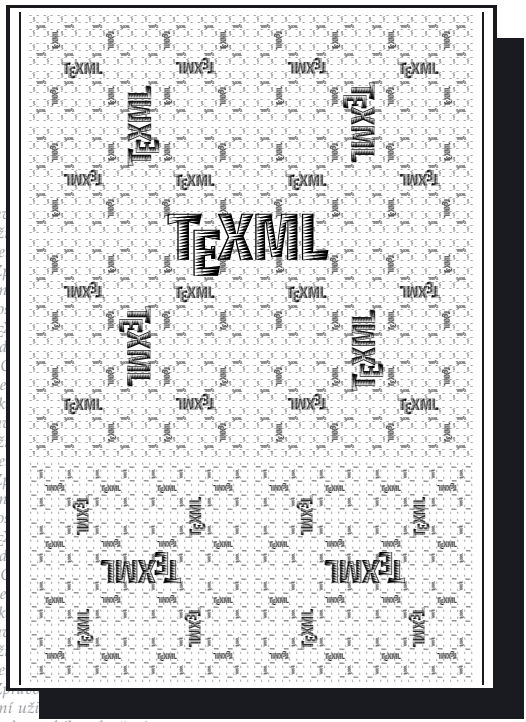
CSTUG



ZPRAVODAJ

Československého sdružení uživatelů TEXu

1
2003

ISSN 1211-6661 Tištěná verze
ISSN 1213-8185 Online verze
Ročník 13

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, nejlépe jako jeden archivní soubor (`.zip, .arj, .tar.gz`). Postupujte podle instrukcí, které najdete na stránce `http://bulletin.cstug.cz/` . 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, formáty Macintosh 1.44 MB a EXT2 jsou též přijatelné. Nezapomeňte přiložit všechny soubory, které dokument načítá (s výjimkou standardních součástí CSTEXu), zejména v případě, kdy vás nelze kontaktovat e-mailem.

ISSN 1211-6661 (tištěná verze)
ISSN 1213-8185 (online verze)

Úvodníček

Petr Olšák

Vážení čtenáři,

dostává se vám do rukou další číslo Zpravodaje našeho sdružení. Převážná část článků je zde zaměřena na XML TEXnologie. Mnozí mohou namítnout, že tyto technologie ve své podstatě nemusejí mít nic společného s TEXem. Možná si ještě někteří čtenáři vzpomínají na vášnivou diskusi Jirky Koska s ostatními TEXisty (včetně mě) na našem e-mailovém listu `cstex@cs.felk.cvut.cz` o postavení TEXu a ostatních formátorů, které se používají v souvislosti s XML.

Možná, že z obsahu tohoto čísla vyplyne, že postavení TEXu v těchto moderních technologiích zdaleka není bezvýznamné, ba právě naopak. Znovu a znovu se tak ukazuje genialita návrhu programu TEX a jeho nadčasovost.

V únoru udělal pan Bittó v rámci své soukromé iniciativy anketu „ako TEXujeme“. Účastníci elektronického listu `cstex@cs.felk.cvut.cz` mu odpovídali na otázky, jakou používají TEXovou distribuci, na jakém systému, jaký je jejich oblíbený formát (balíček maker) apod. Pan Bittó anketu vyhodnotil a výsledky dal k dispozici na web CSTUGu. Rovněž ji nabídl redakci Zpravodaje, takže nyní se mohou podívat na výsledky této zajímavé ankety všichni členové CSTUGu. Děkujeme.

Donald Knuth ohlásil koncem roku 2002 opravu TEXu, METAFONTu a plainu. V případě TEXu se posunuje číslo verze na 3,141592. Na svých stránkách Knuth zdůrazňuje, že oprava nemá význam pro běžné uživatele (tj. nemusejí být rozlítostněni, že zrovna nemají na svém počítači tu nejnovější verzi TEXu), ale jedná se spíše o pokyn hlavně směrem k distributorům TEXu, aby opravu zařadili a začali šířit novou verzi. Pokud jde o `web2c` TEX (prazáklad současně používaných TEXových distribucí), tak tam se kvůli konfliktům s eTEXem povedlo zařadit opravu až koncem února 2003. Ani vy, vážení čtenáři, nemusíte zoufat, že nemáte tuto nejnovější verzi. Pravděpodobnost, že narazíte na chybu, která byla posledním zásahem do TEXu opravena, je mizivá. Jde (zhruba řečeno) o to, že za jistých *velmi* specifických okolností se ve výplni typu `\xleaders` objeví nesprávný počet objektů. Za zmínku stojí skutečnost, že mezi autory bug-reportu je programátor Karel Skoupý z České republiky. Gratulujeme.

Na listopadové konferenci SLT jsem přislíbil, že se budu zabývat problematikou zpracování textů v TEXu v kódování UTF-8, které je odvozeno od UNICODE a má pravděpodobně velkou budoucnost. Umožňuje totiž kódovat v podstatě neomezené množství znaků abeced jazyků celého světa. Bohužel pouze ASCII

znaky jsou v tomto kódování nezměněny a akcentované znaky naší abecedy tam jsou kódovány do dvou bytů. Okamžité řešení na úrovni expand procesoru TEXu (pomocí aktivních znaků) jsem vytvořil rovnou na konferenci během jedné přednášky. Nicméně v zásobě jsem měl i řešení podstatně robustnější na úrovni input procesoru TEXu. Šlo o to rozšířit možnosti mého stávajícího encTEXu. Do toho úkolu jsem se pustil o vánočních svátcích a první chodivá verze se zrodila za zvuku ohňostrojů ze Silvestra na Nový rok 2003. V lednu jsem svou novou verzi encTEXu zveřejnil. V doplnění dalších konverzních tabulek mi pak hodně pomohl pan David Nečas. Nyní jsem tuto verzi encTEXu nabídl k zařazení do standardní distribuce `web2c` TEXu a ukazuje se, že (po počáteční diskusi, při které jsem se snažil obhájit smysl mého rozšíření) bude encTEX zařazen od nejbližší další verze `web2c` TEXu.

Chtěl bych v tomto úvodníčku upozornit na konferenci EuroTEX 2003, která proběhne v červnu v Bretani. Máte-li o tuto akci zájem, navštivte, prosím, stránky `http://omega.enstb.org/eurotex2003/`. Program je na stránkách zveřejněn a bude opravdu bohatý. CSTUG bude hradit náklady aktivním účastníkům této konference z řad členů našeho sdružení.

Dovolte mi na závěr ještě připomenout některé „organizační“ záležitosti našeho sdružení. Především jsme dostali od Ministerstva vnitra potvrzeny Stanovy přesně ve znění, které bylo schváleno na našem posledním valném shromáždění a které bylo otištěno v minulém dvojčísle Zpravodaje. Naše sídlo se tedy konečně přestěhovalo do Prahy.

Poslední valné shromáždění CSTUGu rozhodlo, že koncem června se automaticky ukončí členství těm členům, kteří nezaplatili členské příspěvky za rok 2003. Patříte-li mezi tyto dlužníky, dostanete ještě upozornění o této skutečnosti e-mailem. Využívám ovšem také prostor v tomto úvodníčku k upozornění na tuto skutečnost neadresně všem členům sdružení. Zřejmě najdete v některém z posledních čísel Zpravodaje složenku. Pokud jste již zaplatili (například převodem z bankovního účtu), pak si, prosím, složenky nevšímejte, případně ji nabídněte některému svému kolegovi, který teprve zvažuje stát se členem našeho sdružení.

Připomínám, že členské příspěvky na rok 2003 byly stanoveny valným shromážděním v této výši: řádní členové zaplatí 300 Kč, studenti a důchodci 200 Kč, členové ze Slovenska pak zaplatí stejnou částku, ale ve slovenské měně: 300 Sk nebo 200 Sk. Kolektivní členové mají členský příspěvek ve výši 1850 Kč a kolektivní členové – gymnázia zaplatí 650 Kč. Více informací najdete na našich stránkách `www.cstug.cz`.

6. 4. 2003

Sazba XML

Jiří Kosek

V posledních měsících proběhlo v texové diskusní skupině i jinde několik diskusí o tom, zda je lepší při přípravě dokumentů používat TEX nebo XML. Pokusím se příliš nezneužít možnosti zaplnit stránky Zpravodaje názory, které jsem v diskuzích zastával já. Konec konců celá diskuse neměla moc smysl, protože TEX a XML spolu mohou žít bok po boku bez nějakých velkých problémů.

Asi se všichni shodneme, že TEX je výtečný nástroj pro sazbu dokumentů. Po dlouhá léta se dokumenty pro sazbu připravovaly přímo v TEXu. TEX tak sloužil jako jazyk pro zápis dokumentu a také jako systém pro jeho sazbu a zlom. Pokud je naším cílem jen dokument připravit, pěkně vysázet a vytisknout, je potom TEX vhodným a postačujícím nástrojem.

Pokud však chceme s připraveným dokumentem naložit i jinak, než jej jen vytisknout, třeba převést do HTML nebo z něj dotazem vybrat nějaká data, není už TEX jako formát pro primární pořízení dokumentu nejvhodnějším kandidátem. Vhodnější pro tyto účely jsou značkovací jazyky, v dnešní době především XML. S tím by možná někteří diskutéři nesouhlasili, ale zkusme to brát jako fakt. XML se zkrátka hodí pro ukládání informací, protože je lze dále poměrně snadno zpracovávat. TEX se hodí pro sazbu dokumentů, protože je rychlý a má implementovány kvalitní typografické algoritmy. Nic nám tedy nebrání ve využití výhod obou technologií – dokumenty ukládat do XML a pro potřeby tisku je zpracovávat TEXem. Získáme tak výhody obou světů.

Možností, jak sazbu dokumentu XML provést pomocí TEXu, je několik a my se je pokusíme přiblížit. V navazujících článcích jsou pak ty nejpoužívanější metody rozebrány více do hloubky včetně příkladů použití.

Přímá sazba XML

Asi nejjednodušší variantou je přímá sazba XML pomocí TEXu. Nepotřebujeme žádné přídavné programy, XML se zpracovává čistě prostředky TEXu. Tento přístup má však dva poměrně nepříjemné nedostatky.

Napsat parser XML přímo v TEXu je velmi obtížný úkol. Existující parsery jsou schopné načítat jen podmnožinu XML, obvykle nepodporují entity, které umožňují jeden dokument XML rozložit do více fyzických souborů.

Druhým nedostatkem je samotný princip, kdy musíme mapovat výskyty značek v XML na texové sekvence. V mnoha případech toto mapování můžeme poměrně jednoduše vytvořit, ale jsou případy, kdybychom v místě výskytu značky

potřebovali znát informace z jiné části dokumentu XML. Lze to samozřejmě obejít pomocí ukládání stavových informací nebo víceprůchodovým zpracováním dokumentu apod. Ale v porovnání se speciálními nástroji, které během zpracování umožňují přístup k celému dokumentu, je to velmi nepohodlné.

Pro přímou sazbu XML pomocí TEXu se dnes používají dva systémy. Prvním z nich je xmltex. Jedná se o jednoduchý parser implementovaný v TEXu, kterému můžeme v konfiguračním souboru určit, na jaké TEXové sekvence se mají převádět jednotlivé značky v dokumentu XML. Podrobnější popis xmltexu naleznete v samostatném článku *Použití parseru XML v TEXu* v tomto čísle Zpravodaje.

Druhým produktem, který si s XML poradí přímo, je ConTEXt. Tato uživatelsky příjemná makronadstavba nad TEXem obsahuje i nástroje na zpracování XML.

Konverze XML do TEXu

Přímou sazbu XML TEXem lze použít jen v jednoduchých situacích. I při složitějších požadavcích však nemusíme na použití TEXu rezignovat. Pro převod XML do podoby TEXového kódu můžeme použít sofistikovanější nástroje, které nám umožní pohodlné načtení XML a jeho převedení.

Asi nejvhodnějším nástrojem pro tuto úlohu je jazyk XSLT. Tento jazyk primárně slouží k popsání transformaci dokumentu XML do podoby HTML stránky, případně XML dokumentu s jinou strukturou. XSLT však s jistými omezeními umí generovat i obyčejné textové soubory, a tedy i TEXové dokumenty. XSLT pomocí dotazovacího jazyka XPath nabízí přístup k libovolné části dokumentu XML kdykoliv během zpracování a řeší tak pohodlně problémy jako je generování obsahu, křížových odkazů apod.

Napsání stylu XSLT pro převod XML do TEXu je poměrně jednoduché. Musíme si dát pozor jen na správné ošetření a převod znaků, které mají v TEXu speciální význam (např. `#`, `$`, `&`).

Pěknou ukázku použití XSLT při sazbě knihy z podkladů v XML jste mohli nalézt v minulém čísle Zpravodaje v článku Zdeňka Wagnera *Využití XML a LaTEXu při sazbě odborných knih.*

Využití standardních stylových jazyků

Výše popsané metody nám umožňují z dokumentů XML získat kvalitní tištěný výstup pomocí TEXu. Skalní příznivce značkovacích technologií by však tomuto přístupu přece jen jedno vytkl: formátovací předpis je úzce svázán se systémem TEX – nelze jej využít s jiným sázecím systémem a nemůžeme tak dostat jiný

výstup než formáty podporované (pdf)TEXem – DVI, PostScript a PDF. Co kdybychom však chtěli výstup v jiném formátu, jako je třeba RTF nebo HTML? Vytvořená konverze do TEXu by nám byla k ničemu.

Tento problém ve světě XML řeší stylové jazyky. Ty umožňují popsat vzhled dokumentu poměrně abstraktně a nezávisle na výstupním formátu a médiu. Takovýmto obecným předpisem – stylem – se může řídit převod do mnoha formátů v různých aplikacích. Stačí, když se standardizuje jazyk, kterým se styl popisuje. Je to jen přirozené pokračování snahy vytvořit nezávislý formát dat – tím je XML. Vzhled dokumentu by měl být také definován nezávisle, a to jak na dokumentu, tak na programu, který se použije pro výsledné formátování.

Stylových jazyků, které se hodí pro formátování XML dokumentů, existuje mnoho, ale mezi dva nejpoužívanější a „nejmocnější“ patří DSSSL a XSL. Shodou okolností pro oba dva jazyky existují i jejich implementace, které pro samotné formátování používají TEX. V případě DSSSL se jedná o JadeTEX a v případě XSL pak o PassiveTEX. I jejich základním principům a použití se věnujeme v samostatných článcích (články o JadeTEXu i o PassiveTEXu jsou v tomto čísle Zpravodaje).

Použití stylových jazyků přináší kromě nezávislosti na použitém formátovacím nástroji i další výhody. Většinu parametrů sazby lze nastavit jednoduše jako hodnoty vlastností a není potřeba pronikat do makrojazyka TEXu. DSSSL i XSL obsahují dotazovací jazyk, který umožňuje kdykoliv pracovat s libovolnou částí vstupního dokumentu XML – nejsme tak omezeni jako u TEXu, který vstupní soubor čte sekvenčně. Své výhody má i TEX – lze jej plně programovat, a jeho možnosti záleží jen na schopnostech uživatele, ne na jednou pevně daných vyjadřovacích možnostech standardizovaného stylového jazyka.

