible for one or the other reason.

However, rolling back the kernel state is only doing half of the job: the LaTeX universe consists of many add-on packages and those were not affected by a rollback request. We are therefore now extending the concept by providing a much simpler method for use in packages and classes, one that we think will be straightforward for the developers and also easy for document authors to use.

Unlike the method used by the kernel, which tracks every change individually and is able to roll back the code to precisely the state it had on any given day, the new method for packages and classes is intended to cover only major change points, e.g., the introduction of major new features or (incompatible) changes in syntax or interfaces.

As we will have only a few rollback points per package or class the different releases are all stored in separate files. In the main file it therefore only needs a single declaration per release to enable rollback. The downside is, of course, that for each release the whole package code is stored, instead of managing

[1] There are a few exceptions as some modifications are kept: for example, the ability to accept date strings in ISO format (i.e., 2016-01-01) in addition to the older LaTeX convention (i.e., 2016/01/01). These are not rolled back because removing such a feature would result in unnecessary failures.

the differences between releases. This is one of the reasons why this approach should be used only for major changes, i.e., at most a handful in the lifetime of a package.

From a technical perspective it is also possible to use the method introduced with latexrelease in package and class files, i.e., to mark up modifications using the commands `\IncludeInRelease` and `\EndIncludeInRelease`—the package's documentation [1] gives some advice on how to apply it in a package scenario—but the use of these commands in package code is cumbersome and results in fairly unreadable code, especially when there are many minor changes. This is an acceptable price to pay for fairly stable code, such as the kernel itself, since it offers complete control over the rollback to any date, but it is not really practical in package or class development and so, to our knowledge, it was therefore never used up to now. Section 5.4 gives some advice how to achieve fine-grain control in a somewhat simpler manner.

2 Typical scenarios

A typical example, for which such a rollback functionality would have provided a major benefit (and will do for packages in the future) is the caption package by Axel Sommerfeldt. This package started out under the name of caption with a certain user interface. Over time it became clear that there were some deficiencies in the user interface; to rectify these without making older documents fail, Axel introduced caption2. At a later point the syntax of that package itself was superseded, resulting in caption3 and then, finally that got renamed back to caption. So now older documents using caption will fail whilst documents from the intermediate period will require caption2 (which is listed as superseded on CTAN but is still distributed in the major distributions). So users accustomed to copying their document preamble from one document to the next are probably still continuing to use it without noticing that they are in fact using a version with defective and limited interfaces).

Another example would be the fixltx2e package that for many years contained fixes to the LaTeX kernel. In 2015 these were integrated into the kernel so that today this package is an empty shell, only telling the user that it is no longer needed. However, if you process an old document (from before 2015) that loads fixltx2e then of course fixes originally provided package (like the corrections to the floats algorithm). However, with a kernel rollback they would get lost as they are now neither in the kernel nor in the "empty" fixltx2e package if that doesn't roll back as well.

A somewhat different example would be the amsmath package, which for nearly a decade didn't see any corrections even though several problems have been found in it over the years. If such bugs finally get corrected, then that would affect many of the documents written since 2000, since their authors may have manually worked around one or the other deficiencies of the code. Of course, as with the caption package, one could introduce an amsmath2, amsmath3, ... package, but that puts the burden on the user to always select the latest version (instead of automatically using the latest version unless an earlier one is really needed).

3 The document interface

By default LaTeX will automatically use the current version of any class or package—and prior to offering the new rollback concept it always did that unless the package or class had its own scheme for providing versioning, either using alternative names or by hand-coded options that select a version.

3.1 Global rollback

With the new rollback concept all the user has to do (if he or she wants their document processed with a specific version of the kernel and packages) is to add the latexrelease package at the beginning of the document and specify a desired date as the package option, e.g.,

```
\RequirePackage[2018-01-01]{latexrelease}
```

This will roll back the kernel to its state on that day (as described earlier) and for each package and the document class it will check if there are alternate releases available and select the most appropriate release of that package or class in relation to the given date.

