CSTUG

ZPRAVODAJ

Československého sdružení uživatelů TEXu

3

2010

ISSN 1211-6661 Tištěná verze

ISSN 1213-8185 Elektronická verze

Ročník 20

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/`.

Zpravodaj je zařazen do Seznamu recenzovaných neimpaktovaných periodik vydávaných v České republice, viz `http://www.vyzkum.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, CD, či DVD na adresu: Zdeněk Wagner, Vinohradská 114, 130 00 Praha 3. Redakci lze kontaktovat přes `zpravodaj@cstug.cz`.

Nezapomeňte přiložit všechny soubory, které dokument načítá (s výjimkou standardních součástí TEX Live), zejména v případě, kdy vás nelze kontaktovat e-mailem.

ISSN 1211-6661 (tištěná verze), ISSN 1213-8185 (elektronická verze)
DOI: 10.5300/Zpravodaj

Úvodníček（緒言）

Pavel Stříž

Dobrý den, drahé TEXistky, drazí TEXisté,

dostáváte do rukou netradiční číslo, jehož příprava vznikala pomalu, ale s velkým odhodláním. Číslo začíná překladem článku Denise Roegela（ロジエルデニ）o abaku a pokračuje článkem o sazbě japonštiny pomocí pTEXu (na TEX Live bude poprvé až letos; od roku 2010) z dílny Tima Eyreho（ティムエール）. Následuje krátký vstup o balíčku genzi od Kazuomi Kuniyoshiho（國吉一臣）pro XƎTEX. Práce s třídami znaků je v TEXu relativně nová a stále neznámá.

Po netriviálních, i když oboustranně vstřícných diskuzích získal CSTUG dvouúrovňový souhlas – autor, poté firma Adobe – s přetištěním technické zprávy o novém písmu Kazuraki（かづらき）, včetně náhledů všech glyfů a ligatur, ano, ligatur v japonštině! Ačkoliv písmo není šířeno pod svobodnou licencí, jeho technické aspekty, především přes uvolněný program firmy Adobe AFDKO, http://www.adobe.com/devnet/opentype/afdko/, by mohly nejednoho čtenáře zaujmout, i když třeba pracujete ve FontForge, FreeType2 či jiných.

Bohužel se nepodařilo získat souhlas japonské grafičky u knihy ukázek, ale to našemu čtenáři nemusí vadit, neb snadno nalezne PDF verzi této brožurky, http://store4.adobe.com/type/browser/pdfs/Kazuraki_SPN.pdf. Poděkování patří Kenu Lundemu（小林劍）, kterému se nápad s přetištěním technické zprávy ve střední Evropě líbil, i když v Evropě nikdy nebyl.

Další část tohoto čísla tvoří dva články Tima Eyreho. Jeden o písmu Kanji Stroke Order Font（漢字の筆順のフォント）, stáhnutelné i jako balík pro Linux, a druhý o jeho přístupu ke srovnávání dvou PDF dokumentů. Timu Eyremu patří také velké díky, neboť pečlivě své články čistil a pomohl s korekturou dalších.

Následuje shrnutí sazby čínštiny od Jjgod Jianga（江疆）. Možná někteří znáte jeho balíček gezhu（割注）, japonsky warichu, http://code.google.com/p/gezhu/, či balíček zhspacing, http://code.google.com/p/zhspacing/.

U čínštiny a japonštiny zůstaneme, a to u hry sudoku（数独）. O vykreslení na úrovni METAPOSTu za pomoci kandži se zmiňuje překlad článku Denise Roegela. Následuje článek o balíčku sudokubundle od Petera Wilsona, který patří k pionýrům TEXu. Čtenáři TUGboatu si možná vybavují jeho sloupky nazvané *Glisterings* či jeho čtivý článek z roku 2005 o abecedách a písmových systémech, viz http://tug.org/TUGboat/Articles/tb26-3/tb84wilson.pdf.

Závěr tvoří zmínka několika knih především se vztahem k sazbě čínštiny, japonštiny, korejštiny a vietnamštiny（中日韓越）a zprávy účastníků z různých akcí a konferencí. Přejeme příjemné, nikým a ničím nerušené čtení!

Pavel Stříž（パベル）, *zpravodaj@cstug.cz*

Jednoduché makro suanpan na kreslení čínského a japonského abaku

Denis Roegel

Věnováno 荷花

Abstrakt

Článek představuje způsob, jak si lze v METAPOSTu připravit čínský (算盘, suànpan) a japonský (算盤, そろばん, soroban) abakus.

