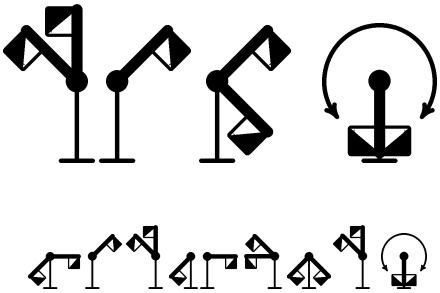
TEX METAFONT



3

ZPRAVODAJ 99

ČESKOSLOVENSKÉHO SDRUŽENÍ UŽIVATELŮ TEXU

ISSN 1211-6661 Ročník 9

OBSAH

Zpravodaj Československého sdružení uživatelů TEXu je vydáván v tištěné podobě a distribuován zdarma členům sdružení. Po uplynutí dvanácti měsíců od tištěného vydání je poskytován v elektronické podobě (PDF) ve veřejně přístupném archívu dostupném přes `http://www.cstug.cz`.

Své příspěvky do Zpravodaje můžete zasílat v elektronické podobě anonymním ftp na `ftp.icpf.cas.cz` do adresáře `/wagner/incoming/`, nejlépe jako jeden archivní soubor (`.zip, .arj, .tar.gz`). Současně zašlete elektronickou poštou upozornění na `mailto:bulletin@cstug.cz`. Uvedený adresář je pro vás „write/only“. Pokud nemáte přístup na Internet, můžete zaslat příspěvek na disketě na adresu:

Zdeněk Wagner
Vinohradská 114
130 00 Praha 3

Disketu formátujte nejlépe pro DOS, formát Macintosh 1.44 MB je též přijatelný. Nezapomeňte přiložit všechny soubory, které dokument načítá (s výjimkou standardních součástí $\mathcal{CS}$TEXu), zejména v případě, kdy vás nelze kontaktovat e-mailem.

ISSN 1211-6661

Some $\mathcal{NTS}$ thoughts

Hans Hagen

The next stage

When we take a look at Peter Breitenlohner's ε-TEX, we see extensions in the spirit of TEX. Based on experiences with macro writing, some limitations are removed (more registers like `\dimen`), some optimizations have taken place (like `\aftergroup`), protection is introduced (`\protected`), there are some more conditional, expansion and scanning features, and when we look at the typographic engine, bidirectional typesetting has been added. In extending TEX a clear distinction has been made between what should go into ε-TEX and what into $\mathcal{NTS}$.

When we look at Hàn Thế Thành's PDFTEX, we see a new backend. This in itself is rather revolutionary, because there are strong movements to stick to DVI and use this format as the intermediate to others. It is also revolutionary, because it removes the postprocessing stage, and thereby forces macro packages to take care of everything themselves, instead of relying on the post-processor. Last of all, we see that PDFTEX introduces an additional paragraph break pass, using infinitesimal horizontal glyph scaling.

Taco Hoekwater has written an extension to ε-TEX that introduces extensive list manipulations that can be of help when writing parameter driven macro packages, as well as provides an alternative input parser, targeted at SGML. His extension does not concern typography.

When we summarize these developments, we can conclude that ε-TEX has smoothed the path to $\mathcal{NTS}$ by providing a suitable arena for discussions. At the same time PDFTEX opened the road to a more drastic deviation from traditional TEX, if only by boosting TEX into the next century where graphics and interaction will dominate. Taco's TEX makes clear that we cannot neglect the input side and should closely follow the developments in the SGML field. The lesson learned is that we should not be too afraid to extend TEX.

In practice only a few people extend TEX in the ways mentioned, and most users will fall back on macro packages that hide most of the details. This aspect cannot be neglected when we look into the future of $\mathcal{NTS}$. Extensions like those provided by PDFTEX can rather well be supported by macro packages, because they often have a concept in which different drivers can be used. The level to which the specific features are supported depend however on the flexibility in the lower level shell. In TEX object reuse is limited to boxes and not reflected at the output level, opposite to PDFTEX. But downward compatibility can be guaranteed to a great extent because in most cases duplicates can be used instead. Moving from TEX to PDFTEX is therefore rather smooth.

When we want to use some ε-TEX features and at the same time want downward compatibility, we are forced to provide (often) poor man's alternatives or fall-backs. For extensions like ε-TEX to be accepted, it's best when all users of a macro package switch to this extension at once. With extended ε-TEX, downward compatibility can be achieved by providing preprocessors or clever but slow macros. So here, massive adaption is less important: use it when you need it.

In this respect the lesson learned is that a change should be guided and guarded, especially when many users depend on the macros other people provide. Before I will go into more detail into some challenging extensions that $\mathcal{NTS}$ can bring, I want to set the framework in which such extensions can take place.

If we value $\mathcal{NTS}$ as just a re-implementation, the project is nearly finished. In about a year the monolithic PASCAL code is transformed into highly structured JAVA, well tested and documented. But right from the start the project team was more ambitious. Of course, when the first version is available, everyone can take the source code and start writing his or her own Yet Another New Typesetting System, but the large body of the current TEX users will quite certainly benefit from some coordination. Ideally the situation around TUG 2000 conference will be as follows:

- There is a robust $\mathcal{NTS}$ environment, that users can easily download and plug into their JAVA enhanced operating system.
- The $\mathcal{NTS}$ project team has set up an infrastructure to host discussions, if possible centered around topics, identified in earlier stages of for instance the development of ε-TEX.
- To enable a smooth adaption of $\mathcal{NTS}$ without losing existing functionality, existing extensions to TEX will become available as plug ins into $\mathcal{NTS}$.
- Extensions are discussed, prototyped, validated, developed and documented in a professional context conforming standards laid down by the project team.
- The $\mathcal{NTS}$ project team will guard consistency and take care of proper hosting of new functionality.
- Macro packages will adapt new features in such a way that continuity is guaranteed for the large TEX user base.

Of course, use and reuse of code by whoever wants to do so will be stimulated. But, one of the strong points of good old TEX is that it is stable. The challenge will be to provide stability as well as flexibility. Imagine a macro package providing five different multi-column methods, each demanding different low level features. Ideally your system will pick up the right (and latest) code needed to use the particular method.

Think of this ...

When the first official release of $\mathcal{NTS}$ becomes available, we can start thinking about implementing all those features missing from TEX. How about a more procedural programming language, name spaces, string manipulations, more advanced error recovery and loading patterns on demand. Sure, these are all worth thinking of, but the typographic extensions will be the most challenging. Here I will discuss some ideas (some of which will be demonstrated in the presentation). Will you stick to using TEX or will you move on to $\mathcal{NTS}$? Maybe the next paragraphs will convince you at least to closely follow what is happening.

Columns and grids

In spite of all efforts, nobody has come up with the perfect multi-column output routines. This is mainly due to the lack of proper support in the kernel of TEX the program. Before computer typesetting came around, much manual effort went into typesetting documents in columns, and translating more intuitive behaviour into an algorithm is not trivial. A few decades of TEX at least have taught us where the complications lay.

- First of all we want our columns to be perfectly balanced. This is trivial for pure text, but imagine lots of white space, like display math.
- We want floats to be moved to the best available location. Of course we want floats to span more than one column, and even spanning one and a half columns with a text flowing around the figure should be possible.
- In double sided output, we want lines to align on the opposing pages (spread). When we hold the paper towards a bright source of light, we want the lines on both sides of the paper to align too.
- We definitely don't want to end up with a few lines or words on the last page. Why not apply a small percentage of glyph scaling in such a way that we get full pages? Of course we will need more than paragraph and page optimization for this: we are dealing with the document as a whole.
- Columns may differ in width. Think of two columns, spanning one third and two thirds of a page. In the middle of such two columns we will want to typeset an illustration, and the text should follow the circular shape of this illustration.
- Talking of illustrations, instead of being something with fixed dimensions, the scale may be adapted, of course consistently, to suit the overall document appearance (grid, spread, and more).
- Are you still thinking from left to right? Text can go in all directions, and will be mixed too. The width of columns may change in the meantime. Anyone who has seen traditional Jewish religion documents, will see the challenge in nested columns with (foot)notes flowing around partial columns.

Do you know a macro package that offers this functionality? Some day there will be packages around, build on top of basic $\mathcal{NTS}$ functionality. The main question is, what and how should $\mathcal{NTS}$ provide this functionality. We don't want Just Another Desktop Publisher, because those are around already. We want the building blocks and want to pack them into meaningful, clever, semi--intelligent macros. Will we still talk of macros? Some of the things mentioned are not too difficult to program, but making the combination behave well is the challenge.

The overall appearance

TEX is famous for its breaking of paragraphs. Actually it is still famous for that, simply because the main stream word processors think in lines. In spite of the—in the eyes of some users—perfect line breaks, users and implementers have been in search for even better ways of dealing with paragraphs.

It is uniform greyness that we want. The more uniform it is, the more comfortable it reads. TEX's paragraph builder can be influenced upto a certain extent, but even experienced users cannot foresee how a combination of settings will influence the look and feel. So what will we need?

- Typesetting is more than manipulating metrics. Don't we need a typesetting system that looks at the glyphs themselves, the small graphics?
- People tend to disagree on what looks best, but experts often agree on what looks worse. Why not build in expert knowledge, or even better, build a system that learns from the user's rating?
- How is greyness calculated? Does $\mathcal{NTS}$ act upon the internal lists of glyphs, or does it first build a bitmap? At least then it knows how the pages come out. Is the validation a function of an output device? Will the shape of glyphs depend on the rating? Will TEX and METAFONT become one?
- Is, in validating the appearance, a model of the page needed, in terms of meaningful areas? If so, how is such a model defined? Do we need pattern recognition?

There is some experience with manipulating glyphs, using either a large set of alternatives or manipulating (stretching or shrinking) glyphs. But far more experimenting is needed in this area. What better tool to use than $\mathcal{NTS}$, where we can plug in a new mechanism without spoiling the rest of the system? Rivers, dirty skylines, big emergency stretched holes, they will all disappear with the years to come. Unless of course the migration from paper to screen has made us loose the feeling for details. And wasn't it the details that made Knuth develop TEX?

Embedded graphics

Ah, who does not want graphics these days. One of the strong points of TEX has always been the separation of isolated graphics from the document source. But at the same time, TEX's repertoire of graphic primitives is limited to those needed for building math glyphs: there are only horizontal and vertical rules. We want more.

METAPOST has demonstrated that combining TEX with graphics can be very stimulating. But at the same time, it demonstrates that exchanging information between a typesetting engine is far from trivial, especially because we are dealing with different concepts (and states).

- $\mathcal{NTS}$ needs a graphics engine, or maybe even several. Models for exchange of information between processes dealing with pure typesetting and drawing shapes need to be developed. Such mechanisms should cooperate naturally with the paragraph and page breaking as well.
- Typesetting along curves, turning shapes into outlines, and applying arbitrary filling and shading, it all makes sense.
- TEX is strong in math, but how about (bio)chemistry? Although satisfactorily results can be reached, more is needed. Haven't we all seen documents that made us wonder how to typeset that in TEX? Lots of thinking needs to go into that area.
- For some languages pasting together glyphs is not enough. Actually drawing glyphs, or even better: words or sentences can be an alternative. Even emotions can make it into typeset text. Strong handwriting oriented graphics has to meet expressive coding.

Layers in text

It was already showing up in DVI viewers quite early: searching through typeset text. As long as we are dealing with non-composed characters, searching is not so much a problem, but TEX is used for more than typesetting English. The internet has stimulated world wide searching and general mark up languages are used to enhance this. Formats used for dissemination of typeset text, like PDF, are already a bit more prepared for searching. How will developments like this influence future typesetting?

- First of all, the new system needs some more understanding about the typeset text. Support for UNICODE, unified glyph names is mandate.
- When searching through a document, some knowledge on what in language the text we're dealing with makes sense. Not only the (many) language(s) of a text, but the direction also plays a role. Complicated ligatures should be recognized properly.
- In more dynamic documents, like fill-in-forms, interaction with a typesetting engine is not a luxury, especially not in European and Eastern

languages. $\mathcal{NTS}$ can be such a plug in, but the document itself should contain the information needed to let $\mathcal{NTS}$ to do its task. A document is more than a collection of graphics and glyphs, and typesetting more than organizing those.

- As PDFTEX already demonstrates, using TEX to embed typeset information like pop-up-help and tool tips is a breeze. Although heavily dependent of features of viewers, $\mathcal{NTS}$ will benefit from a decent model of layers on which we typeset as well as concepts of information hidden in the output but showing up at wish.

Just in case one wonders what gadgets like tool tips have to do with typesetting, think of a tool tip that has to pop up left or right aligned, depending of the position on the page. It does not make much sense to let 10 centimeters of tool tip disappear into the margin.

Conclusion

I did not mention things like colour. Implementing colour is rather trivial although different models are possible. Will we think in colour as an attribute to a glyph? Or a word, or sentence? Is there something like a background, and if so, what is the background of a line? As long as models don't conflict, they can co-exist peacefully. This is unlike different views on paragraph breaking can conflict and lead to unwanted compromised or dead-locks. It will be clear that implementations, $\mathcal{NTS}$ the program, macro packages, and distributions, should all cooperate smoothly to make $\mathcal{NTS}$ a reality.

From ε-TEX we have learned that even when ideas are circulating in our mind, making them explicit and ready for implementation is not trivial. Discussion, conception and implementation takes time. It is no problem to come up with some quick and dirty hacks, but we want to go for the best. It will not be a one year job. ε-TEX has demonstrated that a well documented and structured PASCAL program can be extended to great length.

From PDFTEX we have learned that extensive experimenting has its value. Adding features, but also removing them when not useful or not fitting into TEX's way of dealing with things, has its benefits. The nice thing about PDFTEX is that there are many users who are involved in the testing. Because most changes only affect the low level interfaces of the the main stream macro packages, common users are not that aware of the many new primitives that are needed because specials have become obsolete. PDFTEX has also demonstrated that extending TEX in its current incarnation has reached its limits. It's an interesting breed of PASCAL and C, that incidentally proved that incorporating ε-TEX functionality was more easy than expected.

From extended ε-TEX we can learn that developments can be driven from need. Direct interpretation of SGML was needed and therefore written. Minimal

discussion sometimes pays off. Being an extension not related to output (DVI, POSTSCRIPT, PDF) or specific typesetting, it proves that extensions like that can be hooked into a typesetting system without disturbing and breaking existing code.

One may wonder if $\mathcal{NTS}$ will make *those other TEX's* obsolete. Given that it takes time to come up with real new things, we can be sure that the predecessors of $\mathcal{NTS}$ will be around for a long time, if only because they have virtually no bugs, are supported by macro packages and most of all do their job well. This gives the $\mathcal{NTS}$ developers the time needed to come up with real good concepts and implementations.

Given that the TEX community has demonstrated that extending a stable program is feasible without harming existing, often critical, typographic production processes, $\mathcal{NTS}$ has a great future, although, in many areas, we haven't yet reached the limits of traditional TEX. It takes time and discussion, but talking of life-long tools, we have some time left. It is up to the $\mathcal{NTS}$ team to stimulate and guard this process, and we hope we will not fail you.

It sounds like I'm believing it myself, doesn't it?

Hans Hagen
PRAGMA Advanced Document Engineering
Ridderstraat 27
8061GH Hasselt
The Netherlands
E-Mail: pragma@wxs.nl

$\mathcal{NTS}$: nový sázecí systém

KAREL SKOUPÝ

Počátky projektu

Diskuse o potřebě dalšího vývoje TEXu či vytvoření jeho nástupce se rozvinula již počátkem devadesátých let. Vedly k tomu důvody technické i politické. Politické důvody by se daly velice stručně vyjádřit jako potřeba udržení a zvýšení atraktivity TEXu pro zachování a rozvoj komunity uživatelů. Více informací o tom lze najít zejména v [8].

Technické důvody spočívají především v požadavcích na ještě vyšší kvalitu a rozsáhlejší možnosti zpracování dokumentu. Týká se to například sazby ve

více sloupcích, sazby na rejstřík, optimalizace stránkového zlomu a mnoha dalších problémů, viz. [4]. Zároveň by byla jistě vítána možnost jednoduššího začlenění sázecího systému v rámci jiné aplikace, sdílení jen některých částí (sázecího stroje) jinými systémy a konečně podpory dalších vstupních i výstupních formátů.

Donald E. Knuth, autor TEXu, vyjádřil podporu snahám o vytvoření nového systému, který by byl nástupcem TEXu, pro něho samotného však práce na sázecím systému skončila. Zavázal se pouze odstraňovat případné chyby, ale TEX je již zakonzervován v jeho současné podobě. Vyhradil si také právo na používání názvu TEX, nový sázecí systém se tedy musí jmenovat jinak.

Během diskusí se uvažovalo o několika možnostech dalšího vývoje. Od nejkonservativnější – ponechání TEXu tak, jak je – po nejradikálnější – vytvoření zcela nového sázecího systému pro další století. Projekt začal relativně konservativní cestou, jejímž výsledkem je ε-TEX. Tento systém je vlastně modifikací původních zdrojových textů TEXu, která přidává velmi užitečné primitivy a ruší některá omezení. Práce je prováděna ve stejném programovacím jazyce (Pascal-`WEB`) a je zachována přísná kompatibilita s TEXem.

Již na počátku se však uvažovalo i o radikálnější variantě. Tou se mělo stát úplné přeprogramování TEXu (pomocí jiného programovacího jazyka) tak, aby byl nový systém funkčně kompatibilní s TEXem, jeho architektura aby však byla maximálně otevřená a přístupná novým změnám a rozšířením. Tento náročnější projekt byl ale odložen na dobu, kdy budou k dispozici potřebné finance. Situace se změnila v roce 1997, kdy německé sdružení uživatelů TEXu DANTE e.V. rozhodlo o sponzorování projektu a na úkol mohl být najat programátor na plný úvazek (autor článku).

Základní požadavky

Jak již bylo uvedeno, jedním ze základních požadavků na nový systém je kompatibilita s TEXem. Dosažení tohoto cíle není snadné, má však svoje dobré důvody. Málokdo z uživatelů TEXu používá holý systém, většina využívá formátů a balíků maker, jako je například LATEX. Nový sázecí systém, který by nebyl podporován těmito nadstavbami, by byl těžko akceptovatelný, i kdyby poskytoval mnoho nových možností na úrovni primitiv. Další výhodou kompatibility je možnost testování systému pomocí existujících vstupních souborů. Existuje jich obrovské množství. Důležitým kritériem kompatibility bude absolvování `TRIP` testu (`CTAN:systems/knuth/tex/trip.tex`). Nový systém by jím měl úspěšně projít (až na drobné, dobře zdůvodněné odlišnosti). V neposlední řadě kompatibilita prokáže, že nový systém toho umí alespoň tolik, co starý TEX. V budoucích verších systému se však může stát, že kompatibilita bude bránit dalším rozšířením a v takovém případě bude lepší ji opustit.

Významnějším požadavkem je otevřená a modulární struktura nového systému. Zkušenost totiž ukázala, že modifikace stávajícího programového kódu TEXu je velice obtížná. Jedná se o monolitický program se spoustou globálních proměnných, které jsou přístupné odkudkoliv. Řada datových struktur je optimalizována pro co největší úsporu paměti, řada algoritmů zase pro maximální rychlost, někdy až nestrukturovaným způsobem. Vzhledem k výkonům tehdejších počítačů a k dostupným technologiím to všechno mělo v době vývoje TEXu jistě svůj dobrý důvod, vede to ale nejen k horší čitelnosti, zejména však k nepředvídaným následkům při změnách programu.

Na druhé straně je celý program velmi dobře dokumentován. Knuth pro účely dokumentace vyvinul svůj vlastní nástroj `WEB`, ten podporuje také definice a použití maker a umožňuje změnu pořadí úseků kódu pro zvýšení přehlednosti. Například kód, který nějak manipuluje s určitou datovou strukturou je typicky seskupen poblíž její definice – lze v tom spatřovat významný krok směrem k objektově orientovanému programování.