3.2 Individual rollback

There is further fine-grain adjustment possible: both `\documentclass` as well as `\usepackage` have a second (less known) optional argument that up to now was used to allow the specification of a "minimal date". For example, by declaring

```
\usepackage[colaction]
           {multicol}[2018-01-01]
```

you specify that multicol is expected to no older than the beginning of 2018. If only an older version is found, then processing such a document results in a warning message:

```
LaTeX Warning: You have requested, on input
   line 12, version '2018-01-01' of package
```

```
multicol, but only version
'2017/04/11 v1.8q multicolumn formatting
(FMi)' is available.
```

The idea behind this approach is that packages seldom change syntax in an incompatible way, but more often add new features: with such a declaration you can indicate that you need a version that provides certain new features.

The new rollback concept now extends the use of this optional argument by letting you additionally supply a target date for the rollback. This is done by prefixing a date string with an equal sign. For example,

```
\usepackage{multicol}[=2017-06-01]
```

would request a release of multicol that corresponds to its version in June 2017.

So assuming that at some point in the future there will be a major rewrite of this package that changes the way columns are balanced, the above would request a fall back to what right now is the current version from 2017-04-11. The old use of this optional argument is still available because existence or absence of the = determines how the date will be interpreted.

The same mechanism is available for document classes via the `\documentclass` declaration, and for `\RequirePackage` if that is ever needed.

3.3 Specifying a version instead of a date

Specifying a rollback date is most appropriate if you want to ensure that the behavior of the processing engine (i.e., the kernel and all packages) corresponds to that specific date. In fact, once you are finished with editing a document, you can preserve it for posterity by adding this line:

```
\RequirePackage[⟨today's-date⟩]
               {latexrelease}
```

This would mean that it will be processed a little more slowly (since the kernel may get rolled back and each package gets checked for alternate versions), but it would have the advantage that processing it a long time in the future will probably still work without the need to add that line later.

However, in a case such as the caption package or, say, the longtable package, that might eventually see a major new release after several years, it would be nice to allow the specification of a "named" release instead of a date: for example, a user might want to explicitly use version 4 rather than 5 of longtable when these versions have incompatible syntax, or produce different results.

This is also now possible if the developer declares "named" releases for a package or class: one can then request a named version simply by using this second optional argument with the "name" prefixed by an equal sign. For example, if there is a new version of longtable and the old (now current) version is labeled "`v4`", then all that is necessary to select that old version is

```
\usepackage{longtable}[=v4]
```

Note that there is no need to know that the new version is dated 2018-04-01 (nor to request a date before that) to get the old version back.

The version "name" is an arbitrary string at the discretion of the package author—but note that it must not resemble a date specification, i.e., it must not contain hyphens or slashes, since these will confuse the parsing routine.[2]

3.4 Erroneous input

The user interface is fairly simple and to keep the processing speed high the syntax checking is therefore rather light. Basically the standard date parsing from the kernel is used, which is rather unforgiving if it finds unexpected data.

Basically any string containing a hyphen or a slash will trigger the date parsing which then expects two hyphens (in case of an ISO date) or two slashes (otherwise) and other than these separators, only digits. If it does find anything else, chances are that you get a "`Missing \begin{document}`" error or, perhaps even more puzzling, a strange selection being made. For example, `2011/02` may mean to us February 2011 but for the parsing routine it is some day in the year 20 A.D. That is, it gets converted to the single number `201102`, so that, when this number is compared numerically to, say, `20000101`, it will be the smaller number, i.e., earlier, even though the latter is the numerical representation of January 1$^{\text{st}}$ 2000.

So bottom line: do not misspell your dates and all is fine. That hasn't been a problem in the past, so hopefully it will be okay to continue with just this light checking. If not, then we may have to extend the checks made during parsing.

3.5 Advice for early adopters

If your document makes use of the new global rollback features, then it should be processable at any installation later than early 2015, when the latexrelease package was first introduced. If the installation is even older, then it needs upgrading or, at least, one has to add a current latexrelease package to the installation.