Jedná se o mechanické počítadlo usnadňující elementární matematické operace a úkony. Kuličkové počítadlo používané v prvních ročnících základní školy u nás je jednou z podob abaku. Podrobněji o abaku samotném viz např. webový rozcestník `http://www.ee.ryerson.ca/~elf/abacus/`.

Článek navíc představí způsob sčítání hodnot a implementaci tohoto algoritmu v METAPOSTu. Makro představené v článku lze rovněž stáhnout ze serveru CTAN.ORG.

Klíčová slova: METAPOST, makro **suanpan**, abakus, suànpan, soroban.

doi: 10.5300/2010-3/138

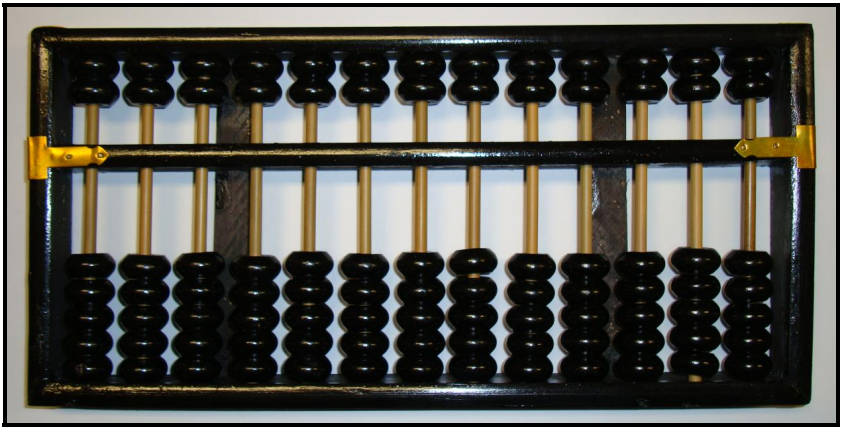
1. Představení abaku

Abakus je jedním z nejstarších počítadel, které se používá dodnes [6, 7, 9–13]. Používají jej především v Asii na základní matematické úkony. Ještě poměrně nedávno byl abakus (anglicky j. č. abacus, mn. č. abaci i abacuses, také v přepisu calculating table, board nebo frame) vyučován na čínských školách a byly z něj skládány zkoušky, pokud měl student zájem se ucházet o některá povolání. V Japonsku byla taková zkouška poprvé skládána v roce 1928 v Tokiu.

Zkušený počtář na abaku může být opravdu rychlý, nezadá si s vámi na běžné kalkulačce, tedy samozřejmě na základní operace s menšími čísly, jako je sčítání a násobení. Abakus může být využit i na náročnější úkony, jako je násobení, výpočet zlomků, druhé mocniny či druhé a třetí odmocniny. Na tento druh úkonů může být potřeba nestandardní abakus, dostatečně velký na to, aby dokázal ukládat potřebné mezivýpočty.

Zjednodušeně řečeno abakus je nástroj, který ukládá čísla pozicí svých korálků (kuliček, oblázků či kamínků; angl. beads) na tyčkách (v žlábcích, na drátech, šňůrkách apod.; angl. on rods). Uložená čísla mohou být jednoduchým

This article is a translation of the article "METAPOST macros for drawing Chinese and Japanese abaci", which first appeared in *TUGboat*, Volume 30 (1), pp. 74–79, 2009. Reprinted with permission. Translation and Czech abstract by Pavel Stříž. The author took the opportunity of this translation in order to make a few minor changes for the sake of clarity, on the suggestion of the translator.



Obrázek 1: Tradiční čínský abakus (算盘, suànpan) se všemi korálky na hodnotě nula. Fotografie je z autorovy kolekce.

způsobem měněna a počtář pracuje s tímto opakujícím se výpočetním vzorem poměrně snadno.

Abakus má dlouhou historii a existovala celá plejáda jeho variant, které v článku představeny nebudou. Stáří čínského abaku je odhadováno na tisíc let, možná je ještě starší. Jiné civilizace, jako byl Řím a Řecko, používaly podobné nástroje, kdy čísla byla ukládána za pomoci oblázků či speciálních žetonů.

V článku představíme makro naprogramované v METAPOSTu, které jednak nakreslí běžné asijské typy abaku a navíc kroky, jak se s nimi počítá.

2. Typy abaku

Zaměříme se na dva typy abaku – standardní verzi čínského a japonského počítadla, které jsou používány i v současnosti.