Nový systém by se měl tedy skládat ze samostatných modulů, které mají jasně definované rozhraní a jsou na sobě co nejméně závislé. Objektově orientované programování se zdá být pro tento účel nejvhodnějším.

Programovací jazyk

Prvním úkolem pracovní skupiny v rámci projektu re-implementace byl výběr programovacího jazyka. V minulosti se uvažovalo o jazycích vyšší úrovně pro rychlou tvorbu prototypu jako jsou PROLOG nebo LISP. Nakonec se ale rozhodování přiklonilo k objektově orientovaným jazykům a výběr se zúžil na tři kandidáty: Common Lisp Object System (CLOS), C++ a Java. Ačkoliv má každý z nich svoje specifické výhody oproti ostatním, byl po pečlivém výběru vybrán jazyk Java. Podrobnějšímu zdůvodnění této volby je věnován článek [19].

Uveďme některé charakteristiky jazyka Java. Jako nejmladší z výše uvedených jazyků se od nich mohl v mnohém poučit. Velmi zdařilým způsobem implementuje třídy, objekty a rozhraní. Třídy jsou překládány do takzvaného byte-kódu a mohou být dynamicky kombinovány v době běhu. Velmi příjemný je zabudovaný čistič nevyužité paměti (garbagge collector), který vylučuje chyby související s porušením paměti. Typy a jejich kontrola mohou také zachytit množství chyb od triviálních až po závažné. Zpracování výjimek vyžaduje přesné deklarace, které zabraňují opomenutí hraničních situací. Java je dokonale přenositelná a to i na úrovni přeloženého byte-kódu. Díky dnes již standardně poskytované možnosti překladu do nativního kódu počítače při spuštění aplikace je také efektivita téměř srovnatelná s aplikacemi psanými v C++. Pevnou součástí Javy jsou též balíky podporující sítě, grafiku, grafické uživatelské rozhraní a mnoho dalších.

Za velmi důležité považujeme spojení Javy s Internetem. Lze si představit, že bude možno (automaticky) natáhnout nové zásuvné $\mathcal{NTS}$ moduly, sdílet prostřednictvím sítě fonty a vstupní soubory nebo si systém interaktivně přizpůsobit svým požadavkům.

Implementace

Následující část je věnována popisu implementace současné rozpracované verse systému. Nejprve krátce vysvětlíme pojmy specifické pro objektově orientované programování a jazyk Java.

Objekt je programová entita, která sdružuje data a algoritmy pro manipulaci s nimi. Objekt má určitý stav, který je dán hodnotou jeho datových atributů a funkce, které umožňují tento stav zjišťovat a měnit, nebo ho využívat k jiným výpočtům.

Metoda je funkce sdružená s objektem (nebo třídou). Množina těchto funkcí příslušná k určitému objektu definuje jeho chování.

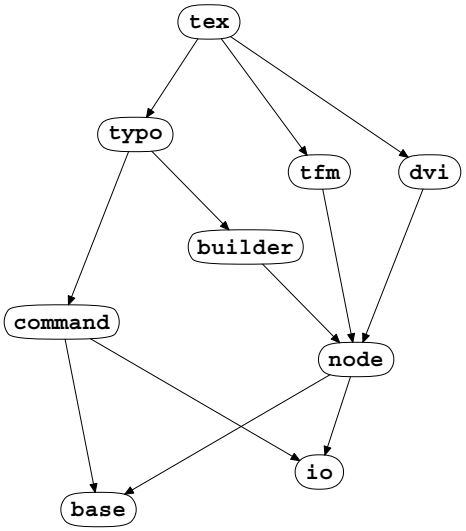
Třída (class) je abstraktní typ objektu. Obsahuje deklarace datových atributů objektu a definice jeho metod. Lze na ni pohlížet také jako na šablonu, podle které se vytváří objekty – její instance. Z již existujících tříd je možno odvozovat nové třídy. Odvozená třída dědí všechny atributy i metody své nadtřídy, může k nim přidávat vlastní atributy a metody nebo existující metody předefinovávat. Atributy a metody deklarované v rámci třídy mohou mít specifikována přístupová práva a to: pouze v rámci dané třídy, pro svoje potomky, v rámci balíku a jako veřejně přístupná. Veřejné metody a atributy představují rozhraní příslušného objektu pro okolní svět. Pokud je metoda (nebo atribut) deklarována jako statická, neváže se ke konkrétní instanci (objektu) ale k celé třídě.

Abstraktní třída je třída, která nevytváří žádné instance. Některé její metody mohou být abstraktní – mít definován počet a typy parametrů ale nedefinované tělo. Typicky je základem hierarchie podtříd, které předefinovávají její abstraktní metody.

Rozhraní (interface) je abstraktní typ objektu, který definuje pouze jeho vnější chování, ne však implementaci. Třída může deklarovat, která rozhraní implementuje (splňuje).

Balík (package) je soubor tříd a rozhraní, která spolu nějak souvisí. Lze ho chápat jako knihovnu určenou k distribuci. Slouží k jasnější organizaci programu a identifikaci jeho částí.

Polymorfismus je princip, který umožňuje objektům stejného typu odlišné (specifické) chování. Objekty, které mají stejné rozhraní, mohou být totiž různě implementovány.



Obrázek 1: Závislosti balíků v $\mathcal{NTS}$

Další popis je rozdělen podle balíků. Pro každý je uvedena stručná charakteristika a popis nejdůležitějších tříd a rozhraní.

Balík `base`

Hlavním posláním tohoto balíku je definice základních datových typů, které jsou používány zbytkem systému. Jedná se o minimální prvek hierarchie balíků $\mathcal{NTS}$, to znamená, že žádná třída zde nezávisí na jiných třídách z ostatních $\mathcal{NTS}$ balíků.

Nejdůležitějšími třídami zde jsou:

`Dimen` představuje rozměr měřený v tiskařských bodech (nebo `mu` jednotkách). Skutečnost, že interní representace je stejná jako u rozměrů v TEXu, zaručuje potřebnou kompatibilitu. Veřejné rozhraní této třídy se snaží být zcela nezávislé na její interní representaci. Vnějšímu světu se `Dimen` jeví jako zlomek (v tiskařských bodech). Definuje metody pro převody z celých čísel, zlomků daných čitatelem a jmenovatelem, reálných čísel a naopak. Poskytuje také základní aritmetické operace. V případě binárních operací jsou podporovány i jejich verse pro kombinace s ostatními převoditelnými číselnými typy. Obecné veřejné rozhraní dovoluje zásadní změnu rozsahu či dokonce vnitřní jednotky representace bez jakýchkoliv důsledků pro ostatní kód.

`Glue` odráží další, z TEXu dobře známý typ. Má svůj přirozený rozměr a velikost a řád roztažitelnosti a stlačitelnosti. Poskytuje aritmetické metody jako sčítání dvou `Glue`, násobení jednoduchým číslem a také varianty pro ostatní převoditelné typy.

`Num` representuje celé číslo. Je to normální integer, je ale zabalený do objektu a může být tedy přímo uložený v tabulce ekvivalentů a lze jej rozlišit od běžných integerů v kódu. Slouží zejména pro vyjádření hodnot číselných registrů (`\count`) (a je jaksi symetrická k `Dimen` a `Glue` pro registry `\dimen` a `\skip`).

Všechny výše uvedené základní třídy poskytují též metody pro získání jejich vyjádření řetězcem znaků, které může být zobrazeno na obrazovce, v log souboru nebo použito primitivou `\the`.

`LevelEqTable` je poslední důležitou a poměrně složitou třídou. Používá se k implementaci tabulky ekvivalentů a hašovací tabulky v TEXu. Zatímco TEX používá asociativní hašovací tabulku pouze pro významy řídících sekvencí, $\mathcal{NTS}$ ukládá mnohem více druhů ekvivalentů asociativním způsobem.

Jakýkoliv objekt může být spojen s jistou kombinací *druhu (kind)* a *klíče (key)*. Pro různé typy ekvivalentů jsou definovány různé druhy: jeden pro významy řídících sekvencí, jiný pro každou třídu registrů, další pro nastavení `\catcode`, atd. Klíč je (v závislosti na druhu) buď objekt (například jméno řídící sekvence) nebo číslo (pro většinu ostatních). Asociativní způsob ukládání hodnot registrů přirozeným způsobem zabraňuje omezení na jakýkoliv určitý počet registrů. Ačkoliv $\mathcal{NTS}$ je s TEXem kompatibilní v tom, že poskytuje pouze 256 registrů každého typu, toto omezení je zavedeno uměle a může být v budoucnu snadno odstraněno.

Jak už samo jméno napovídá, `LevelEqTable` také obhospodařuje zvyšování a snižování úrovní zanoření vyvolané skupinami ve vstupním jazyce a také uschovávání a obnovování asociovaných hodnot.

Ačkoliv registry byly přesunuty ze statických polí do asociativní tabulky, zůstává zde stále ještě jeden druh hodnot, který sice není asociativního typu, podléhá ale uschovávání a obnovování. Jedná se o parametry (jako je `\tolerance`, `\hsize`, ...). Jejich současná hodnota je uložena na jednom konkrétním místě. `LevelEqTable` poskytuje rozhraní i pro tyto externí ekvivalenty a spravuje jejich úschovu a obnovu.

Balík `io`

Tento balík obsahuje třídy a rozhraní potřebné pro čtení znaků ze vstupních souborů a pro zápis do log souborů. Oba tyto soubory mohou představovat i uživatelův terminál. Balík je nezávislý na ostatních balících stejně jako `base`.

`CharCode` je rozhraní a je velmi zajímavé. Vedli jsme v $\mathcal{NTS}$ týmu diskuse o tom, zda mají být kódy znaků representovány některým základním

typem jazyka Java. Padlo rozhodnutí, že je potřeba použít třídu, aby ji bylo možné odlišit od ostatních základních typů. Později (během vývoje systému) se ukázalo, že nejlépe svůj účel splňuje ještě abstraktnější vyjádření pomocí rozhraní. Toto rozhraní deklaruje metody pro získání odpovídající znakové nebo číselné hodnoty, srovnání s jiným `CharCode`, znakem nebo číslem na shodu, vytvoření odpovídající varianty `CharCode` pro malá a velká písmena, zápis na znakově orientovaný výstup a několik predikátů. Většina metod je zde proto, že jazyk TEXu používá znaky velice často nejen pro sazbu, ale také jako číselné hodnoty nebo součásti klíčových slov. Navíc mají některé znaky vliv na vstupní načítání a výstup do log souboru (`\endlinechar`, `\escapechar`, `\newlinechar`).

V současné době používá $\mathcal{NTS}$ pro implementaci rozhraní `CharCode` pouze objekt obsahující obyčejný znak. Existuje ale možnost použití zcela rozdílných representací (např. pojmenované znaky) bez jakékoliv změny programového kódu $\mathcal{NTS}$. Takové objekty mohou proplouvat celým systémem za předpokladu, že je na konci očekává výstupní objekt, který je rozpozná a správně použije. V nějaké budoucí aplikaci může dokonce i několik nezávislých implementací `CharCode` existovat současně.

`Name` se má k `CharCode` stejně jako se má `String` k `char`. Používá se k representaci jmen řídících sekvencí, jména úlohy a jmen fontů a souborů načtených ze vstupu.

`InputLine` odpovídá jednomu řádku ze vstupního souboru nebo uživatelova terminálu. Má metody pro získání dalšího `CharCode` nebo pro pouhé nahlédnutí jaký je příští kód beze změny současné čtecí posice. Interpretuje rozšířené znakové kódy (jako `^^M`), ignoruje koncové mezery a připojuje `\endlinechar`, je-li to potřeba. Další třída, `LineInput`, slouží jako vstupní posloupnost řádků `InputLine` ze souboru nebo terminálu.

`Log` je důležité rozhraní pro tisk informací do log souboru nebo na obrazovku terminálu. Deklaruje metody pro tisk hodnot základních typů a typů `String`, `CharCode` a `Loggable` (viz. níže). Je zde deklarováno i několik metod pro řízení výstupního zalámání řádků. Třída `StandardLog` implementuje rozhraní `Log` standardním TEXovským způsobem.

`Loggable` je velmi jednoduché rozhraní deklarující jednu metodu pro tisk do objektu typu `Log`. To se velice hodí, protože většina důležitých tříd v $\mathcal{NTS}$ toto rozhraní implementuje, a proto je jejich tisk velmi pohodlný.

Balík `command`

Třídy v balíku `command` tvoří interpret vstupního jazyka TEXu. Ačkoliv se jedná o rozsáhlejší balík, nemá zatím nic společného se sazbou. Ve skutečnosti nejméně jedna třetina zdrojového kódu TEXu nemá vůbec co dělat se sázením. Zdejší třídy jsou zodpovědné za proces načítání vstupních symbolů, jejich expanse a za

většinu zpracování nezávislého na sázecím módu, jako jsou definice maker a přiřazení hodnot registrům.

`Token` je abstraktní třída odpovídající vstupnímu symbolu. Deklaruje metody pro získání významu symbolu, přiřazení nového významu (pokud je to dovoleno), testování shody s jiným symbolem a množství predikátů, které říkají, zda lze předefinovat význam, zda se jedná o složenou závorku, písmeno a podobně.

K disposici je několik druhů symbolů a tvoří malou hierarchii podtříd. Typickými příklady jsou `CtrlSeqToken`, `ActiveCharToken`, `SpaceToken`, `LetterToken`, `LeftBraceToken`, ...

`Tokenizer` je schopen poskytnout nějakou posloupnost symbolů. Existují různé podtřídy jako například: `LineInputTokenizer` pro načítání symbolů ze vstupního souboru, `Macro.Expansion` pro tělo makra se specifikovanými makro parametry, `InsertedTokenList` pro seznam symbolů vložených na vstup z registru příslušného typu nebo `BackedToken` pro právě jeden zpátky vložený symbol. Objekty typu `Tokenizer` jsou vkládány do zásobníku `TokenizerStack`, který je analogií vstupního zásobníku TEXu.

`Command` je abstraktní třída representující každičký TEXovský příkaz. Příkazy jsou většinou primitivy TEXu registrované pod svým jménem v tabulce ekvivalentů, existují ale významné výjimky jako `Macro` nebo význam běžného znaku. V TEXu jsou symboly a příkazy representovány jedním typem a často jsou interpretovány oběma způsoby, což může být matoucí. $\mathcal{NTS}$ oba koncepty přísně odděluje. Jak bylo naznačeno výše, `Token` je symbolem načteným ze vstupu, který může mít určitý význam. Typem tohoto významu je právě příkaz `Command`, který je popisován zde.

`Command` má metody pro vykonání a expansi. Pouze některé příkazy mohou být expandovány a tato vlastnost je zjišťována pomocí další metody s pravdivostní hodnotou. Srdcem $\mathcal{NTS}$ je cyklus velmi podobný TEXovskému `main_control`. V rámci jednoho kroku je získán symbol ze vstupu a zkoumá se jeho význam. Je-li příslušný příkaz expandovatelný, volá se odpovídající metoda pro expansi. Vede-li expanse k neprázdnému výsledku, je metoda zodpovědná za jeho vložení do vstupního zásobníku. Není-li příkaz expandovatelný, volá se metoda pro jeho vykonání.

Ještě jedna zajímavost o expandovatelných příkazech. Pokud je jejich expansi zabráněno pomocí `\noexpand`, jsou vykonávány. V tom případě se chovají přesně jako `\relax` (nedělají nic kromě obnovení vnitřního stavu TEXu a zastavení dopředného hledání) a dokonce předstírají, že jsou `\relax`, pokud jsou zkoumány pomocí `\show`. Z tohoto důvodu je celý podstrom expandovatelných příkazů odvozen z příkazu `\relax`.

Další důležitou součástí rozhraní třídy `Command` jsou metody pro získání hodnoty určitého typu a indikaci, zda je příkaz schopen takovou hodnotu

poskytnout. Je to užitečné, pokud se příkaz vyskytne například na pravé straně přiřazení. Proto registry, parametry a několik dalších příkazů umí poskytnout hodnoty typu číslo, rozměr, výplň (glue) nebo seznam symbolů. Obrázek 2 ilustruje hierarchii tříd pro příkazy uvedením některých příkladů. V době psaní tohoto článku bylo pro příkazy implementováno více než 150 tříd (společně s příkazy pro sazbu).

`CommandBase` je nadtřídou třídy `Command`. Definuje pouze statické metody, které souvisí s načítáním různých vstupních prvků (jako čísla, rozměry, jména souborů, klíčová slova, ...), udržováním tabulky ekvivalentů, vstupního zásobníku a několika instancí výstupních objektů typu `Log`. Jak uvidíme později, kromě `Command` existuje více objektů, které využívají tyto služby, a proto jsou pro větší komfort odvozeny z této abstraktní třídy.

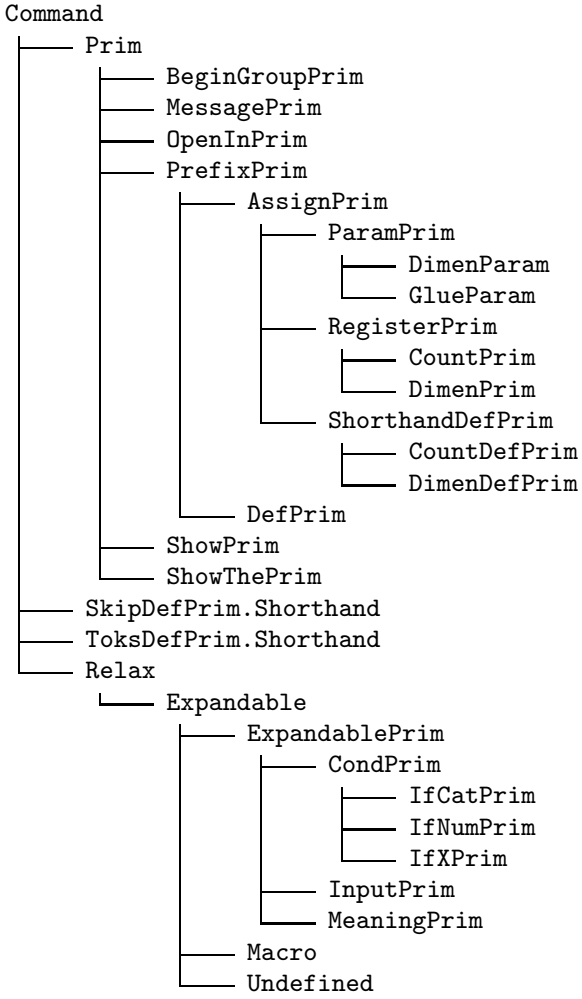
Balík `node`

A nyní se konečně dostáváme k sazbě! Třídy v tomto balíku representují sázecí materiál. Jsou zde také obecná rozhraní pro metriky fontů a generátory výstupních formátů. Balík je relativně nízko v hierarchii, třídy jsou závislé pouze na balících `base` a `io`. To jim dává dobrou šanci k samostatnému použití v jiném sázecím systému, který může používat zcela odlišný vstupní jazyk nebo uživatelské rozhraní.

`Node` je rozhraní představující uzel – základní stavební kámen sázecího materiálu. Má metody pro získání svých rozměrů (dokonce i když jsou ovlivněny roztažením nebo stlačením), k popisu sama sebe v log souboru a ke svému vysázení. Existuje celá hierarchie tříd implementujících rozhraní `Node`. Některé jsou jednoduché, například: `RuleNode`, `HKernNode`, `VKernNode`, `HSkipNode`, `VSkipNode`, `PenaltyNode`; jiné objekty jsou složené a mohou obsahovat seznamy podřízených uzlů: `HBoxNode`, `VBoxNode`.