[2] Of course, more sophisticated parsing could fix this, but we use the fast and simple parsing that scans for slashes or hyphens with no further analysis.

However, if your document uses the new concept for individual rollbacks of packages or classes (i.e., with the `=...` syntax in the optional argument), then it is essential to use a LaTeX distribution from 2018 or later.[3] Earlier distributions will choke on the equal sign inside the argument as they will only expect to see a date specification there.

4 The package/class interface

The rollback mechanism for packages or classes is provided by putting, at the beginning of the file containing the code, a declaration section that informs the kernel about existing alternative releases.

These declarations have to come first and have to be ordered by date because the loading mechanism will evaluate them one by one and, once a suitable release is found, it will be loaded and then processing of the main package or class file will end. If there are no such declarations, or if the older releases are all ruled out for one reason or the other, processing will continue as normal by reading all of the main file.

The old releases are stored in separate files, one for each release, and we suggest using a scheme such as ⟨*package-name*⟩`-`⟨*date*⟩`.sty` as this is easy to understand and will sort nicely within a directory. However, any other scheme will do as well, as the name will be part of the declaration.

The contents of this release file will will be simply the package or class file as used in the past. This means that before making a new version all you need to do is to make a verbatim copy of the current file and give it a new suitable name.[4]

This way it is also very simple to include older releases after the fact, e.g., to take our famous caption example, Axel could provide the very first version of his package as `caption-`⟨*some-date*⟩`.sty` and caption2 as `caption-`⟨*another-date*⟩`.sty` in addition to adding the necessary declarations to the current release.

The necessary declarations in the main file are provided by the two commands, `\DeclareRelease`, and `\DeclareCurrentRelease`, that must be used in a *release selection* section at the beginning of the file. For each old release you can to specify a ⟨*name*⟩, the ⟨*date*⟩ when it was first available and the ⟨*external-file*⟩ that contains the code.

```
\DeclareRelease
      {⟨name⟩}{⟨date⟩}{⟨external-file⟩}
```

Either the ⟨*name*⟩ or the ⟨*date*⟩ can be empty, but not both at the same time. Not specifying a ⟨*date*⟩ is mainly intended for providing "beta" versions that people can explicitly select but that should play no role in date rollbacks.

The current release also gets a declaration, but this time with only two arguments: a ⟨*name*⟩ (again possibly empty) and a ⟨*date*⟩ since the code for this release will be the rest of the current file:

```
\DeclareCurrentRelease{⟨name⟩}{⟨date⟩}
```

This declaration has to be the last one in sequence as it will end the *release selection* processing.

The order of the other releases has to be from the oldest to the newest since the loading mechanism compares every release declaration with the target rollback date and stops the moment it finds one that is newer than this target date. It will then select the one before, i.e., the last one that is at least as old as the target. Since the `\DeclareRelease` declarations with an empty ⟨*date*⟩ argument do not play a role in date rollbacks, they can be placed anywhere within the sequence.

If the rollback target is not a date but a name, the mechanism works in the same way with the exception that a release is selected only if the name matches. If none of the names is a match, then the mechanism will raise an error and continue by using the current release.

5 Special considerations for developers

While loading an older release of a package or class, both types of release declarations are made no-ops, so that, in case the files containing the code also have such declarations, they will not be looked at or acted upon. This makes it possible to simply move the code from an old release into a new file without the need to touch it at all. Of course, removing those declarations doesn't hurt and will make loading a tiny fraction faster.

As mentioned earlier, best practice for release names is to append the release date to the package or class name, but the ⟨*external file*⟩ argument also allows other naming schemes.

You may have wondered why you have to make a declaration for the current release, given that later on there will be a `\Provides...` declaration that also contains a date and a version string and thus could signal the end of the release declaration section. The reason is as follows: if you want to give your

[3] Alternatively you can try to roll the installation forward, by using a current latexrelease package together with a suitable date option.