2.1. Čínský typ abaku: *suànpan*

Čínská verze je nazývána 算盘 (*suànpan*). Slovo 算 (suàn) v čínštině znamená „počítat“ a slovo 盘 (pan) „v rámečku“ nebo na „na destičce“. Abakus *suànpan* může mít celou řadu délek. Běžně má 13 tyček s pěti korálky ve spodní části rámečku (někdy jako pozemské korálky; angl. earth beads) a dvěma korálky (někdy jako nebeské korálky; angl. heaven beads) v horní části (model 2/5 či 2:5), viz Obrázek 1. Horní a spodní část je oddělena příčkou či jinou formou přepážky.

Každý korálek v horní části má hodnotu pěti korálků v části spodní. Pracujeme-li se čtyřmi spodními korálky a jednou horní, je to dostatečné na výpočty v desítkové soustavě. Vypadá to, že prémiové korálky byly původně použity na vyjádření číslic šestnáctkové soustavy, což byla soustava vhodná pro tradiční vážení, kdy jeden jīn (斤) je roven šestnácti liǎng (两) (což je přibližně 50 gramů). Můžeme zároveň vzít v úvahu, že tyto prémiové korálky zjednodušovaly a urychlovaly některé z matematických operací, více o tom [11, str. 85].

2.2. Japonský typ abaku: *soroban*

Japonská verze 算盤 (そろばん, *soroban*) je obdobou čínského *suànpanu*, která ji dala svůj původ. „盤“ je tradiční znak pro žlábek, tento symbol je používán i v současnosti. Také základní verze *sorobanu* má obvykle 13 tyček, jsou zde jen čtyři korálky ve spodní části a jeden v části horní (model 1/4). V některým případech se můžeme setkat s variantou, že je ve spodní části pět korálků a v horní je jen jeden (model 1/5).

Varianta *soroban* má ještě jednu vlastnost, která ji odlišuje od *suànpanu*, a to, že každá třetí tyčka je označena puntíkem. Jedná se o tzv. jednotkové tyčky (angl. unit rods), můžeme si to v současnosti přiblížit k psaní oddělovačů tisíců. Usnadňuje to na *sorobanu* výpočty a zároveň to zpřehledňuje nastavování korálků. Tímto jednoduchým trikem si lze zvýraznit desetinnou tečku.

3. Makro `suanpan` připravené v METAPOSTu

Jak se s abakem pracuje ukážeme na obrázcích, které jsme vygenerovali v METAPOSTu. METAPOST je silný grafický nástroj, který se výborně hodí právě na takové technické a geometrické rýsování [3,5]. Všechny podklady v tomto článku byly připraveny makrem `suanpan`, volně dostupného na serveru CTAN.ORG. Na makro je však potřeba nahlížet jen jako na jádro, které však může být snadno rozšířeno, např. na úpravu tvaru korálků nebo na představení složitějších algoritmů než toho, který si za chvíli ukážeme.

V době psaní tohoto článku byly známy jen další dva balíčky ze světa TEXu, konkrétně od Alaina Delmotta, na kresbu *sorobanu*, buď přes balíček PSTricks nebo PGF, bohužel ani jeden z těchto balíčků (zatím) neumí žádný z výpočetních postupů [1, 2].

4. Počítání na abaku

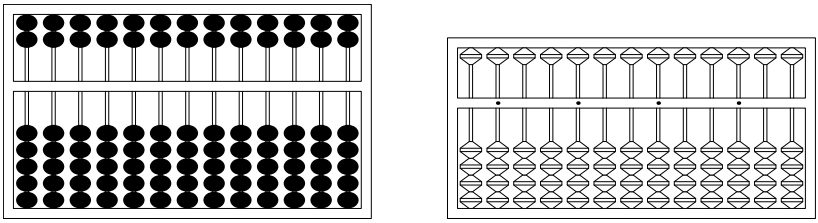
Výpočty s abakem začínají jeho vynulováním, tedy navrácením všech korálků do původní polohy, jinými slovy co nejvzdáleněji od příčky, poté nastavením

první hodnoty posuny korálků, a poté následuje matematický úkol dle známého postupu. Výsledek se dá následně přečíst.

4.1. Základní pozice abaku

Obrázek 1 ukazuje základní pozici skutečného *suànpanu* a Obrázek 2 srovnává abakus čínský (obrázek vlevo) s japonským (ten napravo). Dvě části abaku jsou rozděleny dělicí příčkou, chceme-li přepážkou či trámem (angl. horizontal bar, crosspiece, beam nebo reckoning bar).