`Packer` se používá, když chceme vypočítat rozměry složených boxů, které jsou vytvořeny ze seznamů uzlů. Tento proces se v TEXu nazývá pakování. Algoritmus je v podstatě stejný jak pro vodorovné, tak svislé seznamy uzlů, stačí jen prohodit vertikální a horizontální rozměry. Abstraktní třída `Packer` definuje abstraktní versi algoritmu a poskytuje prázdné virtuální metody pro získání odpovídajících velikostí uzlů. Potom jsou zde specialisované podtřídy pro horizontální a vertikální boxy, ze kterých jsou pak ještě mimo tento balík odvozeny další podtřídy, které ohlašují ta správná varování, pokud se něco nepovede v rámci tolerancí.

`FontMetric` je abstraktní rozhraní pro objekty poskytující informace o metrice fontu. V současné době jsou k disposici pouze známé `tfm` soubory, to ale není omezení pro $\mathcal{NTS}$. Ten je připraven pro jakýkoliv typ metrik, který lze přizpůsobit tomuto rozhraní. Poskytuje metody pro získání identifikace a různých číselných a rozměrových parametrů, aby byla dodržena



Obrázek 2: Část hierarchie tříd v balíku `command`

kompatibilita s TEXem. Ale především jsou zde metody, které umožňují získat instanci `Node` pro určitý `CharCode`, normální mezeru mezi slovy a speciální objekt, který umí produkovat representaci znaků, ligatur a kerningů pro danou posloupnost znakových kódů.

`TypeSetter` má podobné charakteristiky jako `FontMetric`. Definuje obecné rozhraní pro generátor výstupního formátu. Poskytuje metody pro vysázení znaku nebo čáry na současnou posici a posun této posice.

Balík `builder`

Tento balík se stará o oblasti, které souvisí s horizontálními, vertikálními a matematickým módy TEXu. Zatímco TEX má pouze jednu celočíselnou proměnnou, která odpovídá jednomu ze sedmi možných módů (tři výše uvedené jsou buď interní nebo externí a jeden prázdný „no mode“) na vrcholu sémantického zásobníku, $\mathcal{NTS}$ používá objekty, které budují sázecí materiál pro jednotlivé módy.

Tento balík je ve srovnání s balíkem `node` již více závislý na způsobu, jakým pracuje TEX, je ale stále nezávislý na vstupním jazyce. Je poměrně malý a jednoduchý. Určitá míra složitosti musí být vyřešena pro součinnost sázecích příkazů s jednotlivými módy, tuto problematiku ale řeší balík `typo`.

`Builder` je kořenem hierarchie tříd pro jednotlivé módy. Deklaruje některé predikátové metody pro získání jistých charakteristik daného módu a metody pro přidávání uzlu, kernu nebo skoku na konec seznamu, který je právě rozpracován. Vytváří příslušnou versi (horizontální nebo vertikální) kernů a skoků a je-li třeba, provádí další přizpůsobení. V současné době jsou podporovány pouze módy známé z TEXu, do budoucna je ale možné uvažovat o dalších módech (chemický, obrázkový, ...).

Balík `typo`

Balík `typo` je nadstavbou balíku `command`. Obsahuje všechny podtřídy třídy `Command`, které pracují s dosud implementovanou sazbou (přibude ještě balík `maths` pro příkazy matematické sazby). Využívá také balíky `builder` a `node`.

`TypoCommand` je podobný třídě `CommandBase` ale slouží sázecím příkazům. Je to meziúrovňová abstraktní třída, jež definuje několik užitečných statických metod. Udržuje zásobník módů (`Builder`) a současnou metriku fontu. Poskytuje metody pro načítání specifikace fontu nebo boxu ze vstupního souboru a přidání znaku nebo mezery do právě budovaného seznamu.

Mnoho tříd z tohoto balíku je odvozeno přímo ze tříd balíku `command`, protože od nich mohou zdědit užitečné chování. Nemohou však být obsaženy v balíku `command`, neboť vyžadují některé informace přístupné pouze v balíku `typo` (obvykle voláním některé statické metody třídy `TypoCommand`). V zásadě jsou dvojího druhu: jedním jsou `\if` primitivy jako `ifhmode` nebo

`ifvbox`, které potřebují jenom nějakou informaci o současném módu nebo určitém box registru; druhým případem jsou sázecí příkazy, které jsou nezávislé na módu, například `\setbox`, `\wd` a `\chardef`.

`BuilderCommand` je abstraktní nadtřídou příkazů, které jsou závislé na módu. V otevřeném systému, jako je $\mathcal{NTS}$, požadujeme, aby mohly být nové funkce snadno přidávány. Například abstraktní třída `Command` definuje skupinu metod, které mohou nové příkazy implementovat jakýmkoliv rozumným způsobem. Tento druh polymorfismu je podporován přímo zvoleným programovacím jazykem.

Ale co uděláme, pokud budeme chtít v budoucnu naprogramováním příslušné podtřídy `Builder` přidat další mód, který ale nabízí nové funkce dosud nedeklarované v rozhraní `Builder` a nějaké specialisované příkazy, které tyto nové funkce umí využít? Nebudeme chtít rozšířit základní rozhraní (alespoň ne do stávající verse systému) nebo dokonce nebudeme moci (pokud budeme programovat zásuvný modul). Jedinou možností bude zjistit aktuální typ současného `Builder` a použít nechvalně známý operátor přetypování, pokud se bude jednat o náš vylepšený mód.

Je zde ale další problém s existujícími příkazy, závislými na módu. Jak se mají chovat v novém módu? Pro tento účel třída `BuilderCommand` udržuje hašovací tabulku, která přidružuje objekt typu `Action` ke každé kombinaci třídy `Builder` a objektu typu `BuilderCommand`. Přiřazení je realizováno na úrovni konfigurace $\mathcal{NTS}$ a automaticky zohledňuje hierarchii tříd `Builder`. Díky takové pružnosti je velmi jednoduché předepsat chování příkazů v jednotlivých módech pro pozměněný systém.

Třída `Action` je podtřídou `CommandBase` a dědí tedy metody pro načítání vstupu, zacházení s log soubory a chybová hlášení. Podtřídy `Action` jsou obvykle implementovány jako vnitřní třídy příslušné podtřídy `BuilderCommand`. Takto vypadá úryvek z $\mathcal{NTS}$ konfigurace:

```
RulePrim    hrule = new RulePrim("hrule",
                                 default_rule, Dimen.NULL,
                                 Dimen.ZERO, Dimen.ZERO);

RulePrim    vrule = new RulePrim("vrule",
                                 Dimen.NULL, default_rule,
                                 Dimen.NULL, Dimen.ZERO);

hrule.defineAction(VertBuilder.class, hrule.NORMAL);
hrule.defineAction(ParBuilder.class, hrule.FINISH_PAR);
hrule.defineAction(HBoxBuilder.class, hrule.BAD_HRULE);

vrule.defineAction(HorizBuilder.class, vrule.NORMAL);
vrule.defineAction(VertBuilder.class, vrule.START_PAR);
```

Objekt typu `BuilderCommand` odpovídající TEXovské primitivě `\hrule` definuje tři akce: provádí normální operaci ve vertikálním módu, dokončí současný odstavec (pokud nějaký začal) v horizontálním módu a postěžuje si uvnitř horizontálního boxu. `\vrule` se zpracovává normálně v jakémkoliv horizontálním módu a započne nový odstavec ve vertikálním módu. Ve skutečnosti existuje pouze jedna třída (`RulePrim`), která má dvě instance pojmenované `hrule` a `vrule` s rozdílnými parametry. Jsou jim přiřazeny různé `Action` pro stejné módy. Všechny objekty typu `Action`: `NORMAL`, `START_PAR`, `FINISH_PAR` a `BAD_HRULE` jsou instancemi vnitřních tříd uvnitř `RulePrim` nebo jejích nadtříd.

Dalšími příklady `BuilderCommand` jsou: `HBoxPrim`, `VBoxPrim`, `VTopPrim`, `LowerPrim`, `MoveLeftPrim`, `BoxPrim`, `KernPrim`, `CharPrim`, `ExSpacePrim`, `AccentPrim`, `AnySkipPrim`.

`Group` je další podtřídou třídy `CommandBase`. Její podtřídy pokrývají různé typy skupin v TEXu. Existují skupiny jako `SimpleGroup` pro pár složených závorek, `SemiSimpleGroup` pro `\begingroup` a `\endgroup`, `HBoxGroup`, `VBoxGroup` nebo `VTopGroup`. Sama třída `Group` je definována uvnitř třídy `CommandBase` a tam je také udržován zásobník otevřených skupin. Většina jejích podtříd ale patří do balíku `typo`.

Třídy `Group` mají s třídami `Builder` jeden společný problém. Příkazy pro uzavírání skupin se chovají rozdílně v kombinaci s daným typem skupiny. Pravá složená závorka nemůže párovat `\begingroup` a `\endgroup` nemůže párovat levou složenou závorku. Problém je řešen úplně stejným způsobem jako pro kombinace příkazů a módů.

Balík `tfm`

Balík `tfm` implementuje pro $\mathcal{NTS}$ typ metriky fontu používaný v TEXu (`tfm`). Může sloužit jako příklad pro implementaci ostatních typů metrik.

`TeXFm` je třída representující nízkoúrovňový, téměř syrový, formát TEXovského souboru fontové metriky. Některé komplikace jsou odstíněny, ale veřejné rozhraní odráží pouze informace dostupné ze souboru. Používá několik pomocných tříd, `tfm` formát je totiž natolik složitý, že by bylo nepraktické ho podchytit v jednom programovém souboru. Například celý proces načítání `tfm` souboru je realizován třídou `TeXFmLoader`, která vytváří instanci třídy `TeXFm`. Samotná `TeXFm` má metody pro získání informací o jednotlivých znacích, ligaturách a kernincích pro dvojice znaků, receptů pro roztažitelné znaky a posloupností rostoucích znaků. Další metoda je k disposici pro tisk vlastní representace ve formě seznamu vlastností. Toho je využito v malé Java aplikaci `tftopl`, která díky třídě `TeXFm` sdílí většinu kódu s $\mathcal{NTS}$.

`TeXFontMetric` je adaptací `TeXFm`, která implementuje rozhraní `FontMetric` z balíku `node`. Je to vlastně obal, který využívá přirozené metody třídy `TeXFm` a definuje metody vyžadované zbytkem $\mathcal{NTS}$. Tento přístup je pravděpodobně užitečný pro budoucí implementace dalších typů metrik. Rozhraní jsou čistší a kromě toho nemůžeme očekávat, že někdo nezúčastněný poskytne rozhraní vyhovující přesně našim požadavkům, a to ani v případě, že poskytne pro přístup k metrice přímo třídu jazyka Java.

Balík `dvi`

Balík `dvi` implementuje výstupní formát `dvi` jako jeden z možných (budoucích) výstupních formátů $\mathcal{NTS}$. V mnohém se podobá balíku `tfm`.

`DviFormatWriter` představuje nižší vrstvu, která přesně pokrývá možnosti formátu `dvi`. Definuje metody pro výstup jednoho znaku nebo černého obdélníku, posun současné posice na stránce, začátek a konec stránky a pro definice a přepínání fontů. Tyto metody si hlídají konsistenci zapisovaných dat, není například možné vysázet znak nebo ukončit stránku, pokud stránka nezačala, dovolí ale nadefinovat font. Kromě toho si ve vlastní režii provádí optimalizace posunu současné posice.

`DviTypeSetter` je adaptací `DviFormatWriter` na rozhraní `TypeSetter` z balíku `node`. Řešení je analogické jako v případě tříd `TeXFm` a `TeXFontMetric` popsaných výše.

Balík `tex`

Tento balík zastřešuje ostatní $\mathcal{NTS}$ balíky a jedná se o zdaleka nejchaotičtější část systému. Doposud všechny třídy a balíky byly navrženy se záměrem poskytnout čisté a elegantní rozhraní a být co nejméně závislé na ostatních třídách a balících. Ale v samotném TEXu je spousta nejasných závislostí. To bylo také jedním z hlavních důvodů, proč projekt $\mathcal{NTS}$ vůbec vznikl. Třídy v tomto balíku navzájem spojují všechny dosud nezávislé jednotky a navíc sem byly přesunuty problematické případy, které nezapadaly do čistého návrhu ostatních balíků, pokud to bylo možné. To je hlavní důvod, proč zdrojové texty tohoto balíku někdy působí chaotickým dojmem.

Kromě toho jsou zde ještě třídy pro správu seznamu chybových hlášení a příkazy tak nejsou závislé na konkrétním způsobu, jakým jsou chyby hlášeny.

Nejzajímavější částí balíku je třída `Primitives`, která obsahuje konfiguraci celého systému. V popisu balíku `typo` se už vyskytovala jedna ukázka.

Modularita a konfigurovatelnost

Vybudování systému, který je co nejvíce modulární, bylo jedním z hlavních cílů projektu. V současné implementaci TEXu je množství závislostí. Zkušenost ukázala, že složitější změny jsou velmi obtížné a nebezpečné, protože mohou vést k mnoha nejasným a nezamýšleným důsledkům. V $\mathcal{NTS}$ bylo rozhodnuto učinit všechny závislosti zjevnými a čistými. Všechny třídy mají dobře definované rozhraní svých veřejných metod a to je využíváno pro veškerou komunikaci s příslušnými objekty. Nejsou zde žádné nekontrolovatelné změny globálních proměnných. Tento programovací přístup je významně podporován objektově orientovaným jazykem Java.

Dalším motivem pro vytváření nezávislých programových jednotek je možnost záměny některých modulů za moduly se stejným rozhraním ale rozdílnou implementací. Nezávislé třídy a balíky mohou být také použity coby stavební kameny jiného systému. Balíky $\mathcal{NTS}$ jsou tedy navrhovány spíše jako knihovny tříd s přísnou hierarchií.

Při rozkladu TEXu na nezávislé jednotky jsou zajímavým problémem kruhové závislosti. Existuje jich celá řada. Jednoduchým příkladem je vztah mezi TEXovýma „očima“ a „žaludkem“. Žaludek se sytí příkazy, které vnikají někde v očích, činnost očí ale závisí na nastaveních `\catcode`, ke kterým dochází v žaludku.

To činí snahu o udržování necyklické hierarchie balíků obzvláště obtížnou. Na druhé straně je ale tato snaha velmi žádoucí, pokud plánujeme použít pouze některé z balíků v nějaké jiné aplikaci. $\mathcal{NTS}$ k odstranění kruhových závislostí využívá metodu abstraktních rozhraní. Potřebuje-li určitá třída informaci nebo akci, která není dostupná na odpovídající úrovni hierarchie, definuje rozhraní a přijímá objekt vyhovující tomuto rozhraní jako parametr (konstruktoru nebo příslušné metody). Samotná parametrizace se pak provádí na některé z vyšších úrovní, zpravidla pod střechou (nebo možná v mozku $\mathcal{NTS}$), v rámci balíku `tex`.

Závěr

V průběhu prací vyvstaly některé otázky. Vyplatí se opravdu dodržovat kompatibilitu s TEXem? Ukázalo se totiž, že splnění tohoto požadavku vyžaduje mnohem větší úsilí, než se původně předpokládalo. Existuje mnoho hraničních situací, které je potřeba ošetřit a které se dají zjistit pouze pečlivým studiem zdrojového textu a testováním samotného TEXu. Přitom se zdá, že některé okrajové efekty ani nebyly zamýšleny. Na druhé straně je známo, že mnoha „špinavých triků“, které využívají mezní situace, se používá i v LATEXu a dalších nadstavbách. Vzhledem k důvodům, které byly uvedeny na začátku článku, tedy považujeme kompatibilitu za velmi důležitou a odhadujeme, že se v budoucnosti vyplatí.

Další otázka je spojena s použitým programovacím jazykem. Při praktickém používání se zjistilo, že některé jeho nedostatky byly podceněny. V Javě není možno odvodit uživatelské typy z primitivních typů (čísla, znaky), pro dodržení typové bezpečnosti se tedy používají třídy pro všechny odvozené typy. To pravděpodobně povede k nižší efektivitě. Mnohem důležitějším cílem projektu je ale poskytnout program typově čistý, bezpečný, jasný, dobře strukturovaný a pokud možno krásný, tak aby byl snadno udržovatelný a přístupný pro jeho další vývojáře a uživatele. Bohužel i zde má Java svoje nedostatky, chybí výčtové typy a především parametrické typy, které jsou v C++ již delší dobu známé jako šablony (templates). Vývoj Javy ale ještě není ukončen a je možné, že se řešení těchto problémů v dohledné době dočkáme. Konkrétní programovací jazyk ostatně není natolik podstatný, analýza programového kódu TEXu a objektově orientovaný návrh jsou důležitější.

Původní odhad doby potřebné k vytvoření první verse $\mathcal{NTS}$, kterou má být úplná re-implementace TEXu, byl jeden rok. Ukázalo se, že tento odhad byl poněkud optimistický. Systematická návrhářská a programátorská práce začala na jaře roku 1998 a stále ještě zbývá nezanedbatelnou část díla dokončit. Zejména chybí dělení slov, budování stránek, zarovnání a celá matematická sazba. Množství vykonané práce ale nelze měřit pouze množstvím kódu. Je třeba zahrnout analýzu a studium, návrh a vytvoření základní architektury nového systému. Z tohoto hlediska můžeme již dnes hovořit o dobrém výsledku. Je dokončena celá vstupní část systému včetně zpracování maker, většina zpracování nezávislého na módu, práce s metrikami fontů, manipulace s boxy, lámání odstavců a výstup do `dvi` formátu. To všechno zcela kompatibilně s TEXem, jak potvrdila úspěšná ukázka na konferenci EuroTEX '99.

Současný odhad dokončení první verse je začátek roku 2000 a její oficiální představení po předchozím testování a dokončení dokumentace se plánuje na TUG '2000 v létě. Cílem je poskytnout volně přístupnou základnu k experimentování pro všechny zájemce. Z praktických důvodů samozřejmě počítáme s tím, že i další vývoj systému bude pracovní skupinou koordinován.

Další informace o projektu $\mathcal{NTS}$ je možno získat z jeho oficiální stránky na: `http://nts.tug.org`.

Odkazy

[1] Ken Arnold, James Gosling: *The Java Programming Language, Second Edition*, Addison Wesley Publishing Company, Reading, Mass., prosinec 1997.

[2] Michael Barr: *TEX wish list*, v *TUGboat*, Vol. 13, No. 2, pp. 223–226, červenec 1992.

[3] Nelson H. F. Beebe: *Comments on the future of TEX and METAFONT*, v *TUGboat*, Vol. 11, No. 4, pp. 490–494, listopad 1990.