[4] Instead of making a verbatim copy you may want to adjust the commentary added by `docstrip` at the top of the file. Though technically correct, it is a bit misleading if the file still contains the phrase "was generated from ...", given that it is now a frozen version representing a particular state in time, rather than being a generated one that can be regenerated any time as necessary.

current release a name, then it is best practice to to make that name something simple like `v4` (and keep it that way) even though your current package is technically already at `v4.2c` and is listed that way in the `\ProvidesPackage` declaration. For the same reason (given that not every minor change will be provided as a separate version to which people can roll back), the ⟨*date*⟩ in `\DeclareCurrentRelease` reflects when that major release was first introduced. Thus, after a while that date may well be earlier than the current package date.

5.1 Early adopters

For one or two years after the introduction of this new method, there is a danger that people with older installations will pick up an individual package from, say, CTAN that contains release declarations with which their kernel (from 2017 or earlier) is unable to cope. It may therefore be a good idea for developers to additionally add the following lines at the top of packages or classes when using the new rollback feature:

```
\providecommand\DeclareRelease[3]{}
\providecommand\DeclareCurrentRelease[2]{}
```

This way the declarations will be bypassed in case the kernel doesn't know how to deal with them.

As an alternative one could add a statement that requires a minimal kernel version, i.e.,:

```
\NeedsTeXFormat{LaTeX2e}[2018-04-01]
```

so that users get a clear error message that they need to update their installation if they want to use the current file.

5.2 New major release in beta

If you are working on a new major release of your package or class, you may want to get it out into the open so that people can try it and provide feedback. In that case current release is still the official release that should be selected by default and the "beta" version should only be selected if explicitly requested. To achieve that you could add

`\DeclareRelease{beta}{}{`⟨*external-file*⟩`}`

before

`\DeclareCurrentRelease{}{`⟨*some-date*⟩`}`

so that testers can explicitly access your new version by asking for it via

`\usepackage[`⟨*options*⟩`]{`⟨*package*⟩`}[=beta]`

while everyone loading the package without the extra optional argument would get the current release.

5.3 Two major releases in use

One special scenario for which this method is only partially suitable is the case where we have two major releases that are in continuing parallel use and that are both under active maintenance (i.e., receive bug fixes and other updates once in a while). In that case it is necessary to make one version the primary release and allow the other (and its updates) to be accessed only via names: a date rollback can obviously only work for one line of development.

For example, if both `v4` and `v5` of package foo are in use and you consider `v5` as being the go-forward version (even though you are still fixing bugs in the `v4` code), then you can deploy a strategy as in the following example:

```
% last v4 only release:
 \DeclareRelease{}{2017-06-23}
                {foo-2017-06-23.sty}
% first v5 release:
 \DeclareRelease{}{2017-08-01}
                {foo-2017-08-01.sty}
% patch to v4 after v5 got introduced:
 \DeclareRelease{v4.1}{}
                {foo-v4-2017-09-20.sty}
% patch to v5:
 \DeclareRelease{}{2017-08-25}
                {foo-2017-08-25.sty}
% another patch to v4:
 \DeclarelRelease{v4.2}{}
                 {foo-v4-2017-10-01.sty}
% nick name for the latest v4 if you
% want users a simple access via a name:
 \DeclareRelease{v4}{}
                {foo-v4-2017-10-01.sty}

% current v5 with further patches:
 \DeclareCurrentRelease{v5}{2018-01-01}
```

This way users can use `\usepackage{foo}[=v4]` to get the latest v4 release or use the more detailed release names such as `[v4.1]`. Of course, this means that if package foo was requested in version v4 (or one of its sub-releases) it will not change even if there is is a general rollback request via latexrelease.

Normally, this should be just fine, but if you really require automatic date rollback functionality on both major versions, because the two are really equal in rank, then you are essentially saying they are independent works with some common root. In that case you should give them two separate names, e.g., call the older version foo-v4 when you introduce version 5 of foo and from that point on manage the history independently.[5]

[5] While in rare cases this might be the best approach, try to avoid it as long term management will be problematical, to say the least.