V základní poloze jsou všechny korálky co nejdále vzdáleny od příčky, představuje to na všech tyčkách hodnotu nula. Každý korálek představuje jeden řád v desítkové (někdy i šestnáctkové) soustavě. Jednotky jsou nejčastěji vpravo. Tyčky se obvykle číslují, ale tato vlastnost může být v našem makru vypnuta nastavením přepínače `rod_numbers` typu pravda-nepravda (angl. boolean), jak je předvedeno v ukázce.



Obrázek 2: Základní pozice počítadel: 算盘(*suànpan*, vlevo) a 算盤(*soroban*, vpravo).

Užijeme-li makro `suanpan`, lze základní pozici *suànpanu* získat následujícím způsobem:

```
input suanpan
setup_abacus(N=13, NBL=5, NBU=2,
             bead="suanpan", units=0);
beginfig(1);
  rod_numbers:=false;
  reset_abacus;
  draw_abacus;
endfig
end
```

Makrem `setup_abacus` můžeme nastavit počet tyček (`N`), stejně tak počet korálků v obou částech abaku (`NBL` a `NBU`), typ korálků (`bead`) a zdali si přejeme značení u každé třetí tyčky (`units`). Parametry jsou zadávány jako dvojice *proměnná=žádaná hodnota*. V současné verzi makra lze nastavit dva typy ko-

rálků, odpovídající textovým řetězcům `"suanpan"` (téměř korálky kruhu, jemně zploštělé po délce) a `"soroban"` (korálky tvaru dvoukužele).

4.2. Nastavování hodnot

Nastavení hodnoty na abaku znamená posunutí korálků k příčce. Korálek ve spodní části má hodnotu jedné příslušného řádu (jednotky, desítky, stovky atd.) a korálek v horní části představuje hodnotu pěti. Nastavení čtyř korálků ve spodní části a jednoho korálku v horní dává dohromady $5+4=9$. Pokud jsou použité všechny korálky na čínském typu 2/5, a v horní části se stále pracuje s ekvivalentem pěti, dostáváme maximální hodnotu na tyčce rovnu $5+5+5=15$. Tímto způsobem můžeme nastavit všechna čísla od nuly (výchozí postavení) po hodnotu patnáct (maximální hodnota).

Co se týká makra `suanpan`, zde je počet korálků v každé části uložen ve dvou polích s dimenzí. Libovolná hodnota může být u každého pole nastavena ručně takto (Obrázek 3):

```
beginfig(3);
  reset_abacus;
  valL[1]:=2; valL[3]:=5;
  valU[2]:=1; valU[4]:=2;
  draw_abacus;
endfig;
```

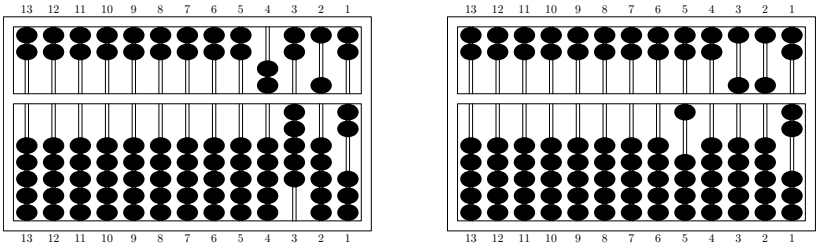
Zpracování tímto způsobem může být užitečné tehdy, když je potřeba nastavit nějakou v desítkové soustavě nestandardní situaci (nastavení čísla 10 552 na Obrázku 3, vlevo). Tohle je konkrétně výše zmíněný případ, kdy bylo potřeba na tyčce nastavit pět korálků ve spodní části. Obvykle nám ve spodní části vystačí vždy jen čtyři korálky. Obdobná výjimka platí pro dva korálky v horní části, počítáme-li v desítkové soustavě.

Makro `suanpan` v současné verzi nepodporuje výpočty v šestnáctkové soustavě, ale mohou být jednoduše zařízeny nadstavením výpočtů založených na soustavě desítkové.

Pomocné makro `set_abacus_val` dokáže operaci nastavení počátečních hodnot zautomatizovat (Obrázek 3, vpravo):

```
beginfig(4);
  reset_abacus;
  set_abacus_val("10552");
  draw_abacus;
endfig;
```

Máme-li pod `N` uložen počet tyček, je automaticky brán zprava jen navolený počet znaků z textového řetězce vstupujícího do makra `set_abacus_val`. Ostatní cifry jsou ignorovány.



Obrázek 3: Nestandardní pozice hodnoty 10 552 v desítkové soustavě (vlevo) a její standardní vyobrazení (vpravo).