[4] Hans Hagen: *Some $\mathcal{NTS}$ thoughts*, v EuroTEX '99 Proceedings, pp. 233–240 září 1999, ISSN 1438-9959, též Zpravodaj Československého sdružení uživatelů TEXu, 9(3), 109–115(1999).
[5] Roger Hunter: *A future for TEX*, v *TUGboat*, Vol. 14, No. 3, pp. 183–186, říjen 1993.
[6] Donald E. Knuth: *The future of TEX and METAFONT*, v *TUGboat*, Vol. 11, No. 4, pp. 489–489, listopad 1990.
[7] Donald E. Knuth: *TEX: The Program*, Addison Wesley Publishing Company, Reading, Mass., 1986.
[8] Joachim Lammarsch: *The History of $\mathcal{NTS}$*, v EuroTEX '99 Proceedings, pp. 228-232, září 1999, ISSN 1438-9959.
[9] Frank Mittelbach: *E-TEX: Guidelines for future TEX*, v *TUGboat*, Vol. 11, No. 3, pp. 337–345, září 1990.
[10] Karel Skoupý: *$\mathcal{NTS}$: a New Typesetting System*, v TUG '98 Proceedings, pp. 167-171 srpen 1998 a v *TUGboat*, Vol. 19, No. 3, pp. 318–322, září 1998.
[11] Karel Skoupý, Philip Taylor: *The implementation of $\mathcal{NTS}$*, v EuroTEX '99 Proceedings, pp. 246-260 září 1999, ISSN 1438-9959.
[12] Philip Taylor: *TEX: The next generation*, v *TUGboat*, Vol. 13, No. 2, pp. 138–138, červenec 1992.
[13] Philip Taylor: *The future of TEX*, v EuroTEX '92 Proceedings, pp. 235-254, září 1992, ISBN 80-210-0480-0 a v *TUGboat*, Vol. 13, No. 4, pp. 433–442, prosinec 1992.
[14] Philip Taylor: *NTS: the future of TEX*, v *TUGboat*, Vol. 14, No. 3, pp. 177–182, říjen 1993.
[15] Philip Taylor: *NTS update*, v *TUGboat*, Vol. 14, No. 4, pp. 381–382, prosinec 1993.
[16] Philip Taylor: *Report of the 2nd meeting of the NTS group, February 1994*, v *TUGboat*, Vol. 15, No. 2, pp. 96–97, červen 1994.
[17] Philip Taylor: *Minutes of the NTS meeting held at Lindau on October 11/12th 1994*, v *TUGboat*, Vol. 15, No. 4, pp. 434–437, prosinec 1994.
[18] Philip Taylor: *NTS & ε-TEX: a status report*, v *TUGboat*, Vol. 18, No. 1, pp. 6–12, březen 1997.
[19] Jiří Zlatuška: *$\mathcal{NTS}$: Programming languages and paradigms*, v EuroTEX '99 Proceedings, pp. 241-245 září 1999, ISSN 1438-9959.

Karel Skoupý
E-Mail: skoupy@fi.muni.cz

Nový štandard textových editorov pre Windows? – II

Igor Podlubný, Katarína Kassayová

Tento článok je venovaný editoru NoteTab 4.6, určenému pre editovanie „plain“ textu. Vďaka viacerým inovačným riešeniam je tento editor výborným predstaviteľom novej vlny textových editorov pre Windows 95/98/NT, ktoré sú okrem iného vhodné aj na prípravu textov v TEXu, LATEXu, a pod.

Vlastnosti tohto editora (v tom čase ešte verzie 4.51), ktoré sú zaujímavé pre široký okruh možných užívateľov, boli podrobne popísané v týždenníku COMPUTERWORLD č.8/99 [1]. Do článku sa však z priestorových dôvodov nedostali ďalšie časti recenzie, ktoré by mohli byť zaujímavé pre pokročilých užívateľov.

Jedinečná vlastnosť – jednoduchá tvorba makier

Textové editory môžu byť v podstate posudzované podľa možností vytvárania a používania makropríkazov (makier), ktoré uľahčujú a urýchľujú prácu. Pre DOS bolo vyvinutých pomerne veľa editorov, ktoré umožňujú definovanie makier. Spomenieme napríklad QEDIT a TSE od firmy Semware, MultiEdit, ale aj vynikajúci editor CSED (autor: Pavel Ševeček), zahrnutý Česko-slovenským združením užívateľov TEXu do distribúcie CSTEXu. Spomenieme aj klasické borlandovské editory (viď článok [2]).

Po masovom rozšírení Windows však záujem o DOS-ovské editory začal klesať až k nule, a pritom užívateľom ušlo, že textové editory pre Windows im nepriniesli možnosť definovania makier. Je zaujímavé, že kým vo Windows 3.x ešte existoval Macro Recorder, ktorý tento problém čiastočne riešil a bol schopný aspoň zaznamenávať a prehrávať postupnosti uskutočnených krokov (dokonca vrátane presunov myšky!), vo Windows 95 už nie je nič, čo by umožňovalo napríklad zjednodušenie práce s NOTEPADom alebo WordPadom.

Tento vývoj má svoje vysvetlenie. Kým v DOSe užívateľ mohol používať rôzne kombinácie klávesov typu `CTRL+kláves`, `ALT+kláves`, alebo niekedy dokonca aj `SHIFT+CTRL+kláves` a `SHIFT+ALT+kláves`, vo Windows sú prakticky všetky takéto kombinácie už využité alebo rezervované pre vlastnú potrebu Windows. Toto veľmi obmedzilo možnosti textových editorov pod Windows. Ťažko prekonateľný „vrchol“ týchto obmedzení predstavuje NOTEPAD od samotného Microsoftu.

Za týchto existujúcich okolností sa autorovi NoteTabu Ericovi Fookesovi podarilo nájsť naozaj jedinečné elegantné riešenie:

- Po prvé, vymyslel spôsob, ktorý prekonáva problémy s definovaním klávesových skratiek pre definované makrá. Namiesto klávesových skratiek umožnil definovanie názvov makier, ktoré sú zobrazované na bočnom paneli editora a môžu byť vyvolané kliknutím myškou na meno makra. Ako tvrdí Kerry R. Krueger, autor recenzie na File Mine (`http://www.filemine.com/showJewel?id=151225`), NoteTab bol prvým textovým editorom, v ktorom sa objavilo toto riešenie.
- Po druhé, vyvinul vlastný veľmi prirodzený, jednoduchý a zrozumiteľný jazyk pre programovanie makier, ktorý prekonáva doteraz najrozšírenejší S-jazyk. Tento jazyk obsahuje okrem iných príkazov aj `^!IF` a `^!GOTO`, čo spolu s možnosťou definovania premenných umožňuje vytváranie plnohodnotných programov. Okrem toho, makrojazyk NoteTabu umožňuje neuveriteľne pohodlné vytváranie dialógových okien v rámci makier a je v tom skutočne neprekonateľný.
- Po tretie, makrá v NoteTabe je možné zlučovať do vytváraných knižníc makier a počas práce sa prepínať medzi existujúcimi knižnicami makier, editovať ich a vytvárať nové knižnice.
- Po štvrté, súčasťou editora je aj Clip Assistant („pomocník“) – súbor makier, ktoré umožňujú jednoduché programovanie nových makier aj laikovi. Clip Assistant môže byť zobrazený na pravom bočnom paneli. Sú to makrá, ktoré zjednodušujú vytváranie ďalších makier. (Opäť sa odvoláme na článok [2], kde bol realizovaný ten istý nápad – vytvoriť makrá pre tvorbu makier.)
- Po piate, súčasťou makra môže byť aj spustenie iného programu (napríklad, Perl, Gawk, Write, Word, Eudora, atď.), ktorý môže ďalej spracovať editovaný resp. zadaný súbor. Ak vezmeme do úvahy, že existujú voľne šíriteľné Perl a Gawk a obrovské množstvo takisto voľne šíriteľných skriptov k nim, užívateľ NoteTabu získava skoro neobmedzené možnosti pre spracovanie akýchkoľvek textových súborov. NoteTab vie komunikovať aj s aplikáciami typu konzola, vrátane získavania chybových hlásení týchto programov s možnosťou reagovania v rámci makra.

Treba poznamenať, že NoteTab neumožňuje zaznamenávanie postupnosti stlačených klávesov podobne tomu, ako to poznáme napríklad z T602, CSEDu, Macro Recorderu pre Windows 3.x alebo Microsoft Wordu 6, 7 a 97. Túto vlastnosť však perfektne nahrádza príkaz `^!KEYBOARD` makrojazyka NoteTabu. Stačí iba poznať klávesové skratky pre jednotlivé položky menu a podmenu (mimochodom, tie treba poznať aj v prípade zaznamenávania makier v spomenutých editoroch, preto tu žiadny principiálny rozdiel v skutočnosti nie je). Napríklad, príkaz

```
^!KEYBOARD ALT+M L S A
```

zodpovedá postupnému zvoleniu položiek menu *Modify – Lines... – Sort... – Ascending*.

Vykonávanie tohto príkazu privedie k vzostupnému usporiadaniu označených riadkov textu (zoznam mien, názvy tovarov, atď.).

Niekoľko knižníc makier je dodávaných spolu s programom, vrátane knižníc pre tvorbu HTML strán. Okrem nich sú na URL `http://www.notetab.com` zdarma k dispozícii knižnice, vytvorené užívateľmi NoteTabu pre prakticky všetko, čo čitateľov môže napadnúť, vrátane programovania v jazykoch C, Pascal, Visual Basic, Delphi, Java, Javascript, Perl, Gawk, HTML, TEX/LATEX/AMSLATEX, Oracle SQL, COBOL, atď.

Z hľadiska uľahčenia tvorby makier a ich používania treba spomenúť aj to, že NoteTab 4.6 ako jeden z mála textových editorov umožňuje zapínať a vypínať zobrazovanie niektorých dôležitých symbolov (medzier, koncov riadkov, koncov strán, tabelátorov).

Príklad – zlepovanie delených slov

Pre ilustráciu krásy, jednoduchosti a stručnosti makrojazyka editora NoteTab uvedieme jednoduchý, ale v česko-slovenských končinách dôležitý príklad, súvisiací s problémom, ktorý často pozorujeme najmä v novinách a časopisoch: uprostred riadku sa vyskytujú slová s rozdeľovacím znamienkom, ktoré by tam byť nemalo, napríklad „`pozo-rujeme`“ a podobne. Takéto slová vznikajú pri importovaní textu, v ktorom pôvodne boli slová rozdelené do dvoch riadkov. S podobným problémom zápasia aj tvorcovia HTML strán, ktorí upravujú dodané texty s delenými slovami.

V editore NoteTab na „zlepovanie“ všetkých delených slov naraz môže poslúžiť makro, obsahujúce iba dva príkazy:

```
^!KEYBOARD CTRL+A ALT+M L T
^!REPLACE "-^p" >> "" WATS
```

Prvý príkaz tohto makra znamená označenie celého dokumentu (`CTRL+A`) a vyvolanie položky menu *Modify – Lines – Trim* (`ALT+M L T`). Takto budú odstránené prípadné medzery na konci riadkov a bezprostredne za rozdeľovacím znamienkom sa bude nachádzať iba koniec riadku (`CR`, `LF`, alebo `CR+LF` podľa pôvodného formátu editovaného súboru). Druhý príkaz znamená nahradenie reťazca „`Rozdeľovacie Znamienko+Koniec Riadku`“ (`"-^p"`) prázdnym reťazcom (`""`) všade (parameter „`W`“ – *Whole document*, parameter „`A`“ – *All*, parameter „`T`“ – *sTring*), pričom skutočnosť, že hľadaný reťazec v danom texte neexistuje (aj to sa občas stáva), nebude ani oznamovaná (parameter „`S`“ – *Silently*). Znak „`^p`“ znamená koniec riadku, nech to bude `CR`, `LF`, alebo ich kombinácia.

Nedostatky

NoteTab má niektoré nedostatky, ktoré sú typické pre Microsoft Windows všeobecne, a teda aj pre iné textové editory pre Windows. Keďže sa o tom spravidla nehovorí a preto sa často ani nevie, povieme o nich aspoň v tomto článku.

Prvým problémom je nedokonalá podpora národných jazykov zo strany Microsoft Windows, najmä ak ide o triedenie a premenu malých písmen na veľké alebo naopak.

Slová, obsahujúce písmená s diakritikou, sú zatrieďované nesprávne. Tento problém môžno jednorazovo a čiastočne obísť tak, že sa v Control Panel / Regional Settings ako implicitný jazyk nastaví požadovaný jazyk a systém sa reštartuje. Nasledujúce triedenie prebehne viac-menej správne, ale niektoré chyby aj tak ostanú. Napríklad v prípade ruštiny bude nesprávne zaraďované malé aj veľké písmeno „jo“. V prípade slovenčiny nebude zohľadnené, že „ch“ je jedno písmeno a nie dve — nevie to ani Microsoft Word. Podobné problémy ostanú aj v prípade iných jazykov krajín centrálnej a východnej Európy. Uvedené čiastočné riešenie môže byť považované iba za núdzové, nakoľko zmena implicitných regionálnych nastavení a nasledujúci reštart systému život a prácu znepríjemňuje – veď treba stále vracať aj pôvodné nastavenia!

Zmena malých písmen na veľké (capitalisation) funguje vo Windows tiež s chybami. Napríklad ruské malé „ja“ sa premení na písmeno inej znakovej sady ako ruská abeceda. Táto chyba sa nedá odstrániť pre Windows všeobecne, ale v NoteTabe samotnom je možné vytvoriť makro, ktoré bude správne nahradzovať malé písmená veľkými v označenom textovom bloku (samozrejme, s využitím príkazu `^!REPLACE`).

Treba však dodať, že ani tieto nedokonalosti nemôžu zatieniť kladné vlastnosti editora NoteTab ako celku.

A čo za to?

Program NoteTab Pro 4.6 je shareware a užívateľ má právo rozhodnúť sa pre nadobudnutie licencie (registráciu) až po vyskúšaní editora v priebehu jedného mesiaca. Cena NoteTab Pro 4.6 v porovnaní s inými editormi, ktoré poskytujú oveľa menej možností a pohodlia, je v podstate polovičná – 19,95 USD oproti bežným 40 alebo 60 dolárom, nehovoriac o 119 USD za 32-bitovú verziu editora TSE (`http://www.semware.com`), ktorý je považovaný za pravdepodobne najlepší editor typu konzola pre programátorov. NoteTab Standard 4.6 stojí ešte menej – 9,95 USD.

Pozornosť si zaslúži aj verzia Light, ktorá je freeware. Neobsahuje však korektor pravopisu a synonymický slovník, a tiež neumožňuje vytváranie „outline“ dokumentov, ktoré však vie zobrazovať. Vo verzii Light pochopiteľne nie

sú sprístupnené niektoré príkazy makrojazyka NoteTabu, čo podstatne obmedzuje možnosti tvorby makier, a nie je dostupná možnosť „HTML-highlighting“.

Je možné vyskúšať si verziu Light a neskôr presedlať na verziu Pro alebo Standard. Je možné začať skúšobnou verziou Pro alebo Standard a povedať si: stačí mi aj freeware verzia Light. V obidvoch prípadoch skúška stojí za to, najmä ak často a radi pracujete s textom v jeho čistej a WYSIWYGom nepoškvrnenej podobe a potrebujete kvalitný editor s možnosťou jednoduchej tvorby dokonalých makier.

Záver

Všetky dobré vlastnosti NoteTabu v podstate predurčujú, že podobné možnosti budú zrejme čoskoro poskytovať aj iné textové editory bežiace pod Windows. Stačí sa letmo pozrieť na vonkajšiu podobu a rozsah možnosti EditPlus (`http://www.editplus.com`), TextPad (`http://www.textpad.com`), WinEdt (`http://home.istar.ca/~winedt`), MultiEdit (`http://www.multiedit.com`), UltraEdit (`http://www.ultraedit.com`) – a hneď vidieť, že NoteTab už je na svojej ceste „prenasledovaný“ textovými editormi, ktoré sú mu v určitých črtách až neuveriteľne podobné.

Na záver môžeme konštatovať, že NoteTab veľmi výrazne prispel a prispieva k nástupu novej generácie textových editorov pre Windows a je zatiaľ jej najlepším predstaviteľom. Bude zaujímavé sledovať, či a kto ho prekoná ...

Odkazy

[1] Podlubný I.: Nový štandard textových editorov pre Windows? *COMPUTERWORLD* č.8/99, príloha *Technology World*, s.3.

[2] Podlubný I.: Makrá: Turbo Turbo, *PC WORLD* 6/94, s.110–114.

Igor Podlubný
Katedra riadenia výrobných procesov
Fakulta BERG
Technická univerzita v Košiciach
`podlbn@ccsun.tuke.sk`

Katarína Kassayová
Ústav fyziológie
Lekárska fakulta
Univerzita P. J. Šafárika v Košiciach

LaTeX2eps clipbook — knižnica makier pre textový editor NoteTab

Igor Podlubný

Je všeobecne známe, že pri vytváraní TEX-ovských dokumentov užívateľ trávi väčšinu času editovaním zdrojového TEX-ovského alebo LATEX-ovského kódu. Zefektívnenie práce s editorom znamená celkové urýchlenie práce na projekte. Jedným zo spôsobov, ako dosiahnuť takéto urýchlenie, je používanie užívateľských textových makier, ak textový editor umožňuje ich tvorbu.

Definovanie vlastných makier však spravidla nie je jednoduchá záležitosť – kvôli zložitosti príkazov, ktoré sa pre definovanie makier používajú. Niektoré editory na to používajú takzvaný S-jazyk, ale ten nie je vhodný pre vytváranie makier takpovediac „za jazdy“.

Iný prístup k zefektívneniu práce predstavujú zaznamenávané (nahrávané) postupnosti stlačení klávesov (klávesové makrá), ktoré pomáhajú urýchliť „naťukávanie“ textu alebo rutinné opakovanie činností, zastúpených položkami menu editora.

Okrem písania však existuje ešte niečo: pripravovaný text často predstavuje zdrojový text v TEXu, LATEXu, alebo napríklad v akomkoľvek programovacom jazyku. V takom prípade je vhodné mať možnosť spúšťať preklad zdrojového textu a prezerať výsledky spracovania bez toho, aby bolo nutné opustiť textový editor. Inými slovami, dobrý editor by mal poskytovať možnosť jeho využitia ako ľahko konfigurovateľného integrovaného vývojového prostredia, čo zahŕňa aj schopnosť spúšťať externé programy pre vykonávanie súvisiacich úloh.

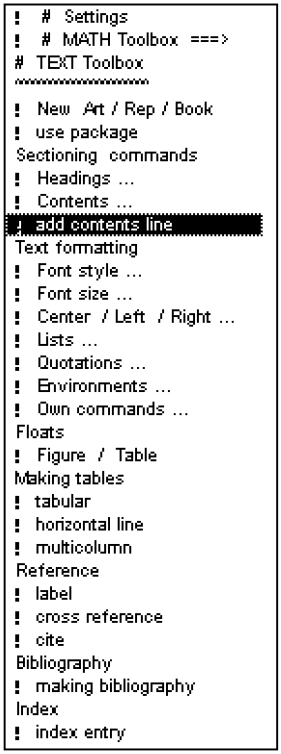
Existujúce editory v rôznej miere poskytujú možnosť programovania makier, možnosť zaznamenávania klávesových makier a možnosť spúšťania externých programov. Avšak ako jeden z najpohodlnejších textových editorov s uvedenými vlastnosťami sa javí editor NoteTab Pro 4.51. Jeho jednoduchý a výkonný jazyk pre tvorbu makier, orientovaný na prácu s textmi a pokrývajúci všetky tri spomenuté oblasti požiadaviek, umožnil vytvorenie balíka makier pre písanie LATEX-ovských dokumentov, ktorý je predstavený v tomto článku.

Predstavovaný balík **LaTeX2eps** poskytuje veľa funkcií, ktoré sa v makrách pre iné editory nevyskytujú alebo sa vyskytovať nemôžu. Jednotlivé makrá sa správajú rôzne v závislosti od nastavení alebo od toho, či je označený nejaký textový blok alebo nie je. Tento balík makier je možné prispôsobiť každému konkrétnemu projektu s jeho špecifickými príkazmi, názvami prostredí, súborom použitých bibliografických databáz, atď. Je vytvorený tak, aby ho bolo možné použiť v súčinnosti s akoukoľvek implementáciou systému TEX, ktorá

beží pod Windows 95/98/NT v normálnom alebo v DOS-ovskom okne. Z tohto dôvodu makrá, ktoré by zabezpečili spúšťanie jednotlivých súčastí systému TEX (compiler, viewer, atď.) zahrnuté neboli – takéto makrá si užívateľ môže doplniť samostatne po oznámení sa s dokumentáciou k editoru NoteTab, kde je na dobrej úrovni popísané spúšťanie DOS-ovských a Windows-ovských programov v rámci makra.

Balík sa skladá z dvoch častí – **TEXT toolbox** a **MATH toolbox**.