Další vývoj

Sazba XML pomocí TEXu je možná a v mnoha případech můžeme získat výborné výsledky. Nicméně jsou dvě oblasti, které by potřebovaly výrazně zlepšit. TEX je sám o sobě 8bitový a zpracování dokumentů používajících jako znakovou sadu Unicode mu činí potíže. Jde je sice obejít, ale postup je krkolomný a výsledek není vždy zcela robustní. Řešením může být přechod na systém Omega, který do TEXu přidává právě podporu Unicode.

Druhým omezením je samotný vstupní procesor TEXu. Jde s ním sice dělat kouzla, ale napsat v něm plnohodnotný parser XML nebo jiného formátu není příliš jednoduché. Vhodnější by byl modulární sázecí systém, kde by šlo použít uživatelské vstupní moduly speciálně připravené na čtení XML. Něco podobného by mělo umožňovat NTS.

Vše záleží na tom, jak rychle se budou „nové verze TEXu“ jako NTS a Omega vyvíjet, a jak rychle se rozšíří mezi uživateli. Bez jejich rozšíření se asi nikdo nebude obtěžovat s přepisováním stávajících maker pro tyto nové nadějné projekty.

Summary: Typesetting XML

This articles summarizes methods suitable for processing XML documents by the TEX system – direct typeseting (xmltex, ConTeXt), conversion to TEX (XSLT) and TEX based stylesheet language implementations (XSL, DSSSL). The article acts as an introduction for more detailed articles about processing XML with TEX.

Jiří Kosek
jirka@kosek.cz

Použití parseru XML v TEXu

Jiří Kosek

Chceme-li XML dokumenty sázet čistě prostředky TEXu bez nutnosti používání dalších pomocných programů, je asi nejvhodnějším nástrojem xmltex. Jedná se o makro nadstavbu na LATEXem, která implementuje parser XML. Ten umožňuje načítání dokumentů XML a umí podle pravidel definovaných v konfiguračním souboru převádět jednotlivé elementy na sekvence texového kódu, které pak řídí sazbu. Autorem xmltexu je David Carlisle, který je uznávaným odborníkem jak na oblast TEXu, tak i XML a zejména pak stylového jazyka XSL.

Většina moderních distribucí TEXu již xmltex obsahuje jako samostatný formát. Vysázení dokumentu XML je pak otázkou spuštění jediného příkazu:

```
xmltex dokument.xml
```

resp.

```
pdfxmltex dokument.xml
```

v případě, že chceme získat výstup v PDF pomocí pdfTEXu. Nemáme-li xmltex připravený jako samostatný formát, jde využít jednoduchý pomocný dokument v TEXu, který nadefinuje jméno dokumentu ke zpracování a poté aktivuje xmltex:

```
\def\xmlfile{dokument.xml}
\input xmltex.tex
\end{document}
```

Tento dokument se pak zpracuje pomocí klasického LATEXu.

Instalace

Samotný xmltex je již nějakou dobu standardní součástí texových distribucí jako je TEXLive nebo tetex. Pro první pokusy s ním tedy není potřeba nic složitě instalovat. Pokud xmltex není součástí vaší texové instalace, můžete si jej stáhnout z CTAN archivu nebo z adresy `http://www.dcarlisle.demon.co.uk/xmltex/` a nainstalovat podle přiložených instrukcí.

Jediným problémem standardně nainstalovaného xmltexu je chybějící podpora českého (a slovenského) dělení slov. Poměrně snadno ji však můžeme doplnit. Postup je popsán dále. Chcete-li si však jen vyzkoušet možnosti xmltexu a prozatím oželíte správné dělení slov, můžete zbytek této části článku přeskočit.

Formát xmltexu je odvozen z formátu LATEXu, který používá pro lokalizaci babel (používá se tedy jiná metoda počeštění než je CSLATEX). Pro správné dělení slov v naší mateřštině proto musíme do LATEXu zahrnout české vzory dělení slov, přegenerovat formát LATEXu a následně i formát xmltexu.

Nejprve musíme definovat, jaké vzory dělení slov ze zahrnou do formátu LATEXu. Definice je uložena v souboru `language.dat`, který se typicky nachází v adresáři `texmf/tex/generic/config`. V souboru musíme odkomentovat řádku, která se stará o načtení českých vzorů dělení slov:

```
czech           czhyph2e.tex
```

Nyní musíme přegenerovat formát LATEXu a xmltexu. Nejjednodušší je použít příkaz `fmtutil --all`, který se postará o přegenerování všech formátů. Ve windowsové verzi TEXLive je příkaz dostupný i v menu *TeXLive* → *Maintenance* → *Create all format files*.

TEXLive tímto způsobem rovnou přegeneruje i formát pro xmltex, distribuci tetex o to musíme požádat ručně:

```
fmtutil --cnffile /usr/share/texmf/tex/xmltex/xmltexfmtutil.cnf --all
```

Od této chvíle by měly být programy `xmltex` a `pdfxmltex` schopné používat české dělení slov.

Vazba mezi XML a TEXem

V okamžiku, kdy xmltexu předáme dokument XML ke zpracování, snaží se xmltex najít definiční soubor, popisující mapování XML na texové sekvence. Tyto soubory mají obvykle příponu `.xmt`. Vazba jednotlivých druhů dokumentů XML na xmt-soubory je definována v katalogu. Vždy se prohledává globální katalog, který musí mít jméno `xmltex.cfg`, a lokální, který se jmenuje stejně jako dokument XML, ale má příponu `.cfg`.

Asi nejspolehlivější způsob svázání XML dokumentů s xmt-souborem je pomocí jmenného prostoru, který lze využít v případě, že zpracovávaný dokument

používá jmenné prostory. Dejme tomu, že máme v XML uložený článek a používáme i jmenné prostory.

```
<clanek xmlns="urn:x-kosek:schemas:clanek:v1.0">
 ...
</clanek>
```

Uložíme-li pak definiční soubor popisující převod elementů článku na texové sekvence například do souboru `clanek.xmt`, musíme do katalogu přidat následující řádku:

```
\NAMESPACE{urn:x-kosek:schemas:clanek:v1.0}{clanek.xmt}
```

Bohužel v mnoha jednodušších případech se jmenné prostory nepoužívají a jednotlivé typy dokumentů XML nelze jednoznačně identifikovat. V těchto případech pak můžeme vazbu na xmt-soubor vytvořit na základě jména kořenového elementu:

```
\NAME{clanek}{clanek.xmt}
```

Musíme být však opatrní, zvláště při použití této deklarace v globálním katalogu, protože se může stát, že budeme mít dva různé druhy dokumentů se stejným kořenovým elementem. xmltex pak nebude mít možnost, jak je správně rozlišit.

Definiční xmt-soubory

Nyní se dostáváme k nejdůležitější části xmltexu, a to je vazba mezi elementy XML a TEXem. V xmt-souboru můžeme pro každý element definovat kód, kterým se nahradí výskyt počátečního a koncového tagu. Základní kostra definice je velmi jednoduchá:

```
\XMLelement{název elementu}
  {zachycení atributů}
  {kód pro začátek elementu}
  {kód pro konec elementu}
```

Ukažme si konkrétní použití na jednoduchém dokumentu XML s článkem. Předpokládejme, že obsah souboru `dokument.xml` je

```
<?xml version="1.0" encoding="iso-8859-2"?>
<clanek>
  <zahlavi>
    <rubrika>Aktuality</rubrika>
    <nazev>Státní správa bude používat TeX místo Wordu</nazev>
    <autor email="wild@duck.cz">Jan Novák</autor>
  </zahlavi>
  <perex>... text perexu ...</perex>
```

```
  <para>... text odstavce ...</para>
  <para>... text odstavce ...
        ... <em>zvýrazněný text</em> ...
        ... </para>
  <para>... text odstavce ...</para>
</clanek>
```

Vysázet tento článek v LaTeXu je velmi jednoduché:

```
\documentclass{article}
\usepackage[czech]{babel}
\gdef\email#1{\\{\small\ttfamily#1}\par}
\begin{document}

\title{Státní správa bude používat TeX místo Wordu}
\author{Jan Novák \email{wild@duck.cz}}
\date{}
\maketitle

\noindent{\bf ... text perexu ...}\par

... text odstavce ...

... text odstavce ...
... {\it zvýrazněný text\/} ...
...

... text odstavce ...

\end{document}
```

V našem případě si dokument XML s latexovým kódem téměř odpovídá, proto je popsání definice mapování elementů XML velmi jednoduché.

Začneme povinnou kostrou dokumentu s latexovou preambulí. Ta se musí objevit v každém dokumentu. Nejpohodlněji ji vygenerujeme v definici pro kořenový element `clanek`:

```
\XMLelement{clanek}{}
        {\documentclass{article}
         \usepackage[czech]{babel}
         \gdef\email##1{\\{\small\ttfamily##1}\par}
         \begin{document}}
        {\end{document}}
```

Při nalezení počátečního tagu se vygeneruje preambule, po nalezení koncového se naopak ukončí dokument. Protože je `\XMLelement` makro, musíme

uvnitř něj při definici vlastních maker s parametry zdvojit `#`. Element `clanek` nemá žádné atributy, a proto jsme druhý parametr makra ponechali prázdný.

Nejjednodušší obsluhu budou mít elementy jako `para`. Na jejich začátku nemusíme dělat nic, a na konci stačí ukončit odstavec pomocí `\par`.

```
\XMLelement{para}{}{}{\par}
```

Podobně můžeme obsloužit i element `em` používaný pro zvýraznění. Na začátku elementu aktivujeme kurzívu (`\it`) a na konec vložíme italickou korekci.

```
\XMLelement{em}{}{\it}{\/}
```

xmltex automaticky vše uzavře do skupiny, takže se nemusíme bát, že od tohoto místa bude kurzívou vysázen i celý zbytek dokumentu.

V definici obsluhy začátku elementu můžeme použít i speciální makro `\xmlgrab`. To načte celý obsah elementu do parametru `#1` a zpřístupní jej v obsluze koncového tagu. Obsluhu elementu `em` proto můžeme alternativně zapsat i jako:

```
\XMLelement{em}{}{\xmlgrab}{\it #1\/}
```

`\xmlgrab` můžeme využít i v případech, kdy chceme obsah nějakého elementu ignorovat. Jestliže nechceme, aby se ve vysázeném výsledku objevil název rubriky, zkrátka jej necháme „sníst“ tím, že v obsluze koncového tagu nepoužijeme obsah nastřádaný v `#1`:

```
\XMLelement{rubrika}{}{\xmlgrab}{}
```

Chceme-li u nějakého elementu zpracovat atributy, musíme si definovat, jaké atributy nám mají být dostupné. Slouží k tomu makro `\XMLattribute` se třemi parametry. První udává jméno atributu v dokumentu XML, druhý texový název, pod nímž bude hodnota atributu dostupná v TEXu, a třetí parametr může určit hodnotu, která se má použít v případě nepřítomnosti atributu:

```
\XMLelement{autor}
        {\XMLattribute{email}{\emailAutora}{}}
        {E-mail je: \emailAutora}
        {}
```

V našem konkrétním případě je situace složitější, protože chceme mailovou adresu předat dovnitř makra `\autor`, které si pak LATEX volá později pro vysázení nadpisu článku. V tom okamžiku však již nebude dočasně nadefinované makro `\emailAutora` platné. Musíme si jej proto uložit do globálního makra (`\gEmailAutora`), které se při definici rovnou expanduje.

```
\XMLelement{autor}
        {\XMLattribute{email}{\emailAutora}{\relax}}
        {\xmlgrab}
        {\xdef\gEmailAutora{\emailAutora}
         \author{#1
           \ifx\gEmailAutora\relax
```

```
\else
  \email{\gEmailAutora}
\fi
}}
```

Zároveň ještě testujeme, zda je atribut vůbec v dokumentu nastaven, abychom zbytečně nesázeli prázdnou e-mailovou adresu.

Chceme-li si vše vyzkoušet, můžeme definice obsluhy elementů uložit do souboru `clanek.xmt` a vytvořit si jednoduchý katalog `dokument.cfg`:

```
\NAME{clanek}{clanek.xmt}
```

Příkazem

```
pdfxmltex dokument.xml
```

tak získáme vysázenou podobu našeho článku v XML. Anebo ne? Na první pohled je vše v pořádku, ale při bližším ohledání zjistíme, že vypadla některá písmena s diakritikou – třeba 'ž', 'ř' a 'č'. Problém je v tom, že XML jako znakovou sadu používá Unicode. xmltex však umí standardně na výstup korektně zapsat jen znaky, které spadají do kódování ISO Latin 1 (tedy znaky západoevropských jazyků).

Naštěstí není nijak složité přidat podporu pro chybějící znaky. Do katalogu (soubor s příponou `.cfg`) můžeme dodefinovat chybějící znaky pomocí makra `\UnicodeCharacter`. Makro má dva parametry – prvním je unicodový kód znaku a druhým sekvence texových příkazů, kterými se znak nahradí. Chybějící písmeno 'ž' (s kódem 382 decimálně) tak můžeme dodefinovat pomocí:

```
\UnicodeCharacter{382}{\v{z}}
```

Pokud nám hrozí, že písmeno použijeme i v matematickém režimu, je bezpečnější definice ve tvaru:

```
\UnicodeCharacter{382}{\ifmmode \check{z}\else \v{z}\fi}
```

Pro lenochy jako já je tu ještě druhá možnost. Součástí PassiveTEXu jsou již připravené definice pokrývající slušnou podmnožinu celého Unicode. Načítají se však až přímo do dokumentu, nevkládají se do katalogu. Do preambule dokumentu stačí přidat:

```
\RequirePackage{unicode}
\RequirePackage{ucharacters}
```

Pro ilustraci se nyní podívejme na celý definiční soubor pro náš článek.

```
\XMLelement{clanek}{}
        {\documentclass{article}
         \usepackage[czech]{babel}
         \RequirePackage{unicode}
         \RequirePackage{ucharacters}
         \usepackage{palatino}
         \gdef\email##1{\\{\small\ttfamily##1}\par}
```

```
          \begin{document}}
         {\end{document}}

\XMLelement{zahlavi}{}{}{\date{}\maketitle}

\XMLelement{nazev}{}{\xmlgrab}{\title{#1}}

\XMLelement{autor}
         {\XMLattribute{email}{\emailAutora}{\relax}}
         {\xmlgrab}
         {\xdef\gEmailAutora{\emailAutora}
          \author{#1
            \ifx\gEmailAutora\relax
            \else
              \email{\gEmailAutora}
            \fi
            }}

\XMLelement{rubrika}{}{\xmlgrab}{}

\XMLelement{para}{}{}{\par}

\XMLelement{perex}{}{\noindent\bf}{\par}

\XMLelement{em}{}{\it}{\/}
```

Další možnosti

Již toho známe dost, abychom mohli pomocí xmltexu formátovat jednoduché dokumenty XML. V následujících odstavcích se podíváme ještě na pár drobností, které xmltex umí a které se nám mohou v některých případech hodit.