5.4 Fine grained control (if needed)

As mentioned earlier, the interface is deliberately designed to be simple and easy to use. As a price, each rollback point is (by default) a separate file. The idea behind this is that there is not much point in managing each and every small change as a rollback point, but only those that possibly alter the behavior of a package within the document so that, when processing older documents, it is important to be able to get back to an earlier state.

However, if you find yourself in a situation where you have many rollback files with only minor differences, and you consider this unsatisfactory, then here is one other command at your disposal that you can use to combine several files into a single file. Within a file corresponding to a `\DeclareRelease` declaration you can use

```
\IfTargetDateBefore{⟨date⟩}
    {⟨before-date-code⟩}{⟨after-or-at-date-code⟩}
```

This must be used after the *release selection* section (if present) and has the following effect: If the user requested, say, `[=2017-06-01]` then the mechanism first selects the file that is supposed to be current on that date, i.e., the release that was introduced on that date or is the last one that was introduced before that date. Now, if in this file we have a statement like the above, then the ⟨*date*⟩ is compared to `2017-06-01` and depending on the outcome either ⟨*before-date-code*⟩ or ⟨*after-or-at-date-code*⟩ is executed.

This way a single external file can hold rollback information for several patches on distinct dates, but of course, the burden is then on the developer to add the appropriate declarations, which is a little more work than just copying and renaming files.

The alternative is to use `\IncludeInRelease` and `\EndIncludeInRelease`. The latexrelease package documentation [1] gives some advice on how to apply those commands.

5.5 Using `l3build` for source management

If you use `l3build` [2] for managing your sources, then it is necessary to ensure that the files for the old releases are copied into the distribution. To support this, the default configuration for `l3build` specifies

```
sourcefiles = {"*.dtx", ".ins",
               "*-????-??-??.sty"}
```

i.e., all `.dtx` and `.ins` files, together with all `.sty` files matching the naming convention suggested in this article, are automatically included in the build.

If you prefer a different naming convention you have to adjust this setting in the `build.lua` file of your project. Otherwise you are ready to go without any adjustments.

6 Command summary

6.1 Document interface

For a global rollback of kernel and packages, use

```
\RequirePackage[⟨target-date⟩]{latexrelease}
```

at the beginning of your document.

To request a rollback for a single package or class, use the second optional argument with the date preceded by an equal sign, i.e.,

```
\documentclass[⟨options⟩]{⟨class⟩}[=⟨date⟩]
\usepackage [⟨options⟩]{⟨package⟩}[=⟨date⟩]
```

6.2 Package and class interface

To declare an old or special release, use

```
\DeclareRelease
      {⟨name⟩}{⟨date⟩}{⟨external-file⟩}
```

Leave the ⟨*name*⟩ argument empty if rollback should be only via dates. Leave the ⟨*date*⟩ empty if this special release should be accessible only via its name.

Always finish this *release selection* section with a declaration for the current release:

```
\DeclareCurrentRelease{⟨name⟩}{⟨date⟩}
```

In this declaration you must provide a ⟨*date*⟩ but the ⟨*name*⟩ can be left empty (which is the usual case).

Within a release file (but after the *release selection* section), you can specify conditional code to be selected based on a requested rollback date by using:

```
\IfTargetDateBefore{⟨date⟩}
    {⟨before-date-code⟩}{⟨after-or-at-date-code⟩}
```

References

[1] The LaTeX Team. *The* `latexrelease` *package*, April 2015. Available at `https://www.latex-project.org/help/documentation`.

[2] The LaTeX Team. *The* `l3build` *package — Checking and building packages*, March 2018. The file `l3build.pdf` should be part of your installation. Run "'`texdoc l3build`" to find it.

◇ Frank Mittelbach
`https://www.latex-project.org`