TEXT Toolbox



V prvých troch riadkoch sa nachádzajú makrá, ktoré sa používajú na zmenu parametrov režimu práce (# Settings) a na prepínanie medzi nástrojmi pre prácu v textovom alebo matematickom móde (# TEXT Toolbox, # MATH Toolbox). Symbol # znamená, že ide o ovládací prvok, ktorý mení nastavenia, ale nemení text pripravovaného dokumentu. Tretí riadok (podčiarknutý) nie je aktívny a slúži ako názov aktívneho toolboxu.

Prvá časť – **TEXT toolbox** – obsahuje makrá pre písanie príkazov LATEX 2_ε, ktoré sa používajú v textovom móde. Tieto príkazy sú rozdelené do nasledujúcich skupín:

- úvodné príkazy (New ... , use package)
- Sectioning commands – definovanie štruktúry a vytvorenie obsahu
- Text formatting – formátovanie textu
- Floats – prostredia `figure` a `table`
- Making tables – formátovanie tabuliek
- Reference – krížové odvolávky
- Bibliography – tvorba zoznamu literatúry
- Index – tvorba registrov slov, mien a označení.

Uvedené názvy sú farebne zvýraznené, aj keď to na obrázku nevidieť.

Niektoré skupiny obsahujú iba jedno makro – ale všetky makrá v knižnici **LaTeX2eps** sú mnohotvárne.

Uvedieme veľmi stručný prehľad jednotlivých skupín makier.

Sectioning commands

Tu sú zaradené makrá, súvisiace s definovaním štruktúry dokumentu.

Makro Headings... poskytuje na výber *prostredie* `abstract` a *príkazy* `\part`, `\chapter`, `\section`, `\subsection`, `\subsubsection`, `\appendix`. Okrem toho je možné uviesť, či číslovateľný prvok štruktúry má byť číslovaný alebo nie. Ak pred vyvolaním tohto makra bol označený textový blok, potom sa označený blok použije ako parameter príkazu alebo obsah prostredia.

Makro Contents... slúži na vkladanie príkazov súvisiacich s generovaním obsahu (`\tableofcontents`, `\listoffigures`, `\listoftables`, `\index`), pričom, ak bola zvolená možnosť multiple indices v makre # Settings, je ponúkaná tiež možnosť voľby typu vkladaného registra.

Makro add contents line slúži na vkladanie príkazu na zaradenie nejakého názvu do obsahu, zoznamu tabuliek alebo zoznamu obrázkov. Umožňuje zadať text pridávanej položky, zvoliť si typ obsahu a tiež úroveň štruktúry dokumentu, ktorej pridávaná položka zodpovedá (časť, kapitola, atď.).

Text formatting

Tu sú sústredené makrá, súvisiace s bežným formátovaním textu. Ak je označený nejaký textový blok, potom sa tento textový blok použije ako parameter vkladaného príkazu alebo ako telo vkladaného prostredia.

Makro Font style... umožňuje výber medzi dvoma režimami práce, ktoré sú symbolicky označené SIMPLE MODE a ADVANCED MODE. Prvý slúži na vkladanie príkazov typu `\emph{}`, `\textit{}`, `\textsc{}` atď., kým druhý umožňuje vkladanie príkazov typu `{\rmfamily\bfseries\itshape }`.

Vkladanie príkazov na zmenu veľkosti písma umožňuje makro Font size....

Makro Center/Left/Right umožňuje výber medzi prostrediami `center`, `flushleft`, `flushright`.

Makro Lists... sformátuje označený blok textu ako zoznam typu `itemize` alebo `enumerate`. Ak nebol označený žiadny textový blok, potom sa iba vloží prázdne prostredie zvoleného typu a jeden príkaz `\item`.

Makro Quotations... umožňuje výber medzi prostrediami, používané pre uvádzanie v texte fragmentov iného textu: `quote`, `quotation`, `verse`, `verbatim`. Ak pred spustením tohto makra bol označený textový blok, potom sa automaticky pridá LaTeX-ovské značkovanie v súlade so zvoleným prostredím. Napríklad, označenie nasledujúcich riadkov ako blok

```
Yesterday
All my troubles
seemed so far away

Now it looks as they are here to stay
Oh I believe in  yesterday
```

spustenie makra Quotations... a zvolenie typu `verse` ihneď dáva:

```
\begin{verse}
Yesterday\\
All my troubles\\
seemed so far away

Now it looks as they are here to stay\\
Oh I believe in  yesterday
\end{verse}
```

Makro Environments... ponúka možnosť výberu prostredia zo zoznamu štandardných a neštandardných LaTeX-ovských prostredí. Ak bol predtým označený textový blok, potom sa tento blok použije ako telo zvoleného prostredia. Je možné pridávať (ADD) a odstraňovať prostredia jednotlivo (REMOVE) alebo všetky naraz (CLEAR), triediť ich (SORT), a tiež načítavať (LOAD) zoznamy prostredí, vytvorené pri práci na iných LaTeX-ovských textoch.

Ak užívateľ potrebuje pridať makrá pre ďalšie štandardné alebo vlastné LaTeX-ovské príkazy, potom je tu makro Own commands..., ktoré umožňuje vytvorenie zoznamu vlastných príkazov s tými istými možnosťami úpravy (ADD, REMOVE, CLEAR, SORT, LOAD).

Floats

Táto skupina obsahuje iba jedno makro – Figure / Table, ktoré označený textový blok „zabalí“ v súlade s požiadavkami užívateľa.

V dialógovom okne je možné zvoliť typ číslovateľného prostredia (`figure` alebo `table`), spôsob sadzby (na šírku stĺpca alebo na šírku strany), požiadavky na umiestnenie objektu na strane (`top`, `bottom`, `here`, atď.), či má byť dovnútra vložené aj prostredie `center`. Je možné zadať aj popis obrázku alebo tabuľky (`\caption`). Na účely odvolávania sa v texte sa pre vkladaný objekt vyžaduje zadanie prezývky (`\label`) .

K zadanej prezývke (napríklad `Ceny`) sa automaticky pridáva zodpovedajúci prefix (napríklad `tab:` alebo `fig:`), a preto úplná prezývka objektu, podľa ktorej sa naňho dá odvolávať, bude `tab:Ceny` alebo `fig:Ceny`.

Making tables

Táto skupina obsahuje tri makrá, ktoré sú vhodné pre prípravu tabuliek (prostredie `tabular`).

Makro tabular je možné používať dvoma spôsobmi.

- Ak nie je označený žiadny textový blok, potom sa objaví dialógové okno, umožňujúce zadanie počtu riadkov a stĺpcov tabuľky, a do textu bude vložená šablóna v súlade so zadanými požiadavkami.

- Ak je ako textový blok označený vhodne pripravený „polotovar“, potom sa iba pridá LaTeX-ovské základné značkovanie. Napríklad, označenie nasledujúcich riadkov ako blok

```
Igor Podlubný      10
Ľubomír Dorčák     20
```

  a spustenie makra tabular okamžite dáva nasledujúci výsledok:

```
%
\begin{tabular}{ll}
Igor Podlubný   & 10 \\
Ľubomír Dorčák  & 20 \\
\end{tabular}
```

  V „polotovare“ sa jednotlivé stĺpce od seba oddeľujú dvoma alebo viacerými medzerami alebo aspoň jedným tabelátorom. Jedna medzera, ako vidieť z práve uvedeného príkladu, nie je pokladaná za oddeľovač stĺpcov.

Makro hline jednoducho pridáva riadok s príkazom `\hline` za riadok, na ktorom sa práve nachádza kurzor.

Makro multicolumn umožňuje zadanie počtu stĺpcov, ktoré sa majú zlúčiť pre zobrazenie ďalšieho políčka tabuľky, a tiež základné rozmiestnenie textu v tomto políčku (centrovať, zarovnať vľavo, zarovnať vpravo).

Iné príkazy, používané v prostredí `tabular`, je možné pridať k vlastným príkazom (makro Own commands...) alebo doplniť ručne.

Reference

Táto skupina obsahuje makrá pre uľahčenie práce s krížovými odvolávkami.

Makro label umožňuje zadanie novej prezývky a kategórie. Z prefixu kategórie a zadaného textu sa vytvára prezývka pre odvolávky v texte (napríklad `eq:Newton law`). Každá novovytvorená prezývka sa testuje na jedinečnosť a v prípade, ak sa takáto prezývka už vyskytla, ponúka sa možnosť zadať inú prezývku.

Makro cross-reference zobrazuje zoznam všetkých použitých prezývok a vkladá zodpovedajúci príkaz `\ref`.

Správanie sa makra cite závisí od nastavení (makro # Settings).

- Ak bolo nastavené používanie bibliografických databáz, potom sa najprv zobrazí zoznam zvolených databáz. Po zvolení databázy sa objaví zoznam prezývok prác z tejto databázy. Takto je možné zvoliť niekoľko prác z rôznych databáz, ktoré majú byť uvedené v spoločnom príkaze `\cite`.
- Ak bolo nastavené, že sa bibliografické databázy používať nebudú, potom sa zobrazuje zoznam prác, uvedených v prostredí `thebibliography` alebo jemu podobnom.

V obidvoch prípadoch je možné zadať komentár (parameter príkazu `\cite`) a zvoliť si spôsob citovania (`\cite` alebo `\nocite`).

Bibliography

Táto skupina obsahuje iba jedno makro – making bibliography, ktoré však má niekoľko podôb v závislosti od nastavení, zvolených v makre # Settings.

1. Ak bolo zvolené `yes` v políčku `Use bibliography databases` v makre # Settings, potom sa objavuje dialógové okno, umožňujúce zadanie bibliografického štýlu a mena (mien) použitých databáz; do dokumentu sa vložia príkazy `\bibliographystyle` a `\bibliography` s takto zvolenými parametrami.
   V takom prípade sa nové bibliografické položky ďalej pridávajú iba do databáz a odvolávky na nich sa uskutočňujú pomocou makra cite.
2. Ak bolo zvolené `no` v políčku `Use bibliography databases` v makre # Settings, potom rozlišujeme dve možné situácie:
   - Ak sa pracuje na projekte, obsahujúcom niekoľko súborov, potom sa v makre # Settings zvolí typ vstupu
     `main file + several \include'd and \input'ed files`.
     V takom prípade makro making bibliography vkladá do textu príkaz `\bibitem`, pričom sa testuje jedinečnosť zadávanej prezývky.
   - Ak sa pracuje na projekte v podobe jedného LaTeX-ovského súboru, potom sa v makre # Settings zvolí typ vstupu `one input file`. V takom prípade makro making bibliography pridáva prostredie `thebibliography`, ak je to potrebné; ak toto prostredie už je prítomné, potom sa pridáva príkaz `\bibitem` na koniec prostredia `thebibliography`. Pridané položky bibliografie sú okamžite dostupné pre odvolávky prostredníctvom makra cite.

Treba poznamenať, že používanie tohto makra (ako aj viacerých iných) je jednoduchšie, ako jeho popis.

Index

Skupina Index tiež obsahuje iba jedno makro – index entry, ktoré poskytuje všetko, čo je potrebné pre jednoduchú prípravu jedného alebo až troch registrov (register termínov, register mien a register označení).

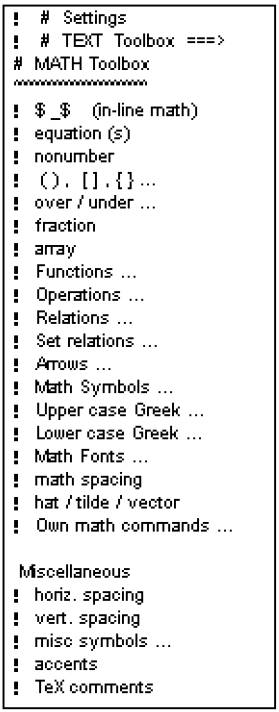
Toto makro umožňuje prostredníctvom dialógového okna zadať

- textový reťazec, určujúci miesto položky v registri,
- text, ktorý sa objaví v registri,
- typ registra (slová, mená, označenia),
- rozsah strán (aktuálna strana, začiatok rozsahu strán, koniec rozsahu strán),

- spôsob formátovania čísel strán (`bold`, `italic`, `roman`).

Na záver sa do textu vkladá príkaz `\index`, vytvorený v súlade s uskutočnenými voľbami.

MATH Toolbox



V prvých troch riadkoch sa nachádzajú makrá, ktoré sa používajú na zmenu parametrov režimu práce (# Settings) a na prepínanie medzi nástrojmi pre prácu v textovom alebo matematickom móde (# TEXT Toolbox, # MATH Toolbox). Symbol # znamená, že ide o ovládací prvok, ktorý mení nastavenia, ale nemení text pripravovaného dokumentu. Tretí riadok (podčiarknutý) nie je aktívny a slúži ako názov aktívneho toolboxu.

MATH toolbox obsahuje makrá pre písanie príkazov LaTeX 2_ε, ktoré sa používajú v matematickom móde.

Časť označená Miscellaneous obsahuje rôzne dodatočné užitočné makrá (príkazy na vkladanie vodorovných a zvislých medzier, rôzne symboly, akcenty, komentovanie / odkomentovanie riadkov).

Väčšina makier v tomto toolboxe umožňuje zvolenie potrebného príkazu LaTeX 2_ε z menu, ponúkaného v dialógovom okne. K takým makrám patrí Functions..., Operations..., Relations..., Set relations..., Arrows..., Math Symbols..., Upper case Greek..., Lower case Greek..., Math Fonts..., math spacing. Ak je označený textový blok, potom sa tento blok spravidla používa ako argument zvoleného príkazu.

Zvyšné makrá stručne popíšeme jednotlivo.

Použitie makra $_$ (in-line math) je zrejmé. Ak je označený textový blok, potom sa tento blok vloží medzi symboly $.

Dialóg v makre equation(s) umožňuje zadať počet rovníc (1, 2, 3, ...) a či majú byť tieto rovnice číslované. V závislosti od zvolených požiadaviek sa do textu vkladá šablóna pre prostredie `equation`, `displaymath`, `eqnarray` alebo `eqnarray*`.

Makro nonumber/number sa správa rôzne v závislosti od aktuálnej situácie.

- Ak nie je označený žiadny textový blok, potom sa do textu iba vkladá príkaz `\nonumber`.
- Ak je označený textový blok, obsahujúci iba číslované prostredia

(`equation` a `eqnarray`), potom všetky tieto prostredia budú zmenené na prostredia nečíslované (`displaymath` a `eqnarray*`).

- Ak je označený textový blok, obsahujúci iba nečíslované prostredia, potom tieto prostredia budú zmenené na prostredia číslované.
- Ak označený blok obsahuje prostredia oboch typov, potom bude ponúknutá možnosť výberu typu konverzie prostredí.

Makro ( ), [ ], { }... slúži na vkladanie príkazov, zodpovedajúcich rozličným zátvorkám, tak párovaným, ako aj nepárovaným, vrátane zátvoriek premenlivej veľkosti (ako napríklad `\left\{` a `\right\}`).

Makro over/under... umožňuje vkladanie jedného z príkazov `\underline`, `\overline`, `\underbrace`, `\overbrace`.

Makro fraction je možné používať štyrmi spôsobmi.

- Ak nie je označený textový blok, potom toto makro vkladá príkaz `\frac{}{}`.
- Ak je označený textový blok typu `2/3`, potom výsledok použitia makra fraction bude `\frac{2}{3}`.
- Ak sú označené dva riadky, napríklad

```
\alpha + 1
\beta - 3
```

  potom výsledok bude

```
\frac{\alpha + 1}
     {\beta - 3}
```

- Ak označený blok obsahuje viac ako dva riadky, potom iba prvý riadok sa použije ako čitateľ a ostatné riadky budú interpretované ako menovateľ.

Makro array sa používa podobne ako makro tabular.

- Ak nie je označený žiadny textový blok, potom sa objaví dialógové okno, umožňujúce zadanie počtu riadkov a stĺpcov, a do textu bude vložená šablóna v súlade so zadanými požiadavkami.
- Ak je ako textový blok označený vhodne pripravený „polotovar“, potom sa iba pridá LaTeX-ovské základné značkovanie. Napríklad, označenie nasledujúcich riadkov ako blok

```
2\alpha + 3     (\alpha > \beta)
\beta -4        (\alpha < \beta)
```

  a spustenie makra array okamžite dáva nasledujúci výsledok:

```
%
\begin{array}{ll}
2\alpha + 3    & (\alpha > \beta) \\
```

```
\beta -4       & (\alpha < \beta) \\
\end{array}
```

V „polotovare“ sa jednotlivé stĺpce od seba oddeľujú dvoma alebo viacerými medzerami alebo aspoň jedným tabelátorom. Jedna medzera, ako vidieť z práve uvedeného príkladu, nie je pokladaná za oddeľovač stĺpcov.

Makro hat/tilde/vector slúži na umiestňovanie uvedených značiek nad písmená alebo skupiny písmen. Ak nie je označený textový blok, uvedú sa všetky možné (úzke aj široké) varianty. Ak je označené jedno písmeno, široké varianty (napríklad, `\widehat`) budú vynechané. Naopak, ak označených písmen je viac ako jedno, potom budú vynechané úzke varianty a ponúknuté iba široké.

Makro Own math commands... je podobné makru Own commands... a umožňuje vytvorenie zoznamu vlastných príkazov s tými istými možnosťami úpravy (ADD, REMOVE, CLEAR, SORT, LOAD).

Z makier, zaradených do skupiny Miscellaneous, spomenieme makro TeX comments, ktoré umožňuje komentovanie alebo odkomentovanie označených riadkov LaTeX-ovského zdrojového textu.

Makrá horiz. spacing a vert. spacing umožňujú vkladanie príkazov pre definovanie medzier pevnej a premenlivej dĺžky, vrátane medzier definovaných užívateľom.

Ďalšie informácie

Podrobný popis všetkých makier knižnice **LaTeX2eps** (v angličtine) je dostupný v HTML podobe spolu s obrázkami, znázorňujúcimi jednotlivé dialógové okná v rámci makier, na URL `http://www.tuke.sk/podlubny/latex2eps`. Súčasťou tejto on-line príručky sú aj rady pre efektívne využívanie makier **LaTeX2eps**.

Igor Podlubný
Katedra riadenia výrobných procesov
Fakulta BERG
Technická univerzita v Košiciach
podlbn@ccsun.tuke.sk

Zvýraznění bloků textu s možností stránkového zlomu

Vít Zýka

Tímto článkem se snažím oslovit dva druhy čtenářů. V prvé části popisuji – pro milovníky cizích makrobalíčků – použití maker `backgr`. Ta zvýrazňují bloky textu (např. rámečkem, podbarvením, svislou linkou), které mohou podléhat stránkovému zlomu. Makra pracují ve formátu plainTEX i LATEX. Ve druhé části textu vysvětluji – pro hlubinné TEXové hloubaly – způsob řešení tohoto problému. Popisuji omezení této implementace a ukazuji obtížnost realizace této třídy úloh prostřednictvím TEXu.

1. Zvýraznění bloku textu

Blokem textu rozumíme v tomto článku část sazby, u které můžeme vertikálně označit její začátek a konec. Tento blok může být libovolně dlouhý, např. jeden box, jeden odstavec, nebo několik stránek. Kromě hladkého textu může obsahovat nadpisy, matematické rovnice, poznámky pod čarou, obrázky, tabulky a jiný sazební materiál.

Tímto problémem jsem se začal zabývat v souvislosti se sazbou knihy [2], která obsahuje kapitoly, ve kterých autoři diskutují s domnělým studentem. Bylo žádoucí, aby odstavce, ve kterých promlouvá student, byly zřetelně odlišeny od ostatního textu. Nejprve mě napadlo změnit řez písma na písmo skloněné. Autorům se však zdálo toto rozlišení příliš málo nápadné. Uvažovali jsme tedy o následujících způsobech zvýraznění:

- výraznější znaková sada stejné písmové rodiny,
- změna šířky tiskového zrcadla,
- symbolem (iniciálou) na začátku bloku,
- pruhem vedle textu (např. vlnitou čarou),
- podbarvením textu (např. světlou šedí).