Během formátování dokumentu vypisuje xmltex do logu informace o struktuře zpracovávaného dokumentu. Zejména u delších dokumentů to může být docela rušivé.

```
<0:clanek (./clanek.xmt)  >
 <0:zahlavi  >
  <0:rubrika  >
  </0:rubrika>
  Grabbed content
  End Grabbed content
  <0:nazev  >
```

```
  </0:nazev>
  Grabbed content
  End Grabbed content
  <0:autor
      0:email = "wild@duck.cz"  >
  </0:autor>
  Grabbed content
  End Grabbed content
 </0:zahlavi>
 <0:perex  >
 </0:perex>
 <0:para  >
  <0:em  >
  </0:em>
 </0:para>
 <0:para  >
 </0:para>
 <0:para  >
 </0:para>
</0:clanek>
```

Chceme-li se hlášení zbavit, stačí do katalogu přidat `\xmltraceoff`.

Ve výpisech je také vidět, že před názvy všech elementů je ještě uvedeno `0:`. Tím nám xmltex dává najevo, že element nepatří do žádného jmenného prostoru. Pokud však jmenné prostory používáme, musíme si pro ně v definičním souboru definovat zástupný prefix, kterým se pak dále identifikují elementy a atributy patřící do daného jmenného prostoru. Např.:

```
\DeclareNamespace{m}{http://www.w3.org/1998/Math/MathML}
\XMLelement{m:math}{...}
\XMLelement{m:mi}{...}
```

Poněkud v rozporu s XML specifikací, za to velmi užitečně, umožňuje xmltex předefinování entit uvedených v lokálních deklaracích XML dokumentu. Ukažme si vše na malém příkladě. Dejme tomu, že se v textu píše často o TEXu a my si proto pro něj deklarujeme interní textovou entitu:

```
<!DOCTYPE clanek [
<!ENTITY TeX "TeX">
]>
<clanek>
  <zahlavi>
    <nazev>Státní správa bude používat &TeX; místo Wordu</nazev>
  ...
</clanek>
```

Při zpracování dokumentu nahradí parser odkaz na entitu `&TeX;` řetězcem „TeX“. To je většině případů žádoucí, protože kromě TEXu neumí jiné systémy logo správně zobrazit (například webový prohlížeč). Pro účely zpracování xmltexem si však můžeme do definičního souboru přidat předefinování entity. Ta nyní bude volat makro pro správné vysázení názvu našeho oblíbeného typografického systému:

```
\XMLentity{TeX}{\TeX{}}
```

Odkazy na entitu se nyní nahradí za „TEX“ a dostaneme tak požadovaný výstup.

Sám autor xmltexu o něm tvrdí, že to není procesor XML, který by 100% odpovídal standardu. Jedním z problémů je chybějící podpora načítání dokumentů kódovaných v UTF-16. V praxi to naštěstí příliš limitující není, protože toto kódování není moc používané. Standardně jsou podporována kódování UTF-8 a ISO Latin 1. Novější verze jsou distribuovány i s podporou středoevropských kódování ISO Latin 2 a windows-1250. Není problém přidat podporu dalších osmibitových kódování. Stačí vytvořit soubor `jméno kódování.xmt` a definovat v něm překódovací tabulku ve tvaru:

```
\InputCharacter{kód}{kód Unicode}
```

Závěr

V článku jsem se pokusil shrnout a popsat základní principy přímé sazby dokumentů XML pomocí parseru napsaného čistě v TEXu. Je to přístup vhodný zejména pro jednodušší dokumenty, které již obsahují text víceméně v tom pořadí, jak se má objevit vysázený na výstupu. V případech, kdy je potřeba před zpracováním dokumentu provést jeho složitější transformace, je vhodnější ke konverzi XML do TEXu použít specializované nástroje, například XSLT transformaci.

Summary: Use of an XML parser in TEX

This article shows how to use xmltex—an XML parser written in a pure TEX—to directly typeset XML documents. Special interest is devoted to correct processing of localized Czech/Slovak documents.

Jiří Kosek
`jirka@kosek.cz`

JadeTEX

Jiří Kosek

JadeTEX je speciální formát pro TEX, který umožňuje formátování SGML a XML dokumentů pomocí stylů zapsaných v jazyce DSSSL. JadeTEX neumí interpretovat DSSSL styl přímo, dokumenty SGML/XML musíme nejprve zpracovat pomocí programu Jade, který vytvoří meziprodukt vhodný pro zpracování JadeTEXem.

DSSSL

Stylový jazyk DSSSL (Document Style Semantics and Specification Language) vznikl jako výsledek snahy o vytvoření jazyka, který umožní popsat formátování třídy dokumentů SGML bez ohledu na konkrétní výstupní formát. To umožní vytvářet předpis formátování nezávislý na konkrétní aplikaci použité pro formátování dokumentu. V roce 1996 byla tato snaha završena vydáním DSSSL jako ISO/IEC mezinárodní normy 10179.

DSSSL definuje v podstatě dva jazyky. První z nich – transformační – umožňuje popsat transformaci SGML dokumentu do jiného SGML dokumentu. To jde využít například při převodu SGML do HTML. Druhý jazyk umožňuje formátování dokumentů tím, že se jednotlivým částem vstupního dokumentu přiřadí prezentační charakteristiky jako zarovnání, písmo, barva apod. Z tohoto abstraktního předpisu pak formátovač vytvoří výsledný vysázený dokument. Řídí se přitom charakteristikami definovanými ve stylu s tím, že se rozhoduje o takových věcech jako jsou řádkové a stránkové zlomy.

Aby šlo v DSSSL vyjadřovat složité transformace a formátování, obsahuje ještě dotazovací jazyk a jazyk pro zápis výrazů. Dotazovací jazyk umožňuje pohyb po stromu dokumentu SGML a výběr jeho jednotlivých částí. Jazyk pro zápis výrazů vychází ze Scheme a je hodně podobný LISPu, který je v našich krajích známější než Scheme.

Jade/OpenJade

Abychom mohli na SGML aplikovat DSSSL styl, musíme mít k dispozici program, který umí pravidla ze stylu aplikovat na dokument. Takovému programu se obvykle říká procesor DSSSL. Jednoznačně nejznámějším je program Jade od Jamese Clarka. Protože na jeho vývoji již James několik let nepracuje, vzniklo na

jeho základě OpenJade, které velmi pozvolným tempem přidává nové vlastnosti a opravuje chyby. Jade/OpenJade[1] jsou open source programy napsané v C++ a jsou k dispozici pro všechny běžně používané platformy, Windows a Linux nevyjímaje.

Jade v sobě obsahuje několik koncových modulů, které jsou schopné výsledek formátování uložit do různých formátů. Mezi tyto formáty patří RTF, MIF (používá ho DTP systém FrameMaker od Adobe) a TEX. V případě TEXu se však nejedná o nějaký běžný formát jako plainTEX či LATEX, ale o dost specifický kód, který jde zpracovat jen JadeTEXem. JadeTEX je tedy jen poslední článek v řetězci, který umožňuje SGML dokumenty pomocí DSSSL převést do tištěné podoby.

Poznamenejme ještě, že Jade umí kromě SGML číst i XML dokumenty (XML je konec konců podmnožina SGML), takže jej lze použít i pro zpracování XML. Dále v textu proto budu mluvit o XML, i když popsaným způsobem jde zpracovat i dokumenty SGML.

Instalace

Jak již bylo řečeno, pro práci s JadeTEXem potřebujeme i program Jade. Můžeme si jej stáhnout z adresy `http://www.jclark.com/jade/`, nebo si můžeme pořídit jeho novějšího následníka OpenJade (`http://openjade.sourceforge.net`). Tuto možnost využijí zejména uživatelé Windows, protože většina linuxových distribucí již obsahuje Jade v podobě balíčku.

Pro správnou funkci Jade je po instalaci ještě potřeba upravit katalogové soubory pro SGML. Do proměnné prostředí `SGML_CATALOG_FILES` uložíme cestu ke katalogovému souboru pro Jade (obvykle `c:\jade\catalog` nebo `c:\openjade\dsssl\catalog`).

JadeTEX je formát TEXu a bývá běžnou součástí modernějších distribucí jako TEXLive a teTEX. Aktuální verzi si můžete stáhnout na adrese `http://jadetex.sourceforge.net`.

Pravidla

Základem DSSSL stylu jsou pravidla. Pravidlo definuje, jak se má určitá část vstupního dokumentu XML zpracovat. Při zpracování dokumentu pak Jade postupně prochází jeho jednotlivé elementy a hledá pravidla, která definují jejich zpracování. Pravidlo má tvar:

```
(element název elementu
        zpracování elementu)
```

[1] Dále v textu bude již používat jen název Jade, i když vše uvedené se týká i OpenJade.

Objekty toku dokumentu

DSSSL styl definuje pravidla, které popíší způsob převodu XML dokumentu do formátované podoby. Aby šlo výsledný vzhled dokumentu popsat neutrálně, nezávisle na nějakém konkrétním formátovacím systému, používá DSSSL abstraktní reprezentaci vzhledu dokumentu v podobě objektů toku (flow object). Dokument můžeme popsat jako sekvenci objektů několika různých tříd. K dispozici jsou například objekty definující rozměry stránky, vytvářející odstavec textu, tabulku, seznam či obrázek. Dokument se pak definuje jako sekvence takových objektů. V žargonu DSSSL se tomu říká sosofo – specifikace sekvence objektů toku.

Úkolem pravidel ve stylu je právě generovat pro jednotlivé elementy sekvence objektů toku. Kdybychom chtěli ve vstupním dokumentu převést element `para` na odstavec na výstupu, stačí v pravidle definovat, že pro element se má vytvořit objekt odstavce.

```
(element para
  (make paragraph
    (process-children)))
```

Pravidlo říká, že pro element `para` se vytvoří odstavec a jako jeho obsah se použije výsledek zpracování obsahu elementu dalšími pravidly (`process-children`).

Charakteristiky objektů

Při generování jednotlivých objektů samozřejmě chceme ovlivnit, jak budou vypadat. U odstavce asi budeme chtít nastavit použité písmo, řádkování, mezery před a za odstavcem, způsob zarovnání apod. U každého objektu proto můžeme nastavit několik charakteristik.

```
(element para
  (make paragraph
    font-size: 12pt
    line-spacing: 14pt
    quadding: 'justify
    font-posture: 'italic
    (process-children)))
```

Dotazovací jazyk

Použijeme-li ve všech pravidlech pouze `process-children`, zpracuje se dokument přesně v tom pořadí, jak byl vytvořen. V praxi však existují situace, kdy

potřebujeme některé části dokumentu vynechat nebo je použít na jiném místě. Pro tyto účely obsahuje DSSSL i dotazovací jazyk, který operuje nad stromovou reprezentací vstupního dokumentu XML.

Dotazovací jazyk umožňuje pohyb po vstupním dokumentu a výběr jeho částí. Například kdybychom chtěli vybrat mezi všemi potomky právě zpracovávaného elementu všechny elementy s názvem `obrazek`, zapíšeme to v DSSSL jako:

```
(select-elements (descendants (current-node)) "obrazek")
```

`current-node` zastupuje aktuální element, `descendants` vybere všechny potomky aktuálního elementu a mezi nimi pak pomocí funkce `select-elements` vybereme všechny s názvem `obrazek`.

Praktická ukázka

Ukažme si teď na jednoduchém příkladě kompletní funkční styl v DSSSL. Předpokládejme, že chceme definovat formátování pro články zapisované v XML. Každý článek má přitom následující strukturu:

```
<?xml version="1.0" encoding="iso-8859-2"?>
<!DOCTYPE clanek SYSTEM "clanek.dtd">
<clanek>
  <zahlavi>
    <rubrika>Aktuality</rubrika>
    <nazev>Státní správa bude používat TeX místo Wordu</nazev>
    <autor email="wild@duck.cz">Jan Novák</autor>
  </zahlavi>
  <perex>... text perexu ...</perex>
  <para>... text odstavce ...</para>
  <para>... text odstavce ...
        ... <em>zvýrazněný text</em> ...
        ... </para>
  <para>... text odstavce ...</para>
</clanek>
```

Chceme přitom nadpis článku vysázet větším písmem a vycentrovat, rubrika článku by se měla objevit v záhlaví a jméno a případný e-mail autora až na konci článku. Může nám k tomu posloužit následující styl:

```
<?xml version="1.0" encoding="iso-8859-2"?>
<!DOCTYPE style-sheet
          PUBLIC "-//James Clark//DTD DSSSL Style Sheet//EN">
<style-sheet>
<style-specification>
<style-specification-body>
```

```
;; základní nastavení charakteristik
(declare-initial-value writing-mode 'left-to-right)
(declare-initial-value font-size 12pt)
(declare-initial-value line-spacing 14pt)
(declare-initial-value font-family-name "Arial")
(declare-initial-value language 'CS)
(declare-initial-value quadding 'justify)
(declare-initial-value hyphenate? #t)

;; obsluha kořene stromu XML dokumentu
(element clanek
    (make simple-page-sequence
      ;; rozměry stránky a velikost okrajů
      page-width: 210mm
      page-height: 297mm
      left-margin: 1in
      right-margin: 1in
      top-margin: 1in
      bottom-margin: 1in
      footer-margin: 0.5in
      header-margin: 0.5in
      ;; obsah záhlaví (název rubriky + čára)
      right-header: (make sequence
                      (literal "Rubrika: "
                               (data (select-elements
                                      (descendants (current-node))
                                      "rubrika")))
                      (make rule
                        orientation: 'horizontal
                        line-thickness: 0.4pt
                        ))
      ;; obsah zápatí (číslo strany)
      center-footer: (make sequence
                       font-posture: 'italic
                       (literal "- ")
                       (page-number-sosofo)
                       (literal " -"))
      ;; obsluha vnořených elementů
      (process-children)
      ;; za textem článku se zobrazí jméno autora
      (make paragraph
        space-before: 12pt
        quadding: 'end
```

```
        font-posture: 'italic
        (literal
         "\U-2013; "     ;; Unicode kód pomlčky
         (data (select-elements
                (descendants (current-node))
                "autor"))
         " \U-2013;"
        ))
      ;; jestliže má autor e-mail, zobrazí se
      (if (attribute-string "email" (select-elements
                (descendants (current-node))
                "autor"))
          (make paragraph
            quadding: 'end
            font-size: 10pt
            font-posture: 'italic
            (make sequence
              font-posture: 'normal
              (literal "e-mail: "))
            (literal
             (attribute-string "email" (select-elements
                                        (descendants (current-node))
                                        "autor"))
             ))
          (empty-sosofo))))

;; rubriku si pro zobrazení vybíráme sami,
;; proto ji standardně ignorujeme
(element rubrika
  (empty-sosofo))

;; autora si pro zobrazení vybíráme sami,
;; proto jej standardně ignorujeme
(element autor
  (empty-sosofo))

;; zpracování názvu článku
(element nazev
  (make paragraph
    quadding: 'center
    font-size: 24pt
    line-spacing: 28pt
    font-weight: 'bold
    space-after: 12pt
```

```
    (process-children)))

;; zpracování perexu
(element perex
  (make paragraph
    font-weight: 'bold
    font-posture: 'italic
    (process-children)))

;; zpracování odstavců
(element para
  (make paragraph
    first-line-start-indent: 18pt
    (process-children)))

;; zvýrazněný text bude kurzívou
(element em
  (make sequence
    font-posture: 'italic
    (process-children)))

</style-specification-body>
</style-specification>
</style-sheet>
```

První zajímavostí stylu, o které jsme se ještě nezmínili, je jeho zápis. I když jádro stylu se vyjadřuje pomocí pravidel zapsaných v jazyce odvozeném od Scheme, styl jako celek má podobu SGML dokumentu. Proto jsou pravidla „obalena“ několika elementy.