Poslední bod se zdál být natolik zajímavý, že jsem se začal ohlížet po jeho řešení, aby mohl být posouzen jeho funkční i estetický účinek[1]. Nalezené techniky se však omezovaly jen na podbarvení nezlomitelného boxu. Přistoupil jsem tedy

[1] Ukázalo se, že je třeba rozlišovat zvýraznění a odlišení textu. Např. šedý podklad upoutá na první pohled a čtenář má dojem, že jde o nejdůležitější pasáže knihy. V našem případě bylo třeba text odlišit, ne zvýraznit. Šedý podklad tedy nakonec nebyl použit.

k napsání vlastních maker nazvaných `backgr`, která rozdělí zvýrazněný text na části podle stránkového zlomu. Tyto části se pak zvýrazňují samostatně.

2. Makra `backgr` z pohledu uživatele

Popis se vztahuje k balíčku maker `backgr` verze 1.0 z května 1999. Je k dispozici na adrese `http://cmp.felk.cvut.cz/~zyka/zykatex.html`.

2.1. Použití

Plain TEX

Načteme makra `\input backgr.tex` a označíme začátek zvoleného bloku textu pomocí značky `\backgr` a konec značkou `\endbackgr`.

LATEX 2ε i LATEX 2.09

Způsob značení z plainu funguje i v LATEXu, kromě něj je definováno i prostředí `backgr`. Lze tedy použít obvyklou syntaxi LATEXu uzavřením bloku do dvojice `\begin{backgr}` a `\end{backgr}`.

2.2. Specifikace umístění značek začátku a konce bloku

Značky začátku nebo konce bloku můžeme umístit pouze do hlavního vertikálního módu [1, str. 85]. Značka vložená do odstavcového horizontálního módu nejdříve automaticky uzavře odstavec příkazem `\par` a přejde tak do žádoucího hlavního vertikálního módu. V ostatních módech nelze značek použít. Zvýrazněné bloky nelze do sebe vnořovat.

Logická proměnná `\ifbackgrstrut` ovlivňuje vertikální zarovnání zvýraznění. Pravdivá (přednastavená) hodnota `\backgrstruttrue` způsobí sazbu zvýraznění o hloubku „strut“ výplňku (hloubku kulaté závorky) níže než poslední účaří. Při `\backgrstrutfalse` se poloha zvýraznění kryje s hloubkou předcházejícího boxu.

2.3. Ruční zpřesnění polohy

Z níže uvedených důvodů není vertikální usazení zvýraznění vždy správné. Proto je třeba při konečné úpravě dokumentu, kdy již nebude měněn stránkový zlom, ručně posunout některé začátky a konce zvýraznění. Provádí se to pomocí registru typu ⟨*dimen*⟩ `\backgrcorrection` před vlastní značkou bloku. Např. nastavení

```
\backgrcorrection=1mm      % LaTeXový zjednodušený zápis:
\backgr                          %   \Backgr[1mm]
  Blok textu učený ke zvýraznění.
```

```
\backgrcorrection=-.5ex
\endbackgr                          %   \endBackgr[-.5ex]
```
posune začátek zvýraznění o jeden milimetr níže, jeho konec o půl střední výšky písma výše, než je vypočtená hodnota. Po provedení značky `\backgr` nebo `\endbackgr` (i jejich LaTeXových ekvivalentů) je hodnota `\backgrcorrection` vrácena na velikost v registru `\backgrconstcorr`, jehož počáteční rozměr je $0\,pt$. Toto nastavení způsobí, že další značka nebude posunuta vůči vypočtené hodnotě, nenastavíme-li znova parametr `\backgrcorrection` na nenulový rozměr. Registrem `\backgrconstcorr` lze korigovat více značek o stejně velkou vzdálenost najednou.

Dojde-li uvnitř zvýrazněného bloku textu k přechodu na novou stránku, vytvoří se v místě zlomu dvě pomocné značky automaticky. Jedna na spodní hraně strany a druhá na začátku strany následující. Chceme-li korigovat polohu těchto pomocných značek, nastavíme registry `\backgrbotcorrection` a `\backgrtopcorrection` na žádoucí rozměry.

2.4. Definice způsobu zvýraznění

Balíček `backgr` obsahuje definice následujících způsobů zvýraznění bloku textu nazývaných *styly zvýraznění*:

zigzag – klikatá linka vedle textu,
frame – rámeček okolo textu (přednastaveno),
gray – šedé pozadí (vyžaduje PostScriptový ovladač nebo pdfTeX),
grayframe – šedé pozadí s rámečkem (PostScriptový ovladač nebo pdfTeX),
empty – žádné zvýraznění (pro dočasné vypnutí).

Lze je měnit makrem, jehož jméno vznikne spojením řetězce `\bgstyle` a názvu stylu. Např. světle šedého podbarvení textu dosáhneme makrem `\bgstylegray`. Změna stylu musí být provedena před značkou začátku zvýraznění.

Chceme-li si vytvořit vlastní styl, nadefinujeme makro s šesti parametry. Ty mají následující význam:

1. Logická proměnná, která udává, zda blok textu pokračuje z předcházející strany. Hodnotu zjistíme testem na shodnost s primitivem `\relax`.
2. Šířka zvýraznění. Je rovna hodnotě makra `\bgwd` při použití značky začátku bloku. Přednastavená hodnota je rovna šířce tiskového zrcadla.
3. Horizontální odsazení vůči levému okraji textu. Rovno okamžité hodnotě makra `\bgind`, přednastaveno $0\,pt$.
4. Vertikální vzdálenost horního okraje zvýrazňujícího boxu od horního okraje stránky.
5. Výška zvýraznění.
6. Logická proměnná udávající, zda blok textu pokračuje i na další stranu.

Důležité je, aby makro neposunulo referenční bod sazby, tj. aby zvýrazňující materiál byl sázen ve vertikálním boxu nulového rozměru. Tento box pak výstupní

rutina umístí na začátek sázené stránky. Příklady definic stylů najde čtenář ve zdrojovém souboru `backgr.tex`. Přepnutí na náš nový styl zařídíme následující definicí `\def\backgrstyle{\noexpand` ⟨*makro realizující náš styl*⟩`}`.

Pokud chceme ovlivnit vlastní zvýrazněný text, můžeme předefinovat makra `\bgbuser` a `\bgeuser`. Ty se expandují v místě značky `\backgr`, respektive `\endbackgr`. Například pro styl `frame` je vhodné nastavit

```
\def\bgbuser{\bgroup\leftskip=4pt \rightskip=4pt
             \advance\parindent by-4pt }
\def\bgeuser{\egroup}
```

Rámeček pak může být sázen na šířku tiskového zrcadla, aniž by byl text nalepen na rámeček.

2.5. Kdy makra selžou

Automatické umístění zvýraznění pracuje zcela přesně v hladkém textu. V následujících případech může dojít k odchylce, kterou je možné ručně kompenzovat. Způsob nápravy je uveden italikou.

- Je-li v hlavním vertikálním seznamu `\mark` nebo `\write`. *Použij* `\backgrcorrection`*, viz. 2.3.*
- Je-li vložen plovoucí insert. Některé typy insertů jsou poměrně složitě a neelegantně ošetřeny, ale např. v současné verzi dojde k hrubé chybě, odsune-li se insert až na další stranu, než byl vložen. *Vlož insert až na té straně, kam byl odsunut.*
- Je-li značka zpracována na konci strany, ale již se tam nevešla. *Zlom ručně stranu (*`\eject` *nebo* `\newpage`*) v místě, kde to udělal TEX sám.*

Představu o úspěšnosti správného umístění zvýraznění si lze udělat z následujícího reálného textu: v něm z celkového počtu 172 značek bylo třeba 26 poopravit.

3. Makra `backgr` z pohledu řešitele

Tato sázecí úloha může být řešena dvěma způsoby:

Synchronně – mezi vstupní materiál se vloží značky a zároveň s textem se zpracovává jeho zvýraznění.

Problémy úlohy *a problémy TEXu*:

- Řešení systémově závislé. *Zvýraznění musí umět použitý* `dvi` *ovladač. Nejsnáze snad vyhoví PostScriptové specialy.*

Asynchronně – změří se velikost a poloha zvýrazněného bloku textu a pak se zvlášť vysází text a zvlášť jeho zvýraznění. Důležité je zajistit, aby obě části na sobě přesně seděly.

Problémy úlohy *a problémy TEXu*:

- Rozdělení bloku přesahujícího jednu stranu na více částí podle stránkového zlomu. *Expanze maker může probíhat na jiné straně, než*

sazba materiálu obsažených v těchto makrech.

- Počítat s pružnými výplňky hlavního vertikálního módu. *Nelze přímo zjistit součet pružných výplňků do místa optimálního, TEXem nalezeného, zlomu strany.*
- Vyrovnat se s plovoucími objekty typu insert. *Insert se může objevit na začátku strany (tj. před značkami, které do strany již vstoupily i těmi, které teprve přijdou) i na jejím konci (tj. za značkami) a také může odplout na následující stranu.*

Protože rozumné synchronní řešení vede na ne zcela triviální použití PostScriptu (a ten znám podstatně méně než jazyk TEXu) a protože jsem si neuvědomoval nastíněné omezení TEXu, zvolil jsem asynchronní implementaci. V dalším textu vysvětlím její nedostatky.

3.1. Nástin řešení

Makra `backgr` pracují ve dvou fázích. V první expandují značky vertikálně ohraničující zvýrazněný text. Druhá fáze probíhá ve výstupní rutině a realizuje vlastní sazbu boxu se zvýrazňujícím materiálem. Podívejme se podrobněji, co je třeba v každé fázi udělat.

1. Značky bloku textu ke zvýraznění. Každá značka způsobí, že TEX zjistí výšku sazby na aktuální straně pomocí primitivů `\pagetotal` a jemu podobných a zapíše ji do ⟨*token*⟩ registru `\backgrlist`. Značka `\backgr`, otevírající zvýrazněný text, dále nastaví logickou proměnnou `\ifmiddlebg` na pravdivou hodnotu, uzavírací značka `\endbackgr` na nepravdivou. Víme tak v každém místě sazby, zda jsme uvnitř zvýrazněného bloku nebo ne.
2. Výstupní rutina. V ní provedeme následující kroky:
   (a) Uzavření boxu. Pokud je `\ifmiddlebg` pravda, pak zvýrazněný blok textu přechází z kompletované strany na další. Uzavřeme tedy blok na této straně vložením značky `\endbackgr` s takovými parametry, jako by byla na dolním konci tiskového zrcadla aktuální strany (dáno výškou a hloubkou boxu 255).
   (b) Tisk zvýraznění. Vysází se obsah registru `\backgrlist`. Každý pár otevírací a uzavírací značky do něj přidal makro s názvem `\backgrbox` s množstvím parametrů o aktuálním místě sazby. Toto makro spočítá usazení zvýrazňujícího boxu na straně. Poté dle nastaveného stylu zvýraznění vysází (před vlastní obsah strany) zvýrazňující vbox nulových rozměrů. Po sazbě zvýraznění se registr `\backgrlist` vyprázdní.
   (c) Otevření boxu. Pokud je `\ifmiddlebg` pravda, pak zvýrazněný text pokračuje z předchozí strany. Proto do `\backgrlist` vložíme značku otevření bloku s nulovými parametry výšky aktuálního bodu sazby.

3.2. Bod sazby a nestejnostranná expanze a sazba

Jaké nám TEX nabízí informace o aktuální pozici sazby na kompletované straně? Jsou to údaje uložené v registrech začínající jménem `\page`. Zde je jejich přehled:

`\pagetotal` (t) popisuje výšku dosud vloženého materiálu, tedy součet výšek všech boxů, linek, kernů a mezer typu ⟨*glue*⟩.

`\pagegoal` udává cílovou výšku materiálu. Je roven `\vsize`, ale vložením každého insertu je zmenšena o jeho skutečnou výšku.

`\pagedepth` (d) je hloubka posledního materiálu; je-li jím kern nebo mezera typu ⟨*glue*⟩, je roven 0 *pt*.

`\pageshrink` shromažďuje stažení všech registrů typu ⟨*glue*⟩ řádu 0. Hodnoty stažení vyšších řádů se nemohou v hlavním vertikálním módu vyskytnout.

`\page[|fi[l|ll|lll]]stretch` reprezentují hodnoty roztažení všech registrů typu ⟨*glue*⟩ zvlášť pro každý řád roztažení 0 až 3.

Hodnoty těchto registrů se počítají z materiálu na aktuální straně a aktualizují se při vložení boxu, linky, kernu nebo mezery do hlavního vertikálního módu.

Pomocí těchto registrů můžeme spočítat *aktuální výšku sazby*. Opomineme-li prozatím pružné výplňky a plovoucí objekty vkládané před značku, platí

$$v_{\text{akt}} = t + d$$

Tuto hodnotu postoupíme spolu s hodnotami stažení a roztažení výstupní rutině prostřednictvím ⟨*token*⟩ registru `\backgrlist`.

Takto zjištěný aktuální bod sazby je platný jen tehdy, pokud se všechen expandovaný materiál ocitne na kompletované straně. To se však nemusí stát, protože algoritmus hledání optimálního zlomu často načte a expanduje více sazby, než se vejde na stranu. Mezi tím textem, který se nevešel, mohly být také naše značky `\backgr` nebo `\endbackgr`. Jejich expanzí se zjistily hodnoty aktuálního bodu sazby pod hranicí tiskového zrcadla a podle nich se vysází zvýraznění. Avšak přetečený (ale již expandovaný) text se vrátí do přípravné oblasti a objeví se až na straně následující [1, str. 238].

Rozumné řešení tohoto problému *nestejnostranné expanze a sazby*[2] části tiskového materiálu neexistuje. Řešení by vyžadovalo obejít algoritmus hledání optimálního zlomu strany. Mohli bychom ručně přidat penaltu menší nebo rovnou −10000 a vynutit zlom, nebo střádat materiál do pomocných boxů a použít primitiv `\vsplit`. Kromě jisté těžkopádnosti nám může vadit, že tím dosáhneme nekompatibility se standardním zalámáním textu. (Když ne pro nic jiného, tak pro nemožnost použití mechanismu insertů.)

3.3. Výpočet pružných výplňků na straně

Nyní budeme pokračovat v určení referenčního bodu sazby v místě značek. Z předchozí podkapitoly víme, jak spočítat jeho polohu, neuplatní-li se pružné

[2] Toto pojmenování se mně zdá výstižnější než používaný termín asynchronní překlad.

výplňky hlavního vertikálního módu. Nespokojme se s tímto omezením a snažme se spočítat korekci polohy podle vložených výplňků. Tento výpočet bude probíhat již ve výstupní rutině.

Pokud je celkový materiál do místa zlomu vyšší než cílová výška strany (`\ht255>\pagegoal`), pak se uplatní stažení. Počítaný referenční bod se posune vzhůru úměrně poměru stažení v tomto místě a stažení celé kompletované strany. Podobně, je-li celkový materiál nižší než cílová výška strany, pak se uplatní roztažení nejvyššího přítomného řádu.

Zatímco hodnoty stažení a roztažení v místě značek jsou známy (značka je přečte z registrů `\page*` a uloží do `\backgrlist`), součet všech stažení a roztažení až do místa optimálního zlomu, stejně jako cílová výška strany, znám není. Registry `\page*` totiž ve výstupní rutině uchovávají hodnoty veškerého expandovaného materiálu, tedy i toho, který je až *za* optimálním zlomem (vliv nestejnostranné expanze a sazby). Registry `\pagetotal` a `\pagedepth` lze zjistit změřením boxu 255, ale pro pružné výplňky nic takového TEX přímo nenabízí [1, pozn. k `\pagetotal`, str. 415].[3]

3.4. Plovoucí objekty a bod sazby

Standardní plovoucí objekty v plainu i LATEXu se mohou vyskytnou na začátku strany, na pozici vložení, na konci strany a mohou být i odsunuty dále a na aktuální straně se vůbec neobjevit. Jejich konečné umístění má vliv na bod sazby značek zvýraznění v případě, že se vloží před značku. Přestože používají primitiv TEXu `\insert`, jsou algoritmy vkládání boxů s plovoucím materiálem závislé na formátu. Z důvodů značné složitosti kompletního ošetření vlivu plovoucích objektů na bod sazby, makra `backgr` téměř všechny inserty ignorují.[4] Jen inserty patřící nahoru (v LATEXu ty, které mají nastaven příznak top, v plainu třída `\topinsert`) způsobí připočtení výšky materiálu těchto insertů k dosud spočítanému bodu sazby. Započítají se i případné pružné výplňky uvnitř insertů.

Sečteme-li hodnotu referenčního bodu sazby, korekce vlivem pružných výplňků, vlivu „top“ insertů a uživatelem zadaného zpřesnění, dostaneme (z obou značek bloku) čtvrtý a pátý parametr makra provádějící vlastní zvýraznění (kap. 2.4). Je již na uživateli, jak spolu s dalšími čtyřmi údaji tuto informaci využije v definici stylu zvýraznění.

[3] Makra `backgr` se snaží tento problém řešit rozkladem boxu 255 na jednotlivé elementy, posčítat výplňky a výsledek separovat na stažení nebo roztažení daného řádu pomocí výpisu výsledku primitivem `\the`. Tento algoritmus však často selže, protože TEX neposkytuje primitiva `\unmark` a `\unwrite` potřebná k rozložení libovolného boxu. V takovém případě jsou použity (obvykle větší) hodnoty z `\page*`. Jiným, snad úplným, ale krkolomnějším, řešením by bylo dvouprůchodové načtení `log` souboru s nastavením `\tracingpages` na kladnou hodnotu a extrakcí potřebných hodnot z něj.

[4] Takovéto chování je správné pro nemalou množinu insertů: umístěné dole na straně (i `\footnote`), na samostatné straně, ty, co se zatím nevešly a plainový `\midinsert`, který není insertem.

4. Co jsem v TEXu postrádal

Zde shrnuji omezení TEXu, která činila řešení popisované úlohy obtížné nebo nedostatečně přesné.

- Možnost znovuzpracování expandovaného ale dosud nepoužitého materiálu z předchozí strany k opětovné expanzi. Toto zlepšení by vyřešilo mnoho jiných problémů, např. sazbu do různě širokého tiskového zrcadla na dvou po sobě jdoucích stranách.
- Možnost získat ve výstupní rutině informace o realizovaném stránkovém zlomu, tj. hodnoty registrů `\page*` (3.2) pro optimální cenu zlomu. Nebo obecněji, získat informace o hodnotách stažení a roztažení boxu podobně, jak lze primitivy `\wd`, `\ht` a `\dp` zjistit jeho rozměry.
- Zjistit polohu vložené značky v rámci boxu.
- Nepřítomnost primitiv umožňující odebrat z boxu mark a write, tedy něco jako `\unmark` a `\unwrite`.

Odkazy

[1] Petr Olšák. *TEXbook naruby.* Konvoj Brno, 1. vydání, 1997.

[2] Michail I. Schlesinger a Václav Hlaváč. *Matematická teorie rozpoznávání.* Vydavatelství ČVUT, Praha, 1. vydání, během 1999.

Vít Zýka, `zyka@cmp.felk.cvut.cz`

Recenze systému pdfTEX

Jiří Mžourek

Úvod

PdfTEX spojuje, jak již sám název napovídá, na jedné straně typografický systém TEX, na straně druhé hypertextový formát PDF firmy Adobe.

Cílem vývoje bylo vytvořit systém umožňující jak přímý hypertextový výstup do formátu PDF, tak i výstup do DVI. Toho bylo dosaženo a konečný výsledek vypadá v obou formátech identicky.

PdfTEX je zpětně kompatibilní s TEXem. Pro hypertextové rozšíření se používají nové příkazy. Současně je k dispozici i pdfLATEX.