Stručně se zmíníme o významu některých klíčových míst stylu. Pomocí `declare-initial-value` můžeme pro mnoho charakteristik nastavit výchozí hodnoty. To je vhodné zejména pro věci, které budou společné pro celý dokument nebo alespoň většinu jeho částí.

Poněkud nezvykle nazvaná charakteristika `quadding` určuje způsob zarovnání. Pro zarovnání do bloku se použije hodnota `'justify`. Zarovnání doleva a doprava se vyjadřuje pomocí `'start` a `'end`. Toto pojmenování je voleno, aby bylo nezávislé na směru sazby textu.

Kořenový element dokumentu generuje objekt `simple-page-sequence`, který umí na výstupu generovat sekvence stránek. Jako charakteristiky objektu lze mimo jiné uvést velikost okrajů nebo definovat obsah záhlaví a zápatí (charakteristiky `*-header` a `*-footer`).

Předpokládejme nyní, že ukázkový dokument XML máme uložen v souboru `dokument.xml` a styl v souboru `clanek.dsl`. Chceme získat formátovaný výsledek pomocí JadeTEXu. Nejprve musíme převést dokument XML do podoby vhodné pro JadeTEX:

```
jade -d clanek.dsl -t tex /cesta/k/xml.dcl dokument.xml
```

Parametr `-d` určuje styl s předpisem formátování, `-t` požadovaný výstupní formát. Soubor `xml.dcl` je deklarace SGML pro XML. Tento soubor je nutný, aby se Jade navržený pro práci s SGML vypořádal s dokumenty XML. Soubor je součástí Jade, v Linuxu ho nejčastěji najdete v `/usr/share/sgml/xml.dcl`, ve Windows pak `c:\jade\xml.dcl`, případně `c:\openjade\pubtext\xml.dcl`.

Dostaneme tak texový soubor, který však není určen pro lidské oči:

```
\FOT{2}\Seq%
{\def\WritingMode%
{lefttoright}\def\fSize%
{12\p@}\def\LineSpacing%
{14\p@}\def\LineSpacingFactor%
{0}\def\fFamName{Arial}\def\Language%
{CS}\def\Quadding%
{justify}\def\Hyphenate%
{1}}\Node%
{}\Node%
{\def\Element%
{0}}\SpS%
{\def\PageWidth%
{595.275\p@}\def\PageHeight%
{841.889\p@}\def\LeftMargin%
{72\p@}\def\RightMargin%
{72\p@}\def\TopMargin%
{72\p@}\def\BottomMargin%
{72\p@}\def\FooterMargin%
{36\p@}\def\HeaderMargin%
{36\p@}}
\SpSOtherBackLeftFooter%
... zkráceno ...
```

Můžeme jej však pomocí JadeTEXu přeložit do PDF nebo DVI:

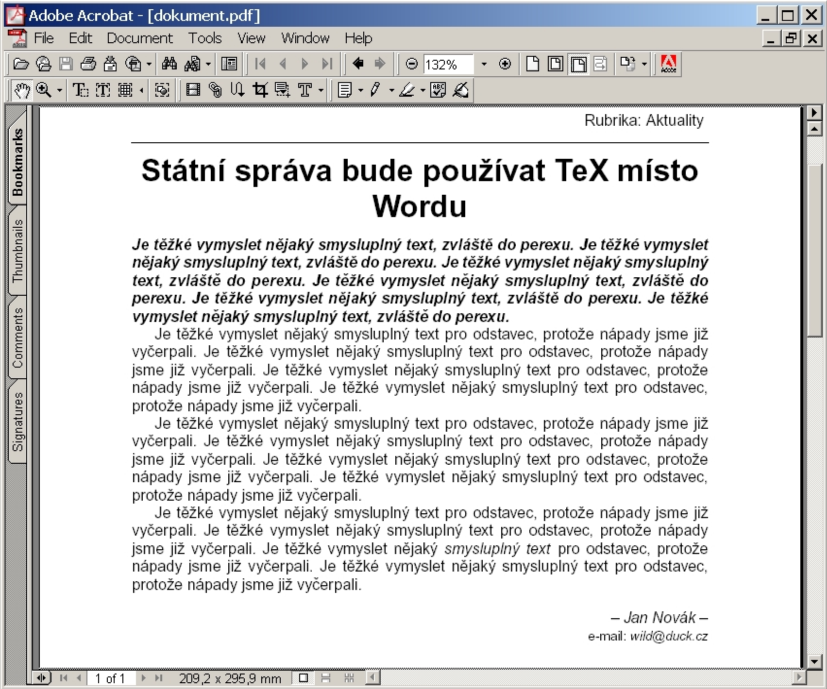
```
pdfjadetex dokument
```

Z našeho dokumentu XML tak dostaneme PDF výstup (viz obrázek 1). Obrázek 2 ukazuje stejný dokument zpracovaný stejným stylem, ale převedený do formátu RTF pomocí příkazu:

```
jade -d clanek.dsl -t rtf /cesta/k/xml.dcl dokument.xml
```

Nezobrazí-li se nám na výstupu správně české znaky, stačí nastavit proměnnou prostředí `SP_ENCODING` na identifikátor použitého kódování (`iso-8859-2` nebo `windows`). Další možností je nastavit proměnnou na hodnotu `XML`. Kódování se pak určí podle XML deklarace na začátku každého dokumentu:

```
<?xml version="1.0" encoding="..."?>
```

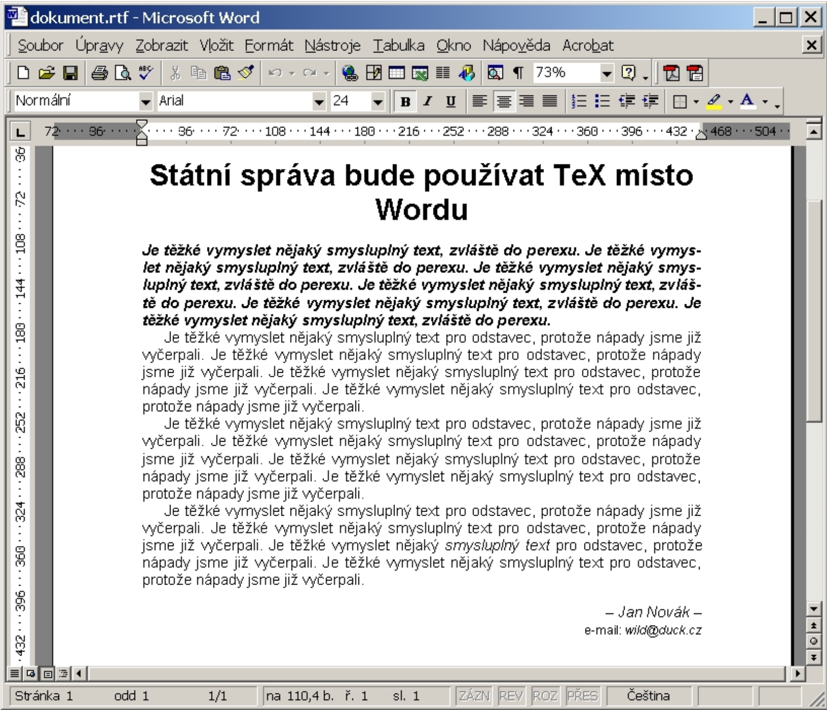


Obrázek 1: Výsledek převodu do PDF pomocí JadeTEXu

Sazba matematiky

DSSSL obsahuje i základní objekty pro vytváření matematické sazby. Můžeme vytvářet zlomky, odmocniny, závorky, operátory (jako suma nebo limita) a matice. Předpokladem samozřejmě je, že ve vstupním dokumentu máme matematiku označkovánu. Pro ilustraci jsem si vymyslel jednoduchou (a v praxi dost nepoužitelnou) notaci pro zápis vzorců obsahujících zlomky a odmocniny:

```
<vzorec>
  <odmocnina>
   <zlomek>
     <citatel>1</citatel>
     <jmenovatel>2</jmenovatel>
   </zlomek>
  </odmocnina>
</vzorec>
```



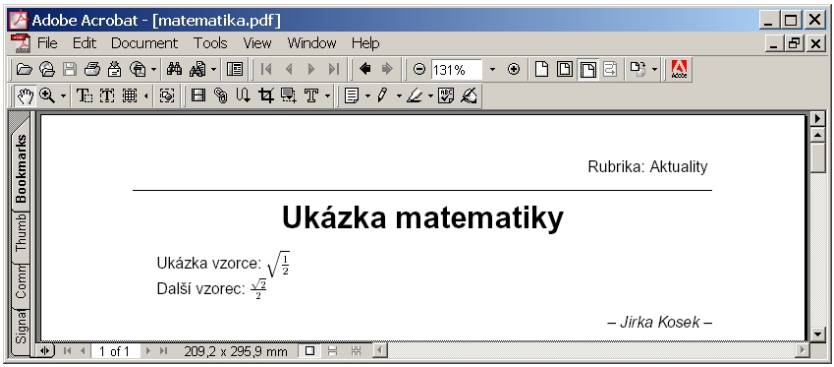
Obrázek 2: Stejný dokument a styl, tentokráte však ve formátu RTF

```
<vzorec>
  <zlomek>
    <citatel>
      <odmocnina>2</odmocnina>
    </citatel>
    <jmenovatel>2</jmenovatel>
  </zlomek>
</vzorec>
```

Obsluha těchto elementů v DSSSL je velmi přímočará, protože je lze snadno mapovat na jednotlivé matematické objekty toku dokumentu.

```
(element vzorec
  (make math-sequence
    math-display-mode: 'inline
    (process-children)))
```

```
(element odmocnina
  (make radical
    (process-children)))

(element zlomek
  (make fraction
    (process-children-trim)))

(element citatel
  (make math-sequence
    label: 'numerator
    (process-children)))

(element jmenovatel
  (make math-sequence
    label: 'denominator
    (process-children)))
```



Obrázek 3: Ukázka formátování matematických vzorců

Pro zápis matematiky v XML existuje standardizovaný jazyk MathML[2]. Na adrese `http://www.nag.co.uk/projects/OpenMath/mml-files/` jsou k dispozici styly DSSSL, které umějí zpracovat podmnožinu MathML.

[2] `http://www.w3.org/Math/`

Závěr

Jazyk DSSSL je zajímavou možností pro formátování XML a SGML dokumentů, která umožňuje zcela oddělit obsah dokumentu od jeho vzhledu. Již dnes je však jasné, že mnohem lépe byl přijat novější stylový jazyk XSL a ve většině aplikací se jím DSSSL postupně nahrazuje.

Odkazy

[1] *ISO/IEC 10179. Information technology – Processing languages – Document Style Semantics and Specification Language (DSSSL).* ISO/IEC, 1996. URL: `ftp://ftp.ornl.gov/pub/sgml/WG8/DSSSL/`

Summary: JadeTEX

JadeTeX is a TeX macro package which is able to process SGML and XML documents according to a DSSSL stylesheet in conjunction with (Open)Jade DSSSL processor. This article briefly describes basic principles of the DSSSL language and its usage for formatting XML documents. Complete working example of a DSSSL stylesheet is shown in the article.

Jiří Kosek
`jirka@kosek.cz`

PassiveTEX

Jiří Kosek

PassiveTEX je implementace jazyka XSL FO založená na TEXu. PassiveTEX umí vysázet dokument, jehož obsah a vzhled je popsán speciálním dokumentem XML, který jako značky používá tzv. formátovací objekty. Pojďme se však nejprve stručně seznámit se samotným XSL, abychom lépe pochopili, jaká je role PassiveTEXu při zpracování dokumentů XML.

Princip XSL

Na jazyku pro formátování XML dokumentů se začalo pracovat v podstatě ve stejné době jako na samotném jazyce XML. Poměrně brzy se přišlo na to, že

pořádný stylový jazyk se skládá ze dvou částí. Stylový jazyk musí umět jednotlivým částem dokumentu přiřadit jejich formátovací vlastnosti. Před tímto přiřazením je však v mnoha případech potřeba provést ještě transformaci vstupního dokumentu. Transformace se použije pro takové operace jako vygenerování obsahu či rejstříku, očíslování kapitol a obrázků nebo pro sečtení položek na faktuře – záleží na našich potřebách a na vstupním dokumentu.

Tento přístup reflektuje i samotný jazyk XSL. Skládá se ze dvou samostatných částí – transformačního jazyka XSLT a tzv. formátovacích objektů (FO). Podobně jako XML i XSL je standardizováno v rámci W3C. XSLT bylo jako doporučení přijato v roce 1999 a formátovací objekty až o dva roky později (2001).

XSLT styl umožňuje popsat transformaci XML dokumentu do HTML, XML s jinou strukturou, případně do obyčejného textového souboru. Transformace je přitom popsána sadou šablon, které definují převod jednotlivých částí XML dokumentu, jež jsou identifikovány pomocí XPath výrazu. XPath je jednoduchý, ale velmi silný dotazovací jazyk nad stromovou reprezentací XML. Pokud chceme nějaký XML dokument transformovat pomocí XSLT, musíme mít k dispozici tzv. XSLT procesor. Mezi nejpoužívanější dnes patří asi Saxon, xsltproc a Xalan, ale existují mnohé další.[1]

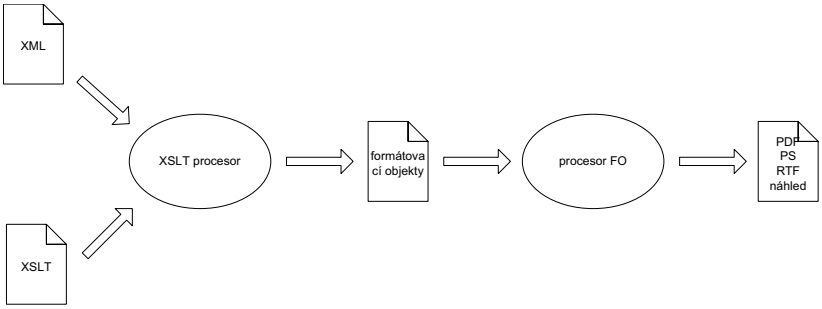
Formátovací objekty – druhá část XSL – jsou jen jakýsi abstraktní slovník, který umí velmi dobře definovat vizuální vzhled dokumentu. FO umožňují definovat layout stránek, a pak popsat formátování textu a dalších objektů, které se mají do těchto stránek naformátovat. Obsah stránek je přitom definován právě pomocí formátovacích objektů, které zastupují bloky textu, buňky tabulky nebo třeba obrázek. Každý objekt může mít několik desítek vlastností, které definují použité písmo, velikost okrajů okolo objektu, barvu textu i pozadí, vzhled rámečku, zakazují nebo povolují zlom stránky atd.

A jak formátovací objekty souvisí s formátováním XML? Celý trik spočívá v tom, že formátovací objekty se zapisují jako XML dokument, kde jednotlivým objektům odpovídají elementy a vlastnostem atributy. Díky tomu můžeme pro převedení XML do formátovacích objektů použít XSLT. Transformace převede sémantické značkování na formátovací objekty definující vzhled. Kromě toho se v tomto okamžiku může například vygenerovat obsah apod. Výsledný soubor FO se pak předá tzv. procesoru formátovacích objektů, který se postará o konečné zalomení formátovacích objektů do definovaných rozměrů stránky a výsledek zobrazí nebo uloží do nějakého vhodného formátu jako PDF nebo PostScript. V roli procesoru FO můžeme použít právě PassiveTEX.

Abychom si vše přiblížili na jednoduchém příkladě, předpokládejme, že máme následující XML dokument:

```
<odstavec>K tomuto účelu se používají
<pojem>stylové jazyky</pojem>,
```

[1] `http://www.xmlsoftware.com/xslt.html`



Obrázek 1: Princip zpracování XML pomocí XSL

```
které umožňují popsat, jak se mají jednotlivé elementy XML
dokumentu zobrazovat.</odstavec>
```

A my jej chceme pomocí XSL převést do tištěné podoby. Musíme si proto vytvořit XSLT styl, který bude generovat formátovací objekty. Dejme tomu, že chceme, aby se obsah elementu `odstavec` zobrazil jako samostatný odstavec, písmem o velikosti 12 bodů, zarovnaný do bloku, s odstavcovou zarážkou 18 bodů. Označené pojmy chceme zobrazit kurzívou. Do XSLT stylu proto přidáme dvě jednoduché šablony, které XML převedou na elementy odpovídající formátovacím objektům.

```
<xsl:template match="odstavec">
  <fo:block font-size="12pt" text-indent="18pt"
                             text-align="justify">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>

<xsl:template match="pojem">
  <fo:inline font-style="italic">
    <xsl:apply-templates/>
  </fo:inline>
</xsl:template>
```

Po zpracování dokumentu a stylu procesorem XSLT dostaneme dokument obsahující formátovací objekty:

```
<fo:block font-size="12pt" text-indent="18pt"
          text-align="justify">K tomuto účelu se používají
<fo:inline font-style="italic">stylové
jazyky</fo:inline>, které umožňují popsat, jak se mají jednotlivé
elementy XML dokumentu zobrazovat.</fo:block>
```

Vidíme, že během transformace se z XML dokumentu nesoucího nějakou sémantickou informaci stal jiný dokument, který již obsahuje jen informace o formátování. Procesor FO z něj dokáže vytvořit formátovaný dokument. Nejprve z něj načte elementy jako formátovací objekty, pak u všech formátovacích objektů doplní nespecifikované vlastnosti hodnotou zděděnou od rodičovského formátovacího objektu nebo implicitní hodnotou, vyhodnotí hodnoty výrazů apod. Výsledek se pak zalomí do oblastí vyhovujících všem omezením a práce je hotova.

Možnosti formátovacích objektů

Zatím jsme v ukázkách viděli jen dva formátovací objekty – `fo:block` a `fo:inline`, které patří mezi dva nejpoužívanější. První z nich umožňuje vytvářet samostatné bloky textu, které se zalamují do odstavce. Druhý pak umožňuje v části takového bloku změnit vybrané formátovací vlastnosti – např. změnit použitý řez písma.

Formátovacích objektů samozřejmě existuje mnohem více. Každý soubor FO na svém začátku nejdříve definuje předlohy stran – můžeme definovat rozměry stránky, velikost jejích okrajů apod. Těchto definic může být více a můžeme je sdružovat do předloh sekvencí stránek – tím lze snadno dosáhnout takových efektů, jako odlišný vzhled první stránky dokumentu či kapitoly nebo odlišnou velikost okrajů a záhlaví na sudých a lichých stránkách.

Za definicí předloh stránek pak následují jejich sekvence (`fo:page-sequence`). Pro každou sekvenci stránek můžeme určit, do jaké předlohy sekvence stránek se má sázet její obsah, jde definovat obsah pro záhlaví, zápatí, levý a pravý okraj. Ve formátovacím objektu `fo:flow` jsou pak obsaženy „běžné“ objekty jako právě `fo:block`, které již generují samotný obsah stránek.

Pro určité druhy objektů, které se mají ve výstupu objevit, nám již použití samotného `fo:block` nestačí. Pro tyto případy jsou k dispozici účelově zaměřené objekty.

K dispozici máme řadu objektů pro tvorbu tabulek. Model tabulek je podobný jako v HTML – tabulka (`fo:table`, `fo:table-and-caption`) se skládá z buněk (`fo:table-cell`), které jsou součástí řádky (`fo:table-row`) a řádka je součástí nějaké skupiny v tabulce (`fo:table-body`, `fo:table-footer`, `fo:table-header`). Pomocí vlastností pak můžeme nastavovat takové věci jako slučování buněk, rámečky okolo buněk a celé tabulky, barvy, pozadí, zarovnání obsahu buněk apod.

Samostatně existují i objekty pro seznamy. Celý seznam je uložen v `fo:list-block`. Každá položka seznamu je ohraničena pomocí `fo:list-item`. Položka seznamu pak má své návěstí (jako je odrážka, číslo, text) `fo:list-item-label` a tělo `fo:list-item-body`.

Obrázky se většinou vkládají jako odkaz na externí soubor pomocí

`fo:external-graphic`. Podporované formáty záleží na konkrétním procesoru FO. Pokud pro obrázky používáme nějaký formát založený na XML (např. SVG), můžeme je vložit přímo mezi formátovací objekty jako obsah elementu `fo:instream-foreign-object`. Při zpracování pak opět záleží na schopnostech konkrétního procesoru.

Celá řada objektů slouží ke vložení prvků, které se objeví na jiném místě, než jsou vloženy. Jedná se především o plovoucí objekty (`fo:float`), jež umožňují obtékání obrázků nebo jiných objektů textem zleva či zprava, případně ke vložení obrázku či tabulky na první vhodné místo v dokumentu. Do podobné kategorie spadají i poznámky pod čarou (`fo:footnote`, `fo:footnote-body`).

Existuje i několik objektů, které jsou vyhodnocovány a nahrazovány konkrétní hodnotou až v okamžiku formátování, protože ve fázi XSLT transformace nemáme dostatek informací. Typickým zástupcem je objekt pro vkládání aktuálního čísla strany `fo:page-number`. Při generování obsahu nebo křížových odkazů se naopak uplatní objekt `fo:page-number-citation`, který se nahradí číslem strany, na níž se vyskytuje formátovací objekt určený pomocí svého identifikátoru. Umožňuje-li to výstupní médium (např. PDF), můžeme z obsahu nebo křížových odkazů udělat jednoduše hypertextové odkazy pomocí `fo:basic-link`.

V mnoha dokumentech chceme mít v záhlaví/zápatí kromě čísla strany např. název kapitoly nebo podkapitoly. K dosažení tohoto efektu můžeme použít objekty `fo:marker` a `fo:retrieve-marker`. „Dynamicky“ se chová i objekt `fo:leader`, který vyplní volný prostor zadaným textem – lze jej použít například pro oddělení názvu kapitoly a čísla strany v obsahu tečkami.

Výčet objektů nebyl úplně kompletní, ale v praxi si s nimi bohatě vystačíme. V zasadě můžeme říci, že FO lze bez problémů použít pro formátování dokumentů obsahujících hladký text včetně obrázků, tabulek, obsahu, křížových odkazů, poznámek pod čarou, plovoucích záhlaví. Problémem není ani vícesloupcová sazba. Je to celkem pochopitelné, protože FO vycházejí z reálných požadavků na přípravu dokumentů.

Praktická ukázka

Ukažme si teď na jednoduchém příkladě kompletní funkční styl. Předpokládejme, že chceme definovat formátování pro články zapisované v XML. Každý článek má přitom následující strukturu:

```
<?xml version="1.0" encoding="iso-8859-2"?>
<!DOCTYPE clanek SYSTEM "clanek.dtd">
<clanek>
  <zahlavi>
    <rubrika>Aktuality</rubrika>
    <nazev>Státní správa bude používat TeX místo Wordu</nazev>
```

```
    <autor email="wild@duck.cz">Jan Novák</autor>
  </zahlavi>
  <perex>... text perexu ...</perex>
  <para>... text odstavce ...</para>
  <para>... text odstavce ...
        ... <em>zvýrazněný text</em> ...
        ... </para>
  <para>... text odstavce ...</para>
</clanek>
```

Chceme přitom nadpis článku vysázet větším písmem a vycentrovat, rubrika článku by se měla objevit v záhlaví a jméno a případný e-mail autora až na konci článku. Může nám k tomu posloužit následující styl:

```
<?xml version="1.0" encoding="iso-8859-2"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:fo="http://www.w3.org/1999/XSL/Format">

<xsl:template match="/">
  <fo:root>
    <!-- Definice layoutu stránky -->
    <fo:layout-master-set>
      <!-- Rozměry stránky a její okraje -->
      <fo:simple-page-master master-name="my-page"
                             page-height="297mm"
                             page-width="210mm"
                             margin="1in">
        <!-- Tiskové zrcadlo - oblast pro samotný obsah stránky -->
        <fo:region-body margin-bottom="15mm" margin-top="15mm"/>
        <!-- Oblast pro záhlaví stránky -->
        <fo:region-before extent="10mm"/>
        <!-- Oblast pro zápatí stránky -->
        <fo:region-after extent="10mm"/>
      </fo:simple-page-master>
    </fo:layout-master-set>

    <!-- Definice obsahu stránky -->
    <fo:page-sequence master-reference="my-page"
                      font-family="Helvetica, sans-serif"
                      font-size="12pt"
                      language="cs"
                      hyphenate="true">
      <!-- Společný obsah všech stránek v záhlaví stránky -->
      <fo:static-content flow-name="xsl-region-before">
```

```
        <fo:block text-align="right" border-after-style="solid"
                  border-after-width="0.4pt">
          Rubrika: <xsl:value-of select="clanek/zahlavi/rubrika"/>
        </fo:block>
      </fo:static-content>
      <!-- Společný obsah všech stránek v zápatí stránky -->
      <fo:static-content flow-name="xsl-region-after">
        <fo:block text-align="center" font-style="italic">
          <xsl:text>- </xsl:text>
          <fo:page-number/>
          <xsl:text> -</xsl:text>
        </fo:block>
      </fo:static-content>
      <!-- Samotný text dokumentu -->
      <fo:flow flow-name="xsl-region-body">
        <!-- Zpracování všech elementů zdrojového dokumentu -->
        <xsl:apply-templates/>
        <!-- Jméno autora zpracujeme až za textem článku -->
        <xsl:apply-templates select="clanek/zahlavi/autor"/>
      </fo:flow>
    </fo:page-sequence>
  </fo:root>
</xsl:template>

<!-- Šablona pro záhlaví článku -->
<xsl:template match="zahlavi">
  <fo:block text-align="center"
            space-after="14pt">
    <xsl:apply-templates select="nazev"/>
  </fo:block>
</xsl:template>

<!--Šablona pro název článku -->
<xsl:template match="nazev">
  <fo:block font-size="24pt" font-weight="bold"
            space-after="12pt" line-height="28pt">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>

<!--Šablona pro autora článku -->
<xsl:template match="autor">
  <fo:block font-style="italic" text-align="end" space-before="6pt">
    <xsl:text>&#x2014; </xsl:text>
```

```
    <xsl:apply-templates/>
    <xsl:text> &#x2014; </xsl:text>
  </fo:block>
  <!-- Pokud má autor e-mail, zpracujeme ho -->
  <xsl:if test="@email">
    <fo:block font-size="10pt" text-align="end">
      <xsl:text>e-mail: </xsl:text>
      <fo:inline font-style="italic">
        <xsl:value-of select="@email"/>
      </fo:inline>
    </fo:block>
  </xsl:if>
</xsl:template>

<!--Šablona pro perex -->
<xsl:template match="perex">
  <fo:block text-align="justify" font-style="italic" font-weight="bold">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>

<!--Šablona pro odstavec -->
<xsl:template match="para">
  <fo:block text-indent="18pt" text-align="justify">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>

<!--Šablona pro zvýraznění -->
<xsl:template match="em">
  <fo:inline font-style="italic">
    <xsl:apply-templates/>
  </fo:inline>
</xsl:template>

</xsl:stylesheet>
```

Při zpracování tohoto stylu se nejprve načte celý dokument XML do paměti v podobě stromu, kde jednotlivé uzly odpovídají elementům XML. Strom se pak prochází a pro každý uzel se hledá odpovídající šablona (`xsl:template`). Uvnitř šablon pak generujeme formátovací objekty, které definují zobrazení daného elementu. V případě, že pro právě zpracovávaný element chceme i jeho podelementy zpracovat dalšími šablonami, použijeme instrukci `xsl:apply-templates`. Pro vložení hodnoty získané ze vstupního dokumentu do výstupu slouží instrukce

`xsl:value-of`. Kromě toho lze uvnitř šablon používat další konstrukce jako podmínky a cykly.

Při formátování dokumentu musíme nejprve dokument XML pomocí stylu XSLT převést do formátovacích objektů. K tomu budeme potřebovat nějaký XSLT procesor, např. Saxon[2] nebo xsltproc[3]. Dokument s FO získáme pomocí příkazu:

```
saxon -o dokument.fo dokument.xml clanek.xsl
```

resp.

```
xsltproc -o dokument.fo clanek.xsl dokument.xml
```

Pro převedení souboru FO do PDF nebo jiného formátu musíme použít procesor FO, v našem případě PassiveTEX. Ten je dnes součástí většiny novějších distribucí TEXu. Nemáme-li jej, je možné si poslední verzi stáhnout vždy z adresy `http://www.tei-c.org.uk/Software/passivetex/`.

```
pdfxmltex dokument.fo
```

Na obrázku 2 vidíme výsledné PDF. Jak si ještě dále řekneme, není PassiveTEX úplná implementace FO. To se projevilo například tím, že záhlaví je špatně zarovnáno a není odděleno čarou od tiskového zrcadla. Zpracujeme-li stejný dokument procesorem FO, který implementuje standard FO úplněji, dostaneme lepší výsledek (viz obrázek 3).