Program vzniká v České republice. Jeho autorem je Hàn Thế Thành, postgraduální student Fakulty informatiky v Brně. Šíření a užívání tohoto programu se řídí GNU GPL licencí. To mimo jiné znamená, že používání programu je zdarma. V současné době jsou k dispozici zdrojové kódy i předkompilované verze pro většinu platforem (Linux, SGI IRIX, Sun SPARC Solaris, Win32 a MS-DOS).

Instalace

PdfTEX je součástí distribuce českého TEXu. Tuto distribuci, obsahující teTEX verze 0.9, lze získat např. na stránce `http://www.cstug.cz/`, nebo s českou instalací Linuxu distribuce Red Hat 6.0 ve formě balíčků RPM. Tímto způsobem jsem se k pdfTEXu dostal i já.

Při nové instalaci Linuxu je, mimo jiné, nabídnuta volba instalace systému TEX. V tomto případě se nainstalují pouze základní balíčky bez pdfTEXu. Ten je nutno poté doinstalovat ručně z balíčku `tetex-pdftex-0.9cs-10.rpm` (příkazem `rpm -i tetex-pdftex-0.9cs-10.rpm` uživatelem `root`). Zkušenější uživatel může ale přímo při instalaci Linuxu specifikovat jednotlivé balíčky a tím si ušetří pozdější práci i místo na disku.

V `Czech-HOWTO` je doporučeno prostudovat soubor `README.cstetex`. Ten jsem bohužel na CD nenalezl. Po jeho získání z `ftp://math.feld.cvut.cz/` jsem zjistil, že obsahuje informace víceméně stejné jako sekce v `Czech-HOWTO` věnovaná TEXu. Zmíněné chyby v instalaci byly v mém případě již odstraněny.

Takže jediné, co zbývá udělat po instalaci, je spustit program `texconfig` a provést alespoň doporučenou konfiguraci programu `dvips`. Pokud ale chcete používat pdfTEX, můžete bez obav i tuto akci vynechat.

Vlastní tvorba dokumentu

Pokud chceme využít rozšíření pdfTEXu, potřebujeme získat jejich dokumentaci. V adresáři `/usr/share/texmf/doc/pdftex/base/` nalezneme několik užitečných souborů. Manuál (`pdftexman.pdf`), často kladené otázky (`pdfTeX-FAQ.pdf`) a ukázkový soubor (`example.tex` s obrázkem `image.png`). K dispozici je ještě on-line manuál na `http://www.pragma-ade.nl/`. Vše je samozřejmě v angličtině. Pro první seznámení doporučuji autorský manuál, je přehledný a obsahuje veškeré potřebné informace pro začátky s pdfTEXem.

pdfTEX

Pokud se rozhodnete používat pdfTEX, obraťte svoji pozornost k manuálu a souboru `example.tex`. V manuálu naleznete kompletní popis všech rozšíření, které

pdfTEX má. Také je zde popis „vnitřku“ pdfTEXu, použití fontů a mnoho dalších užitečných informací.

Každý nový příkaz pdfTEXu začíná prefixem `\pdf`.

Výběr několika užitečných příkazů:

- `\pdfcompresslevel` určuje stupeň komprese.
- `\pdfimage` vloží obrázek, lze určit jeho konečnou velikost. Podporované formáty: PNG, TIFF, PDF a JPEG.
- `\pdfinfo` definuje parametry dokumentu (autor, datum, ...).
- `\pdfdest` cíl skoku.
- `\pdfannotlink` začátek hypertextového odkazu.
- `\pdfendlink` text mezi příkazem `\pdfannotlink` a tímto příkazem je zpracován jako součást odkazu.
- `\pdfpagewidth` nastavuje šířku stránky.
- `\pdfpageheight` nastavuje výšku stránky.

Užitečný je soubor `example.tex`. Ten obsahuje příklady použití většiny příkazů pdfTEXu. Jeho úspěšným přeložením příkazem `pdftex example.tex` si rovněž ověříte, zda máte pdfTEX dobře nainstalovaný. Kompletní popis všech příkazů a nastavení naleznete v manuálu.

pdfLATEX

Podle doporučení z manuálu je vhodné pro využití hypertextových rozšíření použít balíček (package) `hyperref`, který je součástí instalace teTEX.

Jeho použití je jednoduché, stačí pouze přidat jeho jméno do seznamu použitých balíčků. Pokud jich používate více, je dobré ho uvést až na posledním místě, protože předefinovává některé příkazy LATEXu. Příklad hlavičky dokumentu: `\usepackage{czech,a4,hyperref}`.

Tím dostanete k dispozici tato makra:

- `\href` vytvoří odkaz na URL obdobně jako v HTML.
- `\hyperbaseurl` definuje „pevnou“ část URL použitých v dokumentu. Zjednodušuje případné pozdější úpravy.
- `\hyperimage` vytvoří odkaz na obrázek.
- `\hyperdef` definuje cíl skoku.
- `\hyperref` skok na danou URL.
- `\hyperlink` lokální skok v rámci dokumentu.
- `\hypertarget` cíl skoku `\hyperlink`.

Příkazem `\hypersetup` se nastavují některé parametry. Můžete nastavit velikosti papíru, použité drivery (pdftex, dvips ...) různé barvy odkazů, automatické generování záložek (bookmarks) v PDF dokumentu, informace o dokumentu a mnoho dalších.

Příklad:

```
\hypersetup{
  pdftitle={Recenze systému pdfTeX},
  pdfauthor={Jiří Mžourek},
  pdfsubject={Semestrální práce z předmětu Publikační systém TeX}
}
```

Další možností je rozšířit dokument PDF nebo HTML o objekty používané ve formulářích (tlačítka, radio-buttony ...). U těchto objektů lze měnit jejich parametry (barvu, velikost ...). Zpracovat generované události lze pomocí kódu v JavaScriptu.

Závěr

Několik slov závěrem. Mnou používaná verze pdfTEXu (pdfLATEXu) distribuovaná s OS Linux Red Hat 6.0 mě velmi příjemně překvapila. Ačkoliv je stále ve vývojové verzi, nenarazil jsem při běžné práci na žádné problémy způsobené vlastním programem. Dodávaná dokumentace je naprosto dostačující. Vygenerovaný výstup jsem prohlížel programem Acrobat Readerem 4.0 pod OS Linux, Windows NT 4.0 a Windows 98 naprosto bez problémů. Takže pokud chcete produkovat dokumenty ve formátu PDF, nezbývá mi, než Vám pdfTEX doporučit.

Použitá literatura

- Balvínová A., Bílý M.: *Textové informační systémy — Sázecí systém LATEX — cvičení*, ČVUT 1994
- Rahtz S.: *Hypertext marks in LATEX: the hyperref package*, dokumentace k balíčku hyperref, verze z června 1998
- Thành H. T., Rahtz S., Hagen H.: *The pdfTEX users manual*, dokumentace k pdfTEXu, verze z března 1999
- `Czech-HOWTO` distribuované s instalací Linux Red Hat 6.0
- `http://www.cstug.cz`, stránka Československého sdružení uživatelů TeXu

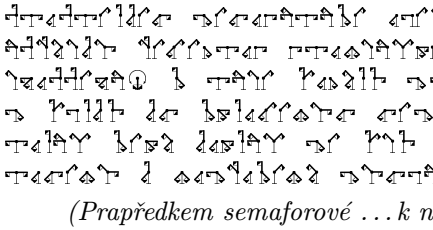
Semaforová abeceda

Vít Zýka

Z historie

Chtěl-li kapitán sdělit okolo proplouvající loďi užitečné navigační informace, zprávy o možném nebezpečí nebo novinkách ze zemí, které navštívil, vyslal na dobře viditelné místo speciálně vycvičeného signalizátora. Ten si vzal po červenožlutém praporu do každé ruky a pomocí dohodnutých poloh odvysílal zprávu. Co jedna kombinace sedmi poloh pro každou ruku, to jedno písmeno zprávy. Slova a věty pro lepší čitelnost odděloval opsáním jednoho, respektive dvou kruhů. Vysílal tak obecněji než vlajkovou abecedou a rychleji než morseovou abecedou. Mezi nutnou kvalifikaci příjemce jistě patřily (kromě schopnosti číst zcela přirozeně znaky – a to nebylo při tehdejší námořní gramotnosti samo sebou) i ostříží oči. Vždyť rozeznání poloh na vzdálenost až 7 kilometrů bylo na kymácejících se lodích stejně obtížné jako dnes.

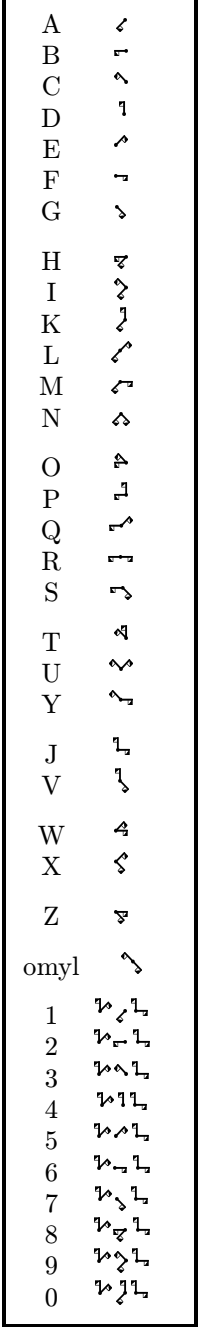
Semafor byl na vizuální přenos zcela odkázán. S rozvojem elektronického přenosu dat byl rychle vytlačen morseovkou. No zkuste semafor procpat telegrafem! Ne, že by to nešlo, ale efektivní výsledek přenosu dostaneme až nyní, v éře počítačů a internetu. Zatímco kapitánovi na lodi pomohl semafor v předání zpráv, v dnešní době tomu bude pravděpodobně naopak. Posuďte sami:

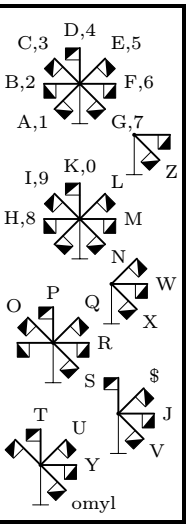


(Prapředkem semaforové . . . k nastavení symbolu.)

Dnešní semafor navrhl jakýsi plukovník anglické armády roku 1822. V Anglii se používal velmi krátce. Více se uplatnil v námořnictvu USA během občanské války.

Semafor





Varianta		Roman `r`	Bold `bf`	Typewriter `tt`	Slanted `sl`
Pillar	`smf`				
Empty	`smfe`				
Person	`smfp`				

Uspořádání abecedy

Sedm poloh dává dohromady 28 kombinací znaků. Základní abeceda obsahující 26 písmen je rozdělena do šesti tzv. kruhů. Písmena jsou v nich rozdělena téměř logicky podle abecedního pořadí. Kruh tvoří skupina písmen, která mají stejnou polohu praporku drženého v pravé ruce.

Na číslice již nezbylo místo, takže se vysílají jako písmena začátku abecedy, jen se před první cifrou vyšle speciální znak začátku číslic a po poslední písmeno . Druhým speciálním znakem, na který ještě zbývá místo, je omyl . Jeho význam je podobný jako klávesy backspace na klávesnici. Ostatní znaky, jako třeba interpunkce, se v semaforu neužívá nebo se musí vypisovat slovy.

Použití abecedy

Pro použití této abecedy v TEXu jsem vytvořil font s názvem `semaf`. Stačí pouze přepnout font a ošetřit v semaforu nedefinované znaky. K dispozici jsou čtyři řezy ve třech variantách, viz. tabulka. Jinou variantu, lišící se nepohyblivou částí kresby, si uživatel znalý METAFONTu může snadno doplnit. K zavedení semaforového fontu je možné použít dodávané soubory `semaf.tex` pro plain nebo `il2semaf.fd` pro LATEX 2_ε. Font je český v kódování IL2 jako $\mathcal{CS}$fonty, akcenty jsou jednoduše vypouštěny.

Koncept zobecnělých ligatur umožnil velmi elegantní řešení sazby čísel. Počáteční i koncový znak čísel je tak vkládán automaticky, dokonce bez nutnosti psaní nějakých maker. Fonty jsou k dispozici na adrese `http://cmp.felk.cvut.cz/~zyka/zykatex.html`.

Hlavním kritériem dobrého fontu je asi jeho čitelnost. Ta je, pravda, u semaforu velmi nízká. Přesto existuje minimálně jedna oblast využití. Rozpoznali ji tvůrci skautského hnutí, když pomocí romantického námořního pozadí semaforem rozvíjeli paměť, postřeh a koordinaci hochů a dívek.

Vít Zýka, `zyka@cmp.felk.cvut.cz`

TrueType-fonty v TEXu

Vladimír Koutný

Len nedávno sa mi dostal do rúk Zpravodaj ČSTUG 4/96, kde sa v článku **Problémy s fonty v TEXu** spomínajú rozličné problémy, ktoré vznikajú, ak sa snažíme používať v TEXu iné typy písma, ako sú štandardné METAFONT-ové, konkrétne **Adobe Type1** a **TrueType**. Konkrétne pre použitie TrueType fontov autor spomínal, že je to možné len konverziou do Adobe Type1, a aj to pomocou komerčných programov. Keďže som asi pred dvoma rokmi potreboval využiť niektoré TrueType fonty priamo v TEXu, rozhodol som sa vtedy vytvoriť konverzný program, ktorý z TrueType fontu generuje priamo METAFONT, čo veľmi uľahčuje použitie týchto fontov.

Základom tohto konvertora je knižnica FreeType od Davida Turnera (The FREE TrueType Font Engine), ktorá sa dá priamo použiť na prezeranie TrueType fontov na obrazovke. Keďže môj konvertor je len doplnkom k tejto knižnici, dá sa použiť aj na prezeranie jednotlivých znakov. V prezeracom režime sa program spúšťa `zoom.exe truetype.ttf`, ovládanie je pomocou kláves `I, O, K, L, +, -, ESC`. Číslo v pravom hornom rohu je číslo znaku. Keďže TrueType fonty vo väčšine prípadov používajú kódovanie UNICODE, môže byť rôznych znakov viac ako 256.

Pôvodné rutiny nepodporovali znaky zložené z iných, vzájomne poposúvaných znakov, čo je najbežnejšia metóda, ako vytvoriť znaky s diakritikou, a preto väčšina fontov takéto znaky obsahuje (aj keď možno nemajú správnu diakritiku slovenských a českých znakov).

Samotná konverzia sa spúšťa príkazom `zoom.exe truetype.ttf -export`. Pri konverzii sa používa súbor `export.ttc`, čo je textový súbor v aktuálnom adresári, ktorý definuje, ktoré znaky sa budú konvertovať a ako sa budú mapovať na 256 znakov použiteľných v METAFONTe. Počas behu programu sa vytvára pomocný súbor, ktorý môže byť veľmi veľký (po konverzii sa automaticky zmaže). Výsledkom je súbor METAFONTu, ktorý je možné (skoro vždy) priamo použiť v TEXu. Za rutinky zapisujúce do METAFONTu vďačím spolužiakovi Riškovi Kráľovičovi.

Samozrejme, že existuje viacero nedostatkov. Asi najpodstatnejším je to, že nie vždy sa skonvertovaný font dá plne automaticky generovať pri nižších rozlíšeniach. Dôvodom je fakt, že TrueType fonty obsahujú informácie, podľa ktorých sa dajú tieto problémy obísť (menenie obrysov podľa veľkosti písmena), ale táto funkcia zatiaľ nie je implementovaná. Pre pochopenie tohto javu treba vedieť, ako sú definované znaky v TrueType fontoch – sú to len čisté obrysy zložené z Bezierových kriviek alebo z úsečiek. Pri konverzii do METAFONTu sa vypĺňa plocha

uzavretá takto definovanou krivkou, a tu je problém – pri malých písmenkách sa môže stať, že sa krivka prekríži, čo sa METAFONTu nepáči. Zo skúsenosti však tento jav nastáva hlavne pri rôznych okrasných a zložitých znakoch. Možnosti sú dve – buď písmenko zväčšiť, alebo počkať na ďalšiu verziu...

Ďalšie obmedzenie: TrueType font je zložený z viacerých tabuliek, pričom obrysy znakov sú uložené v jednej z nich. Veľkosť tejto jednej tabuľky (teda ani žiadnej inej, ale táto je najväčšia) nesmie presiahnuť viac ako 64kB – v opačnom prípade to najpravdepodobnejšie spôsobí zamrznutie systému. Pravidlo: ak je TTF súbor menší ako 70kB, nemal by s tým byť žiadny problém.

Znaky, ktoré sa budú konvertovať, sú určené v súbore `export.ttc`. Je to texotvý súbor, kde na každom riadku je definícia jedného znaku. Na začiatku riadku je určenie, kam sa daný znak namapuje, čiže buď ASCII hodnota znaku (dekadicky) v použitom kódovaní (v mojom prípade Kamenický), alebo `'x`, kde x je priamo znak. Za medzerou je UNICODE číslo daného znaku, buď dekadicky (123) alebo hexadecimálne ($123). Podľa UNICODE čísla znaku sa určí správne číslo znaku vo fonte (číslovanie znakov vo fonte je úplne chaotické, ale je zapísané v jednej tabuľke). V prípade, že sa daný znak vo fonte nenachádza, sa použije prvý znak vo fonte (zvyčajne to býva prázdny obdĺžniček).

TrueType fonty nepodporujú ligatúry, no aj napriek tomu veľká časť fontov obsahuje tieto znaky. Len neexistuje (aspoň som to nenašiel (základná špecifikácia TTF má asi 400 strán)) žiadne prepojenie medzi skupinou znakov a správnou ligatúrou.

Toto je prvá funkčná verzia konvertora. K dispozícii dávam len skompilovaný tvar, nakoľko som zdroják neuznal hodný zverejnenia (poznámky v slovenčine, angličtine a francúzštine, ...). Chystám sa však vytvoriť novú verziu, ktorá s tou terajšou bude mať spoločné asi len to, že bude slúžiť na ten istý účel. Stručný zoznam plánovaných vylepšení:

- úplne nové rutinky na čítanie TrueType fontov
- podpora veľkých TrueType súborov
- interaktívny výber znakov pre konverziu
- možnosť doplnenia ligatúr
- správne generovanie aj pre malé veľkosti znakov
- možnosť pridania nových znakov s diakritikou

Najnovšia verzia a iné informácie sa nachádzajú pravdepodobne na adrese `http://www.linuxee.sk/ttf`, ak nie, tak sa stačí ozvať mailom na moju adresu `kluka@hotmail.com`. Takisto privítam akékoľvek pripomienky alebo rady ohľadom nastoleného problému.

Vladimír Koutný
`kluka@hotmail.com`

LaTeXová kuchařka / 5

Zdeněk Wagner

V této části seriálu bude více než jinde platit to, co bylo uvedeno v prvním dílu. Půjde zde zejména o demonstraci rozličných maker, s jejichž pomocí lze ovlivňovat vzhled dokumentu. Přitom některé ukázky mohou z hlediska typografie vypadat ošklivě. Než tedy taková makra ve svých dokumentech použijete, zvažte důkladně, zda tím svůj text neznehodnotíte.

26. Kde LaTeX vezme rozměry stránky

Rozměry stránky očekává TeX v registrech `\hsize` a `\vsize`. LaTeXisté však nastavují `\textwidth` a `\textheight`. LaTeX je ale jen velkým balíkem maker a zpracovává se stejným programem. Mechanismus, jak funguje nastavování rozměrů stránky, je před uživateli ukryt. My se o něm stručně zmíníme, aby další text byl srozumitelný.