Implementace PassiveTEXu

PassiveTEX je implementován jako makro pro TEX, které rovnou načítá dokument FO a sází jej. Pro načítání XML (připomeňme ještě jednou, že FO mají podobu dokumentu XML) se používá XML parser napsaný v TEXu – xmltex. Technicky je to uděláno tak, že globální katalog xmltexu obsahuje záznam mapující zpracování elementů ze jmenného prostoru FO na makra podle souboru `fotex.xmt`:

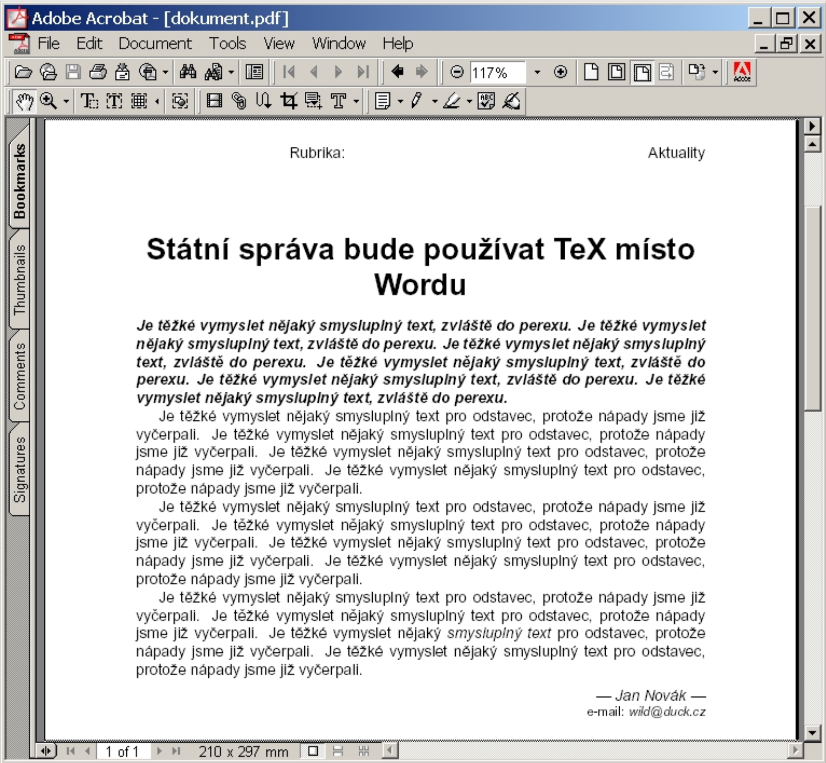
```
\NAMESPACE{http://www.w3.org/1999/XSL/Format}    {fotex.xmt}
```

Pro každý formátovací objekt je v souboru obsažena sekvence texových příkazů, kterými se má nahradit. Kvůli tomu PassiveTEX dnes nepodporuje FO úplně a asi ani nikdy nebude. Formátovací modely TEXu a FO jsou odlišné a jednoduché prostředky xmltexu neumožňují tyto rozdíly zcela překonat. V praxi dnes nejvíce vadí nedokonalé zpracování tabulek a téměř nulová podpora pro relativní délkové jednotky a výrazy uvnitř vlastností.

Používáme-li v dokumentech odkazy, nebo generujeme-li obsah, je potřeba PassiveTEX spustit opakovaně pro správné vygenerování čísel stran.

[2] `http://saxon.sourceforge.net`

[3] `http://xmlsoft.org/XSLT/xsltproc2.html`

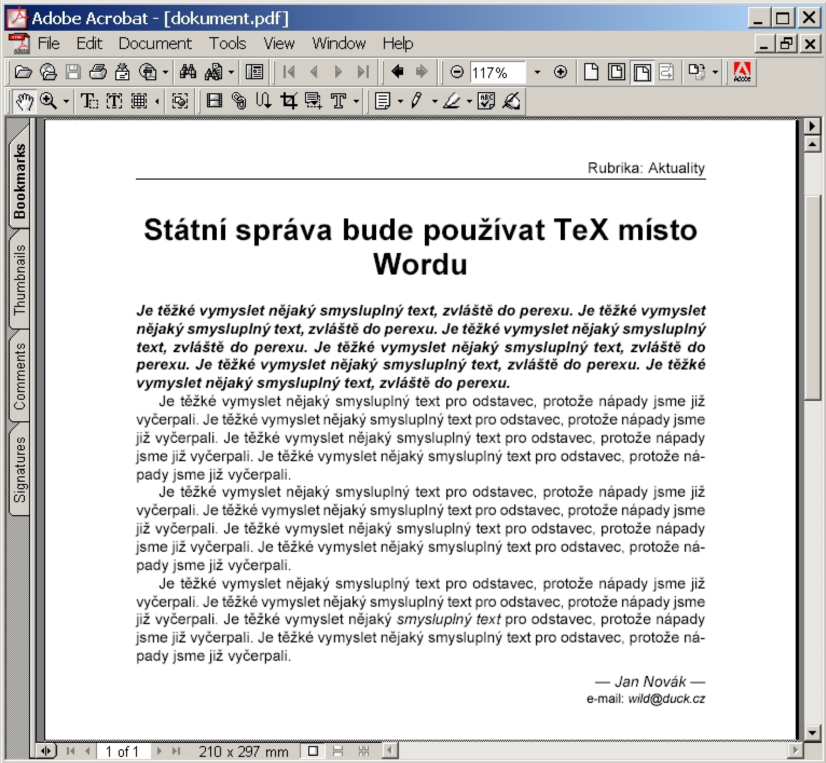


Obrázek 2: Dokument XML zformátovaný pomocí PassiveTEXu

Rozšíření oproti standardu FO

PassiveTEX obsahuje rozšíření pro generování záložek v PDF. Předpokládejme následující vstupní dokument:

```
<kniha>
  <kapitola>
    <název>...</název>
    ...
  </kapitola>
  <kapitola>
    <název>...</název>
    ...
  </kapitola>
```


Rubrika: Aktuality

Státní správa bude používat TeX místo Wordu

Je těžké vymyslet nějaký smysluplný text, zvláště do perexu. Je těžké vymyslet nějaký smysluplný text, zvláště do perexu. Je těžké vymyslet nějaký smysluplný text, zvláště do perexu. Je těžké vymyslet nějaký smysluplný text, zvláště do perexu. Je těžké vymyslet nějaký smysluplný text, zvláště do perexu. Je těžké vymyslet nějaký smysluplný text, zvláště do perexu.

Je těžké vymyslet nějaký smysluplný text pro odstavec, protože nápady jsme již vyčerpali. Je těžké vymyslet nějaký smysluplný text pro odstavec, protože nápady jsme již vyčerpali. Je těžké vymyslet nějaký smysluplný text pro odstavec, protože nápady jsme již vyčerpali. Je těžké vymyslet nějaký smysluplný text pro odstavec, protože nápady jsme již vyčerpali.

Je těžké vymyslet nějaký smysluplný text pro odstavec, protože nápady jsme již vyčerpali. Je těžké vymyslet nějaký smysluplný text pro odstavec, protože nápady jsme již vyčerpali. Je těžké vymyslet nějaký smysluplný text pro odstavec, protože nápady jsme již vyčerpali. Je těžké vymyslet nějaký smysluplný text pro odstavec, protože nápady jsme již vyčerpali.

Je těžké vymyslet nějaký smysluplný text pro odstavec, protože nápady jsme již vyčerpali. Je těžké vymyslet nějaký smysluplný text pro odstavec, protože nápady jsme již vyčerpali. Je těžké vymyslet nějaký *smysluplný text* pro odstavec, protože nápady jsme již vyčerpali. Je těžké vymyslet nějaký smysluplný text pro odstavec, protože nápady jsme již vyčerpali.

— *Jan Novák* —
e-mail: *wild@duck.cz*

Obrázek 3: XEP zvládá mnohem větší část FO a správně formátuje i záhlaví a zápatí stránky

```
  ...
</kniha>
```

Chceme-li mít názvy kapitol v záložkách, musíme každé kapitole přiřadit identifikátor a vložit do výsledného souboru FO rozšiřující element `bookmark`. Ten je v jiném jmenném prostoru, aby nekolidoval se standardními FO.

```
<xsl:template match="kapitola">
  <fo:block id="{generate-id(.)}">
    <fotex:bookmark xmlns:fotex="http://www.tug.org/fotex"
                    fotex-bookmark-level="1"
                    fotex-bookmark-label="{generate-id(.)}">
      <xsl:value-of select="název"/>
    </fotex:bookmark>
```

```
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

Sazba matematiky

PassiveTEX umožňuje do souboru s FO vkládat přímo matematické výrazy zapsané v MathML. Využívá se přitom toho, že xmltex standardně obsahuje podporu pro sazbu MathML. Do dokumentů XML tedy můžeme vkládat vzorce v MathML a v XSLT stylu je beze změny nakopírovat mezi formátovací objekty.

MathML má však jednu velkou nevýhodu a to je jeho ruční zápis. Formát je to poměrně upovídaný a vzorec, který lze v TEXu napsat na půlce řádky zabere v MathML spíše půl obrazovky. Rozumně se dá s MathML pracovat snad jen pomocí vizuálního editoru rovnic nebo pomocí nástrojů, které umějí provést transkripci z nějakého úspornějšího zápisu. PassiveTEX nám proto nabízí jednoduchou možnost, jak dokumenty zapisovat v XML se všemi s tím spojenými výhodami, ale matematické vzorce zapisovat dál v TEXu, jak jsme zvyklí.

Dosáhnout toho můžeme díky tomu, že v dokumentu s formátovacími objekty je rozeznávána speciální instrukce pro zpracování, která může obsahovat libovolný texový kód. Např.:

```
<?xmltex $a^2+b^2=c^2$?>
```

V dokumentu XML si pak stačí vzorce označit nějakým elementem:

```
<vzorec>a^2+b^2=c^2</vzorec>
```

A do stylu přidat šablonu, která z tohoto elementu vytvoří instrukci pro zpracování:

```
<xsl:template match="vzorec">
  <xsl:processing-instruction name="xmltex">
    <xsl:value-of select="."/>
  </xsl:processing-instruction>
</xsl:template>
```

Závěr

PassiveTEX zdaleka neimplementuje celý standard FO. Na druhou stranu jeho použití nám přináší kvalitní typografické algoritmy a podporu sazby matematiky. Dáme-li si pozor a ve stylech použijeme jen ty formátovací objekty a jejich vlastnosti, které PassiveTEX podporuje, můžeme z XML generovat velmi kvalitní tištěný výstup. PassiveTEX tak ukazuje jednu z cest, jak si může TEX zachovat své uživatele i v době, kdy je stále více vstupních dat a dokumentů dostupných právě v podobě XML.

Odkazy

[1] Sharon Adler – Anders Berglund – Jeff Caruso – Stephen Deach – Paul Grosso – Eduardo Gutentag – Alex Milowski – Scott Pernell – Jeremy Richman – Steve Zilles: *Extensible Stylesheet Language (XSL) – Version 1.0.* W3C, 2001. URL: `http://www.w3.org/TR/xsl`

[2] James Clark: *XSL Transformations (XSLT) Version 1.0.* W3C, 1999. URL: `http://www.w3.org/TR/xslt`

[3] Michel Goossens – Sebastian Rahtz: *PassiveTEX: from XML to PDF.* In: *TUGboat.* 3/2000. URL: `http://www.tug.org/TUGboat/Articles/tb21-3/tb68goos.pdf`

Summary: PassiveTEX

PassiveTEX is a TEX-based XSL-FO processor which is able to process XML documents according to an XSL stylesheet in conjuction with any XSLT processor. This article briefly discribes basic principles of the XSL language and its usage for formatting XML documents. Complete working example of an XSL stylesheet is shown in the article.

Jiří Kosek
`jirka@kosek.cz`

Ako TEXujeme

Ladislav Bittó

Vážení TEX-isti,

na základe našej ankety som Vám poslal jej výsledok. Dúfam, že nikoho som nevynechal a teda každý z Vás objaví svoj príspevok, ktorý je medzi dvoma horizontálnymi čiarami. Chcel som to naprogramovať tak, že sa Vášho mailu nedotknem (iba ich ukladať za sebou ako prichádzajú). Bohužiaľ nešlo to tak, lebo niektorí z Vás písali dlhšie komentáre, prehodili poradie, zmenili ste text otázky, niektorí poslali späť aj môj príklad. Aj ja som narobil zmätok tým, že v druhej výzve som vynechal položku prezerač. Na upozornenie som to hneď dal na vedomie, ale už sa to vlieklo. Podľa mňa, najväčší problém vznikol z toho dôvodu, že anketa sa rozbehla napriek tomu, že nebola ani vyhlásená. Ja som

len nadhodil tú tému a čakal reakcie na ňu. Nikto sa neozval (iba pán Kloc - aj ten už neskoro), ale hneď ste ma zasypali vyplnenými mailmy, ako keby ste len na toto čakali. Takže trošku som musel mazať, upravovať, niektoré dlhšie názvy skrátiť (chcel som, aby jeden názov sa zmestil na jeden riadok), aby to ten malý progmámek vedel spracovať k spokojnosti (aspoň mojej). Skrátka mal som o zábavu postarané.

Keď vidím tú tabuľku pred sebou čierne na bielom, tak už mám veľmi dobrú predstavu, že ako vlastne TEX-ujeme. U mňa to je ešte umocnené s tým, že viem presne, kto čo poslal. Mne nešlo o počty (koľko ludí linux-uje, koľko ľudí používa plain, atď.), ale hlavne o to, že ako TEX-uje jeden človek. Ten jeden človek čo a koľko toho používa. O tom je tá tabuľka. Nakoniec do ankety prispelo 55 ľudí, čo je asi dostatočne reprezentačné. Ja osobne som chcel presne takúto tabuľku. Vidieť z nej dobre, že TEX je živý až moc (možno že som zle pozeral, ale nenašiel som dve úplne rovnaké odpovede). Ak niekto je z nej sklamaný a potrebuje niečo iné, údaje sú k dispozícii. Na základe tejto tabuľky by sa dala vypracovať ďalšia anketa (o rok, o dva, o 10, nikdy). Každý by mal na každú kolonku 100 bodov (keď tam dá sedem rôznych vecí, tak ich musí rozdeliť, aby súčet bol vždy 100). Dalo by sa potom z tohto guláša vyrobiť obrázok, ktorý by hneď ukázal viac, než 10 strán textu.

Nakoniec ďakujem všetkým, ktorí ste poslali vyplnenú odpoveď. Bez Vašej pomoci by nevznikla táto tabuľka. Dúfam, že som Vám tým nenarobil veľké problémy. Ak uznáte za vhodné, môžme ju o 10 rokov opakovať, alebo aj skôr :-)

Čo znamenajú jednotlivé položky:

systém:	operačný systém na ktorom beží Váš počítač
TEX:	inštalácia TEX-u
formát:	použitý TEX-ovský formát
editor:	náš obľúbený editor, v ktorom píšeme svoje TEX-ty
prezerač:	náš obľúbený prezerač, v ktorom pozeráme svoje TEX-ty
prog. grafika:	programovateľná grafika, v ktorom programujeme svoje grafy, obrázky
ostat. grafika:	ostatná grafika, kde kreslíme (opravujeme/prekódujeme) svoje grafy, obrázky
graf. výst.:	grafický výstup, ktorý vkladáme do TEX-u

	systém	TEX	formát	editor	prezerač	prog. grafika	ostat. grafika	graf. výst.
1	W2k	TeXLive6	cslatex pdfcslatex	TexShell 0.63	Yap 0.98n		CorelDRAW	PDF
2	RedHat7.3 Win98	tetex	pdfcslatex cslatex	nedit EditPad	acroread xdvi	-	Corel Draw	pdf
3	OS/2	emTeX	cslatex	EPM	dvipm gsview Acrobat Reader	gnuplot PStricks	gimp	PS PDF
4	OS/2 Linux Windows	emtex tetex MikTeX	latex cslatex	EPM gvim gvim	dvipm; gsview xdvi; kdvi; gs yap; gsview	gnuplot	xfig	PS
5	WIN2k	MikTeX pdfTeX	csplain	WinEdt	GV		scan	PS PDF
6	Linux RedHat	tetex	cslatex revtex	vim	xdvi gv	gnuplot grace		PS
7	eCS (OS/2) Linux	emTeX teTeX	csplain	EPM mc kwrite	GSview dvipm GV xdvi	gnuplot Metapost	Embellish Qcad gimp	PS
8	Win98	MikTeX	cslatex pdfcslatex	WinEdt UltraEdit	Yap GsView Acrobat Reader	Metapost		PS PDF
9	linux Solaris HP-UX	tetex web2c	csplain pdfcsplain pdfcslatex	Vlm vi	gv ggv xdvi acroread	Metapost gnuplot	GIMP	PS PDF
10	Linux	TeXlive7	csplain pdfcsplain	kwrite vim	kdvi	GIMP	Metagraf	PS PDF
11	Linux RedHat	tetex-beta	csplain pdfcsplain	Emacs	Ghostview	gnuplot xfig	gimp	ps pdf
12	Linux	Web2C 7.3.1	cslatex	vim	gv	pstrick		PS

	systém	TEX	formát	editor	prezerač	prog. grafika	ostat. grafika	graf. výst.
13	Linux (vyhradne)	web2c teTeX TeXlive	csplain	Emacs vi	xdvi	xfig	gimp	ps pdf
14	Linux SuSE 8 Windows 2000 XP	teTeX TeXlive	latex cslatex pdflatex pdfcslatex	Emacs AUCTeX CDLATEX WinShell	xdvi gv/gs windvi gsview/gs	Metapost(V) gnuplot(V) Mathematica(V)	Gimp(B)	PS PDF
15	WIN98 Linux RH 6.2	LaTeX	pdfcslatex cslatex	xwindows	acrobat GV dviscr?	gnuplot	tgif AI	ps pdf
16	W98(DOS okno) Linux	emTeX	csplain	latman joe	dviscr GV	PStricks PicTeX	CorelDraw(V) CorelPhP(BM) Origin(V)	PS
17	Linux (debian)	tetex	csplain	vim	dvipdfm	dvipdfm		pdf
18	Linux (RH 6.2)	tetex (1.0)	pdfcsplain	vi	dvisvga gv		jpeg	pdf
19	Linux	teTeX	csplain cslatex latex plain	vim	xdvi gv acroread	Metapost Gnuplot	gimp	PS PDF
20	Linux	tetex	csplain eplain	vi	xdvi gv	Metapost	gimp	ps
21	Win2000 RedHat Linux	TeXLive	csplain pdfcsplain	gvim vim	Acrobat Reader GV	Metapost		PS PDF
22	Linux Win2000 Win98 Solaris	teTeX TeX Live 7	csplain plain	Vi Emacs	GV	Metapost	Gimp PSP	PS
23	(SGI)Irix Linux	pdftex (TeXLive teTeX)	cslatex	gvim Kile	Xpdf GSView Acroread	Metapost Gnuplot Xfig	Corel PSP	PDF

	systém	TEX	formát	editor	prezerač	prog. grafika	ostat. grafika	graf. výst.
24	Windows XP	MikTeX	cslatex	WinEdt	GhostView yap Acrobat		PaintShop AdobePhotoS.	PS PDF
25	Linux 2.4	tetex	latex	vim	xdvi gv	S-plus	xfig imagemagick sdraw	eps
26	win98 w2k	MikTeX	cslatex	WinEdt Vim	Yap	metapost	Photoshop CorelDraw	PS PDF
27	linux (Debian GNU/Linux)	tetex	cslatex pdfcslatex	vim	gv acroread xpdf	gnuplot	xfig sketch sodipodi gimp	pdf png ps
28	Linux	teTeX	pdfcslatex	vim	gv xdvi	gnuplot metapost	ziadne	EPS
29	linux Win98 Win2000 DOS	emtex tetex MiKTeX fpTeX	csplain pdfplain	Winedt vim CSed	Yap Acrobat Reader xdvi xpdf dviscr	metapost Metafont Mathematica	Corel Draw mspaint IrfanView	PS PDF PNG
30	linux (Debian SuSE)	teTeX	csplain cslatex	vim	gv	PSTricks vim	ruznoroda +a2ps pstops	PS
31	linux	tetex	[pdf][cs]latex	joe vim	xdvi ghostview	gnuplot	xfig	eps
32	linux	TeXlive	csplain pdfcsplain latex	emacs mcedit lyx	tkdvi xdvi gv xpdf acroread	-	gimp xfig	ps pdf

	systém	TEX	formát	editor	prezerač	prog. grafika	ostat. grafika	graf. výst.
33	linux	tetex	latex cslatex pdfcslatex	vi	ghostview kghostview Acrobat Reader xpdf	gnuplot	Tgif	ps pdf
34	linux Redhat 7.3	TeXlive tetex emtex	latex cslatex pdfcslatex	emacs vim	xdvi ghostview acroread	mfpic metapost	xfig	eps
35	w2k	TeXlive	cslatex pdfcslatex	winedt	GSview Adobe Acrobat	metapost	imagemagick PSP 7.04	PS PDF
36	Linux	Web2C (encTeX) TeXLive7	csplain plain cslatex pdfcslatex	Vim	gv xdvi	Gnuplot Perl Python C Metafont	XFig Gimp Sodipodi	ps pdf
37	linux	tetex	latex	vim	xdvi gv	root gnuplot		ps
38	linux	web2c	cslatex latex	vim	xdvi	metapost	ImageMagick	PS
39	linux	TeXLive	plain (pdf)(cs)plain (pdf)(cs)latex Context	emacs	gv acroread	metapost Matlab xypic	gimp ImageMagick (perl) photoshop	pdf ps
40	WinXP Linux	MikTeX TeXlive	cslatex csplain	WinEdt Emacs	Yap xdvi	Metapost gnuplot	Origin Photoshop	ps pdf
41	win95	emtex(DOS)	amstex	latman	dviscr ?	gnuplot		ps
42	linux WinNT	tetex	latex cslatex	emacs	xdvi gv	gnuplot	xfig Corell Draw	ps
43	Linux	tetex	csplain plain (musixtex)	vim	xdvi kdvi gv	-	gimp	ps

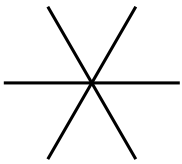
	systém	TEX	formát	editor	prezerač	prog. grafika	ostat. grafika	graf. výst.
44	linux	tetex	latex pdflatex	LyX vim	xdvi kdvi kghostview xpdf	metapost gnuplot		PDF
45	linux	tetex TeXLive	csplain	vim	ghostview		xfig	ps
46	win98	TeXLive	csplain pdfcsplain	Ultraedit	windvi	metafont xypic	-	dvi pdf
47	Win2000	Miktex 2.0	cslatex csplain (pdftex)	emacs	yap Acrobar Reader		gimp	PDF
48	Linux Solaris windows	tetex MikTeX	cslatex	emacs	xdvi yap	xfig matlab visio		eps
49	Linux RH7 DOS6.22	tetex emtex	pdfcslatex cslatex	gvim vim csedit Dos Navigat.	Acroread Xpdf gv dviscr	gnuplot	xfig xpaint qcad xtexcad neopaint texcad	PDF PNG PS
50	W2k	fpTeX	[pdf][cs]latex	UltraEdit	windvi GV Acrobat reader	tex-pict (epis eepic)	Corel draw Photoshop	PS PDF
51	Solaris Linux	tetex	latex pdflatex	emacs	xdvi gv	TEX	xfig	ps pdf
52	OS/2	emtex	cslatex	epm	dvipm	metapost	Maple	ps
53	W98(dos okno)	emtex	cslatex	latman	GV	FORTRAN	PaintShop	ps
54	Linux WIndows 2000	Tetex TeXLive6	cslatex	gVIM	DVI GS		GIMP	PS PDF

	systém	TEX	formát	editor	prezerač	prog. grafika	ostat. grafika	graf. výst.
55	linux SunOS IRIX Win98	TeXlive	pdfcsplain csplain	Vim	xpdf gv acroread	METAPOST	GIMP	PS PDF

Fraktální obrazce v PostScriptu

Zdeněk Wagner

Fraktály jsou velmi důležité objekty, které nacházejí využití v mnoha vědních oborech. Mají praktický význam i v oblastech, kde bychom to možná ani netušili. Uplatní se při studiu členitosti mořského pobřeží, při popisu povrchu katalyzátorů i při zkoumání podmínek proudění tekutin v potrubí. Je zřejmé, že rozsah jejich použití je značný, a pochopitelně se podrobnému popisu zde věnovat nebudeme. Pro nás budou fraktály jen zajímavým geometrickým útvarem, který lze graficky využít. Předvedeme si, jak lze jednoduché fraktály generovat přímo v PostScriptu a jak byl vytvořen obrázek na obálce tohoto čísla Zpravodaje.



Velmi zjednodušeně lze říci, že fraktál je struktura, která sama sebe pravidelně opakuje ve zmenšené velikosti. Zvětšíme-li tedy libovolnou část fraktálu, najdeme v ní stejný geometrický motiv. Programově lze tedy takový objekt generovat pomocí rekurzivních funkcí. Podle definice se základní fraktální struktura musí opakovat donekonečna. Jenže na vykreslení by počítač musel disponovat nekonečnou velikostí paměti a vykreslování by trvalo nekonečně dlouho. To však není praktické a matematicky dokonalé fraktály nemusí vždy vypadat hezky. Proto ukončujeme vykreslování v určité, poměrně malé hloubce vnoření. Jako první příklad si uvedeme sněhovou vločku, jejíž základní motiv obsahuje šest paprsků vycházejících ze společného středu.

Než se pustíme do vysvětlování, musíme trošku odbočit. Článek obsahuje několik fraktálních obrázků, které sdílejí společná makra. Navíc k ladění a drobným úpravám maker dochází i při psaní článku. Proto si uložíme všechna makra do souboru `fraktaly.ps` a na začátku LATEXového souboru použijeme příkaz

`\special{header=fraktaly.ps}`. Na základě tohoto povelu DVIPS vloží makra do PostScriptového dokumentu a fraktální obrázky je mohou využívat. Struktura souboru maker je následující:

```
%!PS
20 dict /Fraktaly exch def
Fraktaly begin
definice maker
end
```

Vidíte, že jsme si založili vlastní slovník a definice budeme vkládat do něj.

Nyní se můžeme vrátit ke sněhové vločce. Jak jsme již uvedli, budeme na základní motiv pohlížet jako na šest paprsků vycházejících ze společného středu. Paprsek tedy vytvoříme jako úsečku jednotkové délky, na níž budeme aplikovat transformace: posun, zvětšení a otočení. Vykreslíme jej tímto makrem:

```
% <w> <rot> <x> <y> beam
/beam {
  .2 4 index sqrt div setlinewidth
  gsave
    translate rotate dup scale
    newpath 0 0 moveto 1 0 lineto stroke
  grestore
} bind def
```

Parametry makra jsou postupně šířka, definující zvětšení, úhel otočení a souřadnice středu. Všimněte si, že tloušťku čáry neměníme ve stejném poměru.

Paprsek nyní využijeme jako základní kámen k vytvoření hvězdy. Sestavíme z něj rekurzivní makro:

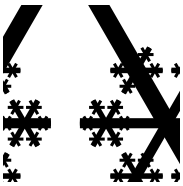
```
% <depth> <w> <x> <y> star
/star {
  0 60 333 {
    4 index 0 gt {
      4 index 1 sub        % depth w x y r depth'
      4 index .3 mul       % depth w x y r depth' w'
      5 index .7 mul dup   % depth w x y r depth' w' 2w' 2w'
      4 index cos mul      % depth w x y r depth' w' 2w' dx
      6 index add exch     % depth w x y r depth' w' x' 2w'
      4 index sin mul      % depth w x y r depth' w' x' dy
      5 index add          % depth w x y r depth' w' x' y'
      star
    } if
    3 index exch  % w x y w r
```

```
    3 index 3 index beam
  } for
  4 {pop} repeat
} bind def
```

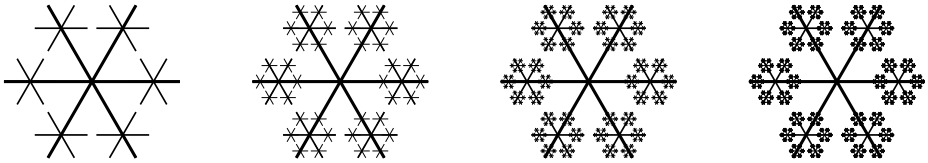
Parametry tohoto makra jsou vyžadovaná hloubka rekurze, šířka (vlastně délka paprsku) a souřadnice středu. V makru snadno najdeme cyklus s krokem 60°. Neděste se horní hranicí 333. Sice při celočíselné aritmetice k zaokrouhlovacím chybám nedochází, ale používání horní meze nad skutečnou požadovanou hranicí je dobrým zvykem. Požadujeme-li nulovou hloubku rekurze, vniřní podmíněný příkaz se nevykoná. Na vrcholu zásobníku máme tedy úhel, který tam vložil operátor `for`. Abychom připravili parametry pro makro `beam`, musíme ze zásobníku zkopírovat šířku a prohodit ji s úhlem. Dále musíme zkopírovat souřadnice středu. Makro `beam` tyto hodnoty ze zásobníku odstraní, takže při dalším průchodu cyklem je zásobník ve stejném stavu. Po skončení cyklu odstraníme parametry operátorem `pop`. Takto jsme vytvořili již dříve uvedený základní motiv.

Při nenulové hloubce rekurze se dostane ke slovu podmíněný příkaz. Na každý paprsek do vzdálenosti 70 % délky nakeslíme další hvězdu zmenšenou na 30 % původní velikosti. Nejprve zkopírujeme hloubku rekurze a zmenšíme ji o jednotku. Dále vypočteme novou šířku a nové souřadnice středu. Tyto hodnoty pak předáme makru `star`. Výsledek pro zvyšující se hloubku rekurze vidíte na obrázku 1. Obrázky byly získány jednoduchým způsobem. Soubor maker nám při zpracování LaTeXového dokumentu vložil DVIPS, takže při hloubce 4 stačí psát:

```
%!PS
%%Pages: 1
%%BoundingBox: 0 0 66 66
%%Page: 1 1
Fraktaly begin
4 33 33 33 star
end
showpage
%%EOF
```

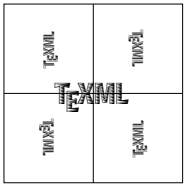


Podívejme se na jedenáctkrát zvětšený výsek spodní části paprsku směřujícího doprava. Získali jsme jej tak, že jsme v příkazu `\includegraphics` nastavili nepovinné parametry na hodnoty `[clip,viewport=54 25 60 31,width=66bp]`. Ač se při použití hloubky rekurze 4 z obrázku 1 zdá, že se malé vločky slijí do jednolité černé plochy, při zvětšení je zřejmé, že tomu tak ve skutečnosti není. Stejný motiv se opakuje donekonečna. I kdybychom použili nekonečnou hloubku rekurze, abychom získali skutečný fraktál, zůstaly by všechny



Obrázek 1: Sněhové vločky s hloubkou rekurze 1–4

vločky „děravé“ a stále by opakovaly svoji strukturu. Tak je předvedena základní vlastnost fraktálů v praxi.



Po této rozcvičce se můžeme pustit do programování textového fraktálu použitého na obálce. Základní strukturou je nápis TEXML. Byl vybrán proto, že hlavní náplní tohoto čísla jsou články o sazbě XML souborů v TEXu. Vnitřní písmeno X bylo vynecháno úmyslně. Text je sázen písmem LexonGothic Xylo ze Střešovické písmolijny. Na obrázku je vidět příklad s hloubkou rekurze 1. Při nulové hloubce bychom ve středu čtverce viděli jen základní text. Každý další rekurzivní krok rozdělí čtverec na čtyři menší a do jejich středu vloží stejný text otočený o 90° ve směru nebo proti směru hodinových ručiček.

Makra jsou v tomto případě složitější. Nejprve nadefinujeme makro pro výběr písma, kde jako jediný parametr zadáme požadovanou velikost:

```
% <size> LX
/LX {/LexonXylo exch selectfont} bind def
```

Samostatně by však takové makro nefungovalo, protože tento font není v PostScriptovém RIPu přítomen. Fontový soubor jsme tedy programem T1ASCII převedli do formátu PFA. Přidáme jej k dokumentu podobně jako fraktální makra, tedy příkazem `\special{header=lexonxyl.pfa}`.

V makru se využívá operátor `selectfont` z PostScriptu Level 2. Máme-li pouze zařízení Level 1, musíme před definici makra `LX` přidat:

```
% Emulation of selectfont by Level 1 operators
/selectfont {exch findfont exch scalefont setfont} bind def
```

Při sazbě textu TEXML budeme vodorovně i svisle posunovat písmeno E tak, jak je to v logu TEX zvykem. Hodnota posunu byla určena empiricky tak, aby to vypadalo hezky. Zde je makro, které jako jediný parametr přijímá velikost písma. Předpokládá se, že aktuální bod sazby již byl nastaven.

```
% <size> texml
/texml {
```

```
  dup LX                    % size
  (T) show                  % size
  neg dup 25 div exch       % -size/25 -size
  dup 4 div dup neg exch    % -size/25 -size size/4 -size/4
  3 -1 roll 10 div exch rmoveto
  (E) show rmoveto (XML) show
} bind def
```

Při dalším zpracování budeme potřebovat šířku a výšku textu, abychom jej mohli umístit na střed. Změříme tedy ohraničovací rámeček textu TEXML. Započítáme pouze korekci na horizontální posuny:

```
% <size> sz <width> <height>
/sz {
  dup LX % size
  gsave newpath 0 0 moveto
  (TEXML) true charpath flattenpath pathbbox % size llx lly urx ury
  3 1 roll sub                               % size llx urx height
  exch 3 1 roll sub                          % size height width
  2 index 25 div sub 3 1 roll 10 div sub grestore
} bind def
```

Nyní můžeme text vysadit na požadované místo. Zadáme úhel otočení, souřadnice středu čtverce a velikost písma. Všimněte si, že svislá velikost nebyla předchozím výrazem určena správně, přestože byl použit operátor `flattenpath`. Proto je do následujícího makra vložena empirická korekce. Text otáčíme vždy o násobek 90°, proto místo skutečného úhlu používáme jen malá celá čísla.

```
% <rot> <x> <y> <size> xput
/xput {
  dup sz                            % rot x y sz width height
  2.5 div neg exch 2 div neg exch   % rot x y sz -h/2.5 -w/2
  6 3 roll                          % sz -w/2 -h/2 rot x y
  gsave translate 90 mul rotate moveto texml grestore
} bind def
```

Nyní můžeme všechny stavební kameny složit do výsledného makra. Je opět rekurzivní, proto si jej nejprve ukážeme a dodatečně vysvětlíme.

```
% <w> <r> <x> <y> <sz> <depth> rxput
% w means half width of the square, x and y is the square's centre
/rxput {
  dup 0 gt {
    5 index 2 div               % w r x y sz depth w/2
    5 index 1 exch sub          % w r x y sz depth w/2 r'
    5 index 2 index sub         % w r x y sz depth w/2 r' x'
```

```
    5 index 3 index add          % w r x y sz depth w/2 r' x' y'
    5 index 2 div 5 index 1 sub  % w r x y sz depth w/2 r' x' y' sz' depth'
    rxput
    5 index 2 div                % w r x y sz depth w/2
    5 index -1 exch sub          % w r x y sz depth w/2 r'
    5 index 2 index add          % w r x y sz depth w/2 r' x'
    5 index 3 index add          % w r x y sz depth w/2 r' x' y'
    5 index 2 div 5 index 1 sub  % w r x y sz depth w/2 r' x' y' sz' depth'
    rxput
    5 index 2 div                % w r x y sz depth w/2
    5 index -1 exch sub          % w r x y sz depth w/2 r'
    5 index 2 index sub          % w r x y sz depth w/2 r' x'
    5 index 3 index sub          % w r x y sz depth w/2 r' x' y'
    5 index 2 div 5 index 1 sub  % w r x y sz depth w/2 r' x' y' sz' depth'
    rxput
    5 index 2 div                % w r x y sz depth w/2
    5 index 1 exch sub           % w r x y sz depth w/2 r'
    5 index 2 index add          % w r x y sz depth w/2 r' x'
    5 index 3 index sub          % w r x y sz depth w/2 r' x' y'
    5 index 2 div 5 index 1 sub  % w r x y sz depth w/2 r' x' y' sz' depth'
    rxput
  } if
  pop xput pop
} bind def
```

Při nulové hloubce rekurze se podmíněný příkaz neprovede. Odstraníme-li nyní hloubku rekurze, máme na vrcholu zásobníku přesně ty parametry, které vyžaduje makro `xput`. Po jeho provedení zbude v zásobníku nepoužitá poloviční šířka čtverce, kterou musíme odstranit. Výsledkem je tedy nápis TEXML uprostřed základního čtverce. Rekurzivní volání se provádí uvnitř podmíněného příkazu. Nejprve zmenšíme šířku čtverce na polovinu. Aby správně fungovala rotace textu, musíme od požadovaného úhlu odečíst rotaci čtverce, z něhož rekurzi voláme. Pak vypočteme souřadnice středu nového čtverce, zadáme poloviční velikost písma a nakonec vložíme hloubku rekurze zmenšenou o jednotku. Postup opakujeme pro všechny menší čtverce.

V příkladech si chceme zobrazit rámeček a v ukázce základního motivu i rozdělení na menší čtverce. Dosáhneme toho jednoduchými makry:

```
% <w> <r> <x> <y> <sz> <depth> frxput
% framed rxput
/frxput {
  gsave
    3 index 3 index translate
    5 index dup scale
    5 index 4000 div setlinewidth 0 setgray
    newpath
    -1 -1 moveto -1 1 lineto 1 1 lineto 1 -1
```

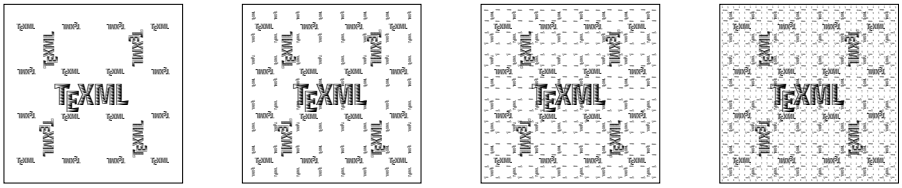
```
    lineto closepath stroke
  grestore
  rxput
} bind def

% <w> <r> <x> <y> <sz> <depth> Frxput
% framed rxput with middle cross
/Frxput {
  gsave
    3 index 3 index translate
    5 index dup scale
    5 index 6000 div setlinewidth 0 setgray
    newpath
    0 -1 moveto 0 1 lineto 1 0 moveto -1 0 lineto
    closepath stroke
  grestore
  frxput
} bind def
```

Nyní již můžeme vykreslit fraktály. Opět jsou vlastní soubory jednoduché, neboť jen otevřeme odpovídající slovník a zavoláme příslušné makro. Základní motiv byl vytvořen makry:

```
%!PS
%%Pages: 1
%%BoundingBox: 0 0 77 77
%%Page: 1 1
Fraktaly begin
33.5 0 33.5 33.5 11 1 Frxput
end
showpage
%%EOF
```

Obrázky s vyšší hloubkou rekurze se liší jen tím, že netiskneme vnitřní dělení čtverce:

```
%!PS
%%Pages: 1
%%BoundingBox: 0 0 77 77
%%Page: 1 1
Fraktaly begin
33.5 0 33.5 33.5 11 5 frxput
end
```



Obrázek 2: Fraktální text s hloubkou rekurze 2–5

```
showpage
%%EOF
```

Hotové fraktály s hloubkou rekurze 2–5 jsou na obrázku 2. Vidíme, že motiv je čtvercový, ale prostor na obálce, vyhrazený pro obrázek, je obdélníkový. Proto jsme pod větší fraktální motiv přidali dva poloviční s hloubkou rekurze o jednotku nižší a otočený nohama vzhůru. Soubor pro použití na obálce pak vypadá takto:

```
%!PS
%%Pages: 1
%%BoundingBox: 0 0 168 252
%%Page: 1 1
Fraktaly begin
84 0 84 168 24 6 rxput
42 2 42  42 12 5 rxput
42 2 126 42 12 5 rxput
end
showpage
%%EOF
```

Na závěr jsme ještě přidali vertikální linky, aby prostor byl využit vizuálně symetricky. Obálka je tedy vytvořena makry (viz styl pro Zpravodaj na stránce `http://bulletin.cstug.cz`):

```
\def\xkr{\kern2.5dd}
\def\xvr{\vrule height 252bp depth 0dd width .3dd}
\def\texml{\includegraphics{texml.eps}}
\Obalka[\Centerbox{\xvr\xkr\texml\xkr\xvr}]
```

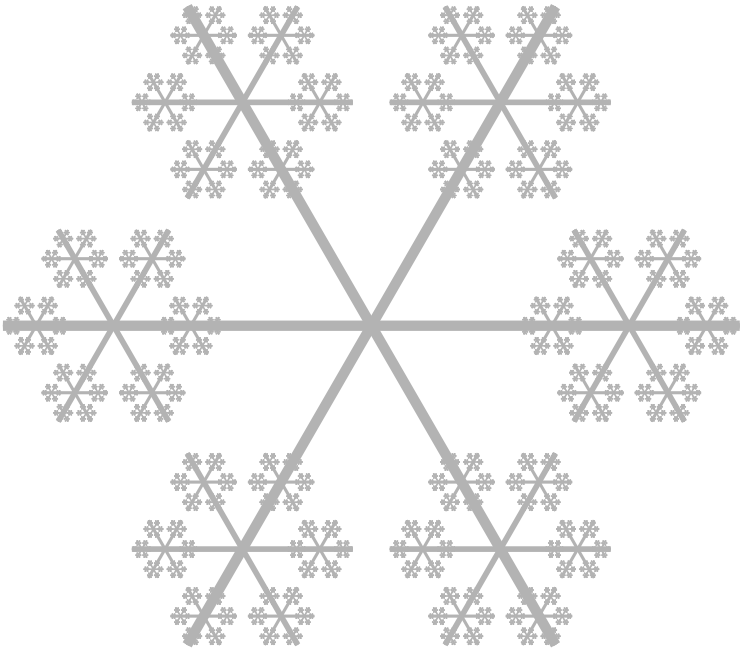
Nakonec ještě poznamenáme, že text souborů byl do tohoto článku vkládán příkazem `\verbatiminput`. Tím je zajištěno, že ukázky obsahují přesně stejný kód, který byl při kreslení obrázků použit.

Přestože kód popisovaných maker není složitý, opisování jistě není zábavné. Proto je kód těchto maker zveřejněn na WWW stránkách Zpravodaje, konkrétně na `http://bulletin.cstug.cz/bul20031.shtml` (ovšem bez komerčního střešovického fontu).

Summary: Fractal Images in PostScript

The picture used on the cover of this issue is an example of a fractal image. The article describes the PostScript macro by means of which the picture was created.

Zdeněk Wagner
wagner@cesnet.cz



TUGboat 22(1–2), March/June 2001

Redakce prosí všechny potenciální autory o příspěvky do Zpravodaje č. 3 a 4/2003. Jejich uzávěrky byly předběžně stanoveny takto:

3/2003	4. 9. 2003
3/2003	20. 11. 2003

Příští číslo nebude monotematické. Najdete v něm, mimo jiné, *Manuál k CSTEXu*. Autorem článku je Petr Olšák.

Zpravodaj Československého sdružení uživatelů TEXu
ISSN 1211-6661 (tištěná verze), ISSN 1213-8185 (online verze)

Vydalo:	Československé sdružení uživatelů TEXu vlastním nákladem jako interní publikaci
Obálka:	Antonín Strejc
Počet výtisků:	750
Uzávěrka:	7. dubna 2003
Odpovědný redaktor:	Zdeněk Wagner
Redakční rada:	Petr Aubrecht, Matěj Cepl, Jiří Demel, Jana Chlebíková, Jiří Kosek, Jaromír Kuben, Petr Sojka, Martin Tkadlčík
Tisk a distribuce:	KONVOJ, spol. s r. o., Berkova 22, 612 00 Brno, tel. +420 549 240 233
Adresa:	CSTUG, c/o FEL ČVUT, Technická 2, 166 27 Praha 6
Tel:	+420 224 353 611
Fax:	+420 233 332 938
Email:	`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

`gacstug@cstug.cz`
grantová agentura CSTUGu

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

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

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

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

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

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

Uzávěrka příštího čísla: 29. května 2003