Plain TeX neobsahuje žádné makro pro vícesloupcovou sazbu. Chceme-li sázet text do více sloupců, musíme se o to postarat sami. Text se musí vysázet do dlouhého úzkého sloupce. Dále musíme TeXu vnutit jinou `\output` rutinu, která tento polotovar rozdělí na sloupce a usadí je vedle sebe. LaTeX obsahuje podporu pro dvousloupcovou sazbu. Lze buď použít volbu `twocolumn` nebo přepínat pomocí příkazů `\twocolumn` a `\onecolumn`. V obou případech se vyjde z celkové šířky stránky zadané v `\textwidth`. Při jednosloupcové sazbě je tím definováno vše, při dvousloupcové sazbě musíme vypočítat šířku sloupce. Odečteme tedy mezeru mezi sloupci zadanou v proměnné `\columnwidth` a výsledek vydělíme dvěma. Šířku sloupce pak vložíme do `\hsize`, `\linewidth` a `\colwidth`. Tuto činnost provede nejprve `\begin{document}`, potom přepínací makra `\twocolumn` a `\onecolumn`. Podobně se zachází s registrem `\vsize`. Vidíme tedy, že LaTeX nakonec vloží rozměry sloupce (před rozdělením) do registrů `\hsize` a `\vsize`. Právě hodnota registru `\hsize` bude pro nás v dalším textu důležitá.

27. Rozměry odstavce

Základní rozměr odstavce je uložen v registru `\hsize`. Z něho je odvozena šířka řádku. Na levé straně řádku je vynecháno místo velikosti `\leftskip`, na pravé straně je volné místo velikosti `\rightskip`. Tyto registry jsou inicializovány

na nulovou hodnotu, takže text zabírá přesně šířku `\hsize`. Drobný rozdíl je v prvním a posledním řádku odstavce. Na začátku prvního řádku je navíc mezera velikosti `\parindent`, pokud ovšem není potlačena příkazem `\noindent`. Na konci posledního řádku je mezera velikosti `\parfillskip`. Ta má obvykle nekonečnou roztažitelnost.

> V tomto odstavci si demonstrujeme použití `\leftskip` a `\rightskip`. Levý okraj byl nastaven na 2 cm, pravý na 1 cm. Můžete si to snadno ověřit pravítkem. Všimněte si, že první řáden má navíc odstavcovou zarážku stejné velikosti jako jiné odstavce. Je to tím, že hodnota registru `\parindent` nebyla změněna. Zůstala zachována i hodnota `\parfillskip`, takže nedojde k problémům při zlomu posledního řádku. Registrům lze přiřadit i záporné hodnoty. Text pak bude přesahovat mimo sazební obrazec. Dále si musíme uvědomit, že algoritmus řádkového zlomu použije ty hodnoty, které jsou platné na konci odstavce. Pokud měníme hodnoty `\leftskip`, `\rightskip` nebo jiných registrů jen lokálně, musíme ukončit odstavec ještě před koncem skupiny.

Vynulováním registrů `\parindent` a `\parfillskip` lze sázet do bloku, jehož východová řádka bude zcela zaplněna. Vypadá to efektně, ale většinou tím způsobíme přetečené a podtečené boxy. Jedinou nápravou je v takovém případě zásah do textu. I zde byla poslední věta upravována tak dlouho, až to vyšlo.

Někdy potřebujeme vykousnout do textu pravoúhlý otvor. Pro tyto účely můžeme využít registrů `\hangafter` a `\hangindent`. Hodnota vložená do registru `\hangindent` určuje velikost odsazení. Je-li hodnota kladná, odsadí se text zleva.

> Záporná hodnota způsobí odsazení vpravo. Registr `\hangafter` určuje, po kolika řádcích odstavce se má začít odsazovat. Pokud je hodnota záporná, pak je tím naopak označeno, kolika počátečních řádků se odsazení týká. Uvažuje se samozřejmě absolutní hodnota. V tomto odstavci jsme do `\hangafter` vložili hodnotu 3 a odsazení bylo nastaveno na 2 cm.

Zde jsme nastavením `\hangafter -4 \hangindent -1cm` přestěhovali otvor do protilehlého rohu odstavce. Vhodnou změnou znamének lze otvor vynechat i ve dvou zbývajících rozích. Všimněte si, že v obou případech nám navíc zůstala odstavcová zarážka v prvním řádku. Její velikost je rovna `\parindent`. V souvislosti s tímto příkladem ale musíme zdůraznit velmi důležitou vlastnost stylů CZECH a SLOVAK. Použijí-li se s parametrem `split`, je sice automaticky při dělení slova opakován spojovník na začátku následujícího řádku, ale uživatel je za to penalizován změnou kategorie znaku „-“ na 13. Následkem toho nelze v dokumentu zadávat záporná čísla. Lze to řešit tím, že v preambuli dokumentu, dokud ještě kategorie změněny nejsou, použijeme definici `\def\minus{-}`. Výše uvedenou hodnotu registru `\hangafter` pak

budeme zadávat jako `\minus 4`. Podobně naložíme i s rozměrovým registrem `\hangindent`.

Musíme se ještě zmínit o příčině nejčastější chyby, která způsobí, že primitivy `\hangafter` a `\hangindent` nefungují. TEX totiž na začátku odstavce tyto hodnoty vynuluje. Pokud je tedy zadáme ještě před prvním znakem odstavce, jsme stále ve vertikálním režimu a v okamžiku zpracování prvního znaku o ně přijdeme. Pokud je z estetických důvodů nechceme nastavovat uvnitř odstavce nebo na jeho konci, musíme před těmito příkazy použít `\leavevmode` a nesmíme za nimi nechat prázdný řádek.

Podívejme se nyní, jakého efektu dosáhneme prostředím CENTERLAST, jehož definice je uvedena níže:

```
\newenvironment{centerlast}{%
  \parindent 0mm
  \leftskip 0mm plus 1fil
  \rightskip -\leftskip
  \parfillskip 0mm plus 2fil
  \parskip\baselineskip
  \def\\{\unskip\hskip\parfillskip\break\ignorespaces}}%
  {\par\vspace{\parskip}}
```

Toto nastavení způsobí, že poslední řádek odstavce bude vycentrován. Jak je toho dosaženo? Nejprve je odstraněna odstavcová zarážka. Dále vložíme nekonečnou roztažitelnost do registru `\leftskip`. Roztažitelnost v registru `\rightskip` je také nekonečná, ale se záporným znaménkem. Při sazbě řádků se tedy vyruší. Všechny řádky proto budou zabírat plnou šířku `\hsize`. Poslední řádek má kromě `\rightskip` ještě `\parfillskip`. Zde TEX sečte hodnoty roztažitelnosti a výsledkem bude `1fil`. Ta je stejná jako v `\leftskip`. Tím je zařízeno centrování posledního řádku.

Odstavce by nepůsobily příliš estetickým dojmem, kdyby byly sesazeny těsně za sebe. Proto do registru `\parskip` vložíme hodnotu `\baselineskip`. Mezi odstavci tak bude vynechán právě jeden volný řádek.

Někdy nám poslední řádek vyjde extrémně krátký. Ani to nepůsobí vzhledně. Lze si pomoci tím, že zkrátíme předposlední řádek a necháme jej vycentrovat. Pro přechod na nový řádek používáme v LATEXu makro `\\`. Aby fungovalo i zde, musí mít jinou definici. Pro jistotu nejprve odebereme předchozí mezeru. Pak musíme vložit stejnou mezeru, jakou používáme na konci odstavce, a přidat `\break`, což vynutí řádkový zlom. Pak budeme ignorovat všechny následující mezery. Příklad použití je zde.

Aby nemusel uživatel pamatovat na to, že musí ponechat prázdný řádek před `\end{centerlast}`, vložíme do druhé části definice prostředí explicitní `\par`. Na konci prostředí bude také nastavena původní hodnota registru `\parskip`. Vertikální mezeru tedy také vložíme zde. V tomto příkladu mlčky předpokládáme, že původní hodnota `\parskip` byla nulová. Pokud mohla být nenulová a nechceme mít na konci tohoto prostředí vertikální mezeru jiné velikosti, budeme potřebovat poněkud chytřejší makro. Tuto úlohu však ponecháme na čtenářích.

Ještě složitější vzhled odstavce dosáhneme použitím primitivu `\parshape`. Tento primitiv má lichý počet parametrů. Prvním parametrem je počet řádků, které jsou dále specifikovány. Pro každý řádek je uvedena dvojice rozměrů: odsazení a šířka řádku. Má-li odstavec méně řádků než kolik dvojic rozměrů je uvedeno, jsou nadbytečné informace ignorovány. V opačném případě se poslední dvojice rozměrů odsazení/šířka použije pro všechny následující řádky. Podobně jako `\hangafter` je i `\parshape` vynulován na začátku odstavce. Musíme si tudíž dát pozor, abychom jej nenastavovali ve vertikálním režimu.

Příklad použití zde neuvedeme. Najdete jej například ve Zpravodaji č. 1 z roku 1996. V něm byl tímto způsobem vsazen do textu obrázek čarodějnice na straně 38. Ještě důmyslnější využití popsal Petr Olšák v knize TEXbook naruby na straně 236.

Ruční odměřování parametrů pro `\parshape` není zrovna pohodlné. Proto lze na CTAN najít styly SHAPEPAR, WINDOW a řadu jiných, které umožní zadání tvaru odstavce a parametry se vypočtou vhodným makrem.

28. Usazujeme iniciály

Cílem této kapitoly je předvedení dalšího způsobu, jak lze využít parametrů `\hangafter` a `\hangindent`. Ukážeme si zde, jak lze usazovat iniciály na začátek odstavce. Z cvičných důvodů použijeme iniciálu na začátku každého odstavce. V běžném textu to není zvykem, iniciály se obvykle používají pouze na začátku kapitoly.

Na začátku odstavce musíme vynechat obdélníkový otvor. Jak jsme si již ukázali, můžeme pro tento účel využít `\hangafter` a `\hangindent`. Šířku vynechaného otvoru přizpůsobíme šířce písmene. Často se první řádek připojuje k iniciále těsněji než řádky následující. Toho snadno docílíme nastavením záporné hodnoty do registru `\parindent`. Nyní si předvedeme definici prostředí, které tuto činnost provede.

```
\newdimen\Iwidth
\newbox\InicBox
```

```
\newcommand\Iniciala
   {\usefont{IL2}{ppl}{m}{n}\fontsize{44}{44}\selectfont}
\newenvironment{inicialy}{\parindent -5dd
  \def\iniciala##1{\setbox \InicBox=\hbox
          {\kern-2dd\Iniciala ##1\kern -\parindent}%
    \Iwidth=\wd\InicBox
    \leavevmode \hangafter -3 \hangindent\Iwidth
    \vadjust{{\everypar{}\noindent
      \smash{\lower 20dd \hbox to \z@ {\box\InicBox\hss}}}}}%
    \everypar{\iniciala}}{\par}
```

Nejprve jsme si definovali pomocný rozměr `\Iwidth`, box `\InicBox` a font pro tisk iniciály. Uvnitř prostředí pak nastavíme záporný `\parindent`. Dále si nadefinujeme makro `\iniciala`, které vyžaduje jeden parametr. Tímto parametrem je požadované písmeno, které máme vytisknout jako iniciálu. Definici zapisujeme uvnitř definice prostředí, proto musíme znak `#` zdvojit.

Makro `\iniciala` začíná svoji činnost tím, že vysadí iniciálu do pomocného boxu. Opticky vypadá iniciála lépe, když poněkud vyčnívá doleva. Proto je na začátku boxu záporný `\kern`. Za iniciálou je další `\kern`. Jeho velikost je záporně vzatá hodnota `\parindent`. Bez této mezery by druhý a třetí řádek byly těsně nalepeny na iniciálu a první řádek by do ní zasahoval. V následující instrukci změříme šířku takto vytvořeného boxu a vložíme ji do `\Iwidth`.

Touto definicí jsme ukončili přípravné akce a můžeme zahájit sestavení odstavce. Nejprve je nutno zajistit přechod do horizontálního režimu. Použijeme k tomu `\leavevmode`. Dále nastavíme příslušné hodnoty registrů `\hangindent` a `\hangafter`. O vložení záporné mezery specifikované v registru `\parindent` se postará algoritmus řádkového zlomu. Iniciálu pak vložíme na správné místo pomocí `\vadjust`. Musíme přejít do horizontálního režimu, abychom směli použít `\lower`. Tím totiž usazujeme box s iniciálou o dva řádky níže. Makro `\smash` vynuluje hloubku a výšku boxu. Bez této operace by TEX vynechal patřičnou vertikální mezeru, což my ale nechceme. K registru `\everypar` se vrátíme za malý okamžik.

Závěrečným krokem je zajištění, aby se naše makro volalo automaticky na začátku každého odstavce. Využijeme k tomu další mechanismus, který nám TEX poskytuje. Na začátek odstavce se totiž automaticky vloží obsah registru `\everypar`. Do tohoto registru tedy vložíme token `\iniciala`. Jakmile TEX přejde do horizontálního režimu, vloží do vstupní fronty obsah registru `\everypar` následovaný textem odstavce. Na začátku fronty je tudíž token `\iniciala`. Ten odebere jako parametr první token odstavce. Bohužel to nemusí být zrovna znak – pokud odstavec nezačne písmenem nebo číslicí, dojde ke katastrofě.

Vraťme se nyní k usazování iniciály s použitím `\vadjust`. Řekli jsme si, že musíme přejít do horizontálního seznamu. V tom okamžiku ovšem TEX vloží do fronty obsah registru `\everypar`, který skrývá makro pro sazbu iniciály. Pokud bychom na začátku `\vadjust` registr `\everypar` nevyprázdnili, snažil by se TEX vysázet iniciálu v iniciále v iniciále v iniciále... Tento nekonečný cyklus by skončil zahlcením paměti. Navíc je prvním tokenem tohoto odstavce `\noindent`, což by způsobilo další chyby. Vyprázdnění `\everypar` provádíme lokálně uvnitř skupiny, aby TEX opět použil makro pro sazbu iniciály v dalším odstavci.

Obvykle nebudeme sázet iniciálu na začátku každého odstavce. Problém s `\everypar` nám tím odpadne. Častěji budeme vyžadovat iniciálu na začátku kapitoly. Nebudeme tedy definovat prostředí `inicialy`, ale makro `\iniciala` bude posledním příkazem v makru `\chapter`. Zde nás čeká jiná obtíž: pokud pod příkazem `\chapter` vynecháme volný řádek, bude parametrem makra `\iniciala` token `\par`. Vyřešíme to tím, že posledním příkazem makra `\chapter` bude `\inic` s následující definicí:

```
\def\inic#1{\ifcat #1A\def\next{\iniciala #1}\else
            \let\next\inic\fi
            \next}
```

Tato definice zajistí, že všechny tokeny kromě písmen s kategorií 11 budou mlčky ignorovány. Nejsou tím ovšem odstraněny všechny potenciální problémy. Mohou nastat případy, kdy je nutné před začátkem prvního odstavce kapitoly provést nějakou akci. Pak budeme muset v makru `\inic` uvést vhodný příkaz `\if`, který potřebnou činnost provede.

Skutečná makra pro sazbu iniciál budou muset být poněkud složitější. Tvar písmen se liší a často bude nutné upravit odstup odstavce od iniciály. Také písmeno V požaduje jiné zacházení. Jak se můžete přesvědčit výše, nevypadá díra vedle této iniciály nijak hezky. Navíc iniciála nemusí být jen písmeno z nějakého fontu. Můžeme použít obrázek buď naskenovaný nebo vektorový. Již dříve jsme zmínili Zpravodaj č. 1 z roku 1996. Libor Sýkora použil iniciály v článku Vkládání obrázků do LATEXu. Každou iniciálu měl ovšem v jiné velikosti, což bylo redakčním zásahem sjednoceno. Tím se vysvětlí, proč nejsou všechny parametry makra použity (jinak bych musel zasahovat přímo do textu článku). Iniciála ovšem není na začátku každého odstavce, takže se nevkládá pomocí `\everypar`. Definici upraveného makra, které bylo v článku použito, uvádíme níže bez vysvětlení.

```
\newlength{\inicl}
\newbox\inibox
\newcommand\inic[2]{%
  \setbox\inibox=\hbox{\epsfysize=30dd \relax
                       \epsffile{#1#1.eps}}%
```

```
\inicl=\wd\inibox \advance\inicl .5em
\noindent \hangafter\minus3 \hangindent\inicl
\vadjust{\smash{\vbox{\noindent
                \hbox to0pt{\lower20.5dd\box\inibox\hss}}}}%
\hspace*{\minus4pt}\ignorespaces}
```

Další příklady použití parametrů `\leftskip` a `\rightskip` (například pro sazbu veršů) budou uvedeny v příštím pokračování LaTeXové kuchařky.

29. Literatura

První citace odkazuje na příklad, který je v tomto článku popisován. V dalších knihách je uveden podrobný popis algoritmů řádkového zlomu.

1. Libor Sýkora: *Vkládání obrázků do LaTeXu*. Zpravodaj Československého sdružení uživatelů TeXu, **6** (1), 37–43 (1996).
2. D. E. Knuth: **The TeXbook**. Addison Wesley, Reading 1984. ISBN 0-201-13448-9.
3. P. Olšák: **TeXbook naruby**. Konvoj, Brno 1997. ISBN 80-85615-64-9. Elektronická verze je k dispozici na `http://math.feld.cvut.cz/olsak/tbn/`.

Zdeněk Wagner
`wagner@mbox.cesnet.cz`

TUG 2000: Call for papers

The 21st annual meeting of the TeX users Group is to be held in Oxford, UK between Sunday 13th August and Wednesday 16th August 2000.

Details of the conference are available at the conference Web site:

`http://tug2000.tug.org`

Papers are invited on any subject relevant to TeX.

Authors wishing to present a paper at the conference should submit an extended abstract (in English) via email to:

`tug2000-papers@tug.org`

The last date at which proposals for papers will be accepted is: **15th January 2000** however it would be helpful if proposals are submitted before the end of this year.

Proposals will be reviewed and a decisions on acceptance of the papers will be made by: **31st January 2000**

For papers accepted for the conference, other important dates are:

31st March 2000	Deadline for submission of paper for refereeing.
31st May 2000	Deadline for final version of paper for inclusion in Conference proceedings

Sebastian Rahtz
sebastian.rahtz@oucs.ox.ac.uk
(TUG 2000 Chair)

David Carlisle
davidc@nag.co.uk
(TUG 2000 Program Coordinator)

TUGboat 19(4), December 1998

TUGboat 20(1), March 1999

TUGboat 20(2), June 1999

Zpravodaj Československého sdružení uživatelů TEXu
ISSN 1211-6661

Vydalo:	Československé sdružení uživatelů TEXu vlastním nákladem jako interní publikaci
Obálka:	Bohumil Bednář
Počet výtisků:	710
Uzávěrka:	4. listopadu 1999
Odpovědný redaktor:	Zdeněk Wagner
Tisk a distribuce:	KONVOJ, spol. s r. o., Berkova 22, 612 00 Brno, tel. 05-740233
Adresa:	CSTUG, c/o FI MU, Botanická 68a, 602 00 Brno
fax:	05–412 125 68
e-mail:	cstug@cstug.cz

Zřízené poštovní aliasy sdružení CSTUG:

`bulletin@cstug.cz`, `zpravodaj@cstug.cz`
korespondence ohledně Zpravodaje sdružení

`board@cstug.cz`
korespondence členům výboru

`cstug@cstug.cz`, `president@cstug.cz`
korespondence předsedovi sdružení

`cstug-members@cstug.cz`
korespondence členům sdružení

`cstug-faq@cstug.cz`
řešené otázky s odpověďmi navrhované k zařazení do dokumentu CSFAQ

`secretary@cstug.cz`, `orders@cstug.cz`
korespondence administrativní síle sdružení, objednávky CD-ROM

`bookorders@cstug.cz`
objednávky tištěné TEXové literatury na dobírku

ftp server sdružení:
`ftp://ftp.cstug.cz/`

www server sdružení:
`http://www.cstug.cz/`

Podávání novinových zásilek povoleno Českou poštou, s. p. OZJM Ředitelství v Brně č. j. P/2–1183/97 ze dne 11. 3. 1997.