4.3. Operace sčítání

Jakmile máme počáteční hodnotu nastavenu, můžeme ji systematicky měnit. V tomto článku se zaměříme jen na operaci sčítání. Dokonce i pro tak triviální operaci můžeme zvolit několik způsobů, my si ukážeme typický způsob sčítání ne zprava doleva, ale zleva doprava.

Abychom ukázali, jak to funguje, připravili jsme v makru `suanpan` makro, chceme-li funkci, `add_val`, která sčítání rozloží na nezbytný počet kroků. Jen si dejme pozor, že tohle makro nemůže být běžnou součástí prostředí `beginfig` a `endfig`, je to z toho důvodu, že makro tato prostředí generuje za svého běhu.

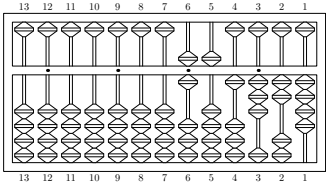
Ukážeme si příklad na *sorobanu* s počáteční hodnotou 651 324 tak, jak to zmiňuje následující zdrojový kód (Obrázek 4):

```
setup_abacus(N=13, NBL=4, NBU=1,
             bead="soroban", units=1);
set_abacus_val("651324");
add_val(v="82363456", iv=100, fig=true);
```

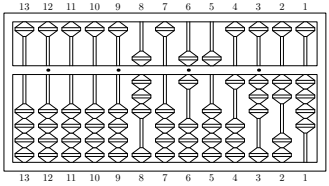
Užíváme-li jako řídicí soubor `abacus.mp`, pak zmíněná sekvence příkazů v něm uložených vygeneruje soubory `abacus.100`, `abacus.101`, ..., `abacus.108`, které již mohou být rutinně přiloženy do TEXového dokumentu.

Po vynulování následují kroky dle Obrázku 4. Úvodní stav (a) ukazuje nastavení prvního sčítance 651 324, zde náš proces sčítání začíná. V prvním kroku (b) přidáváme 8 na osmou tyčku tím způsobem, že přesuneme k příčce tři spodní korálky a jeden horní. S ostatními korálky není hýbáno. Následuje krok (c), který přidává 2 na sedmou tyčku. Zatím máme život snadný, neboť změny se dějí vždy na jedné tyčce, poněvadž byly nastaveny na hodnotu 0. Je to dáno tím, že první sčítanec nemá řád miliónů ani desetimiliónů. V kroku (d) přidáme 3 na šestou tyčku. Z hodnoty 6 se stává hodnota 9.

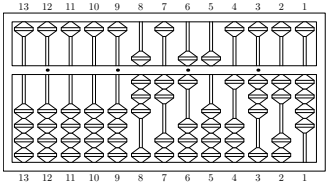
Krok (e) přidává 6 na pátou tyčku, která obsahuje 5. To nám součtem dává 11. Na páté tyčce nastavíme hodnotu 1 a jedničku si budeme chvíli pamatovat a pokusíme se tuto jedničku přidat na tyčku šestou (tyčka vlevo). Tato vzniklá situace



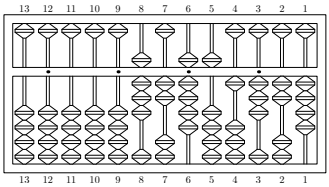
(a) Nastavíme 651 324.



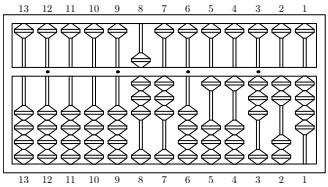
(b) Po přičtení 80 000 000.



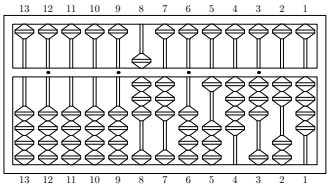
(c) Po přičtení 2 000 000.



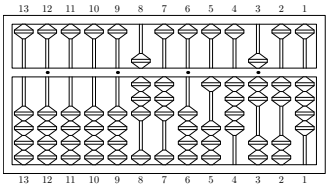
(d) Po přičtení 300 000.



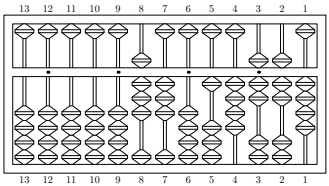
(e) Po přičtení 60 000.



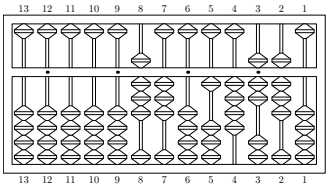
(f) Po přičtení 3 000.



(g) Po přičtení 400.



(h) Po přičtení 50.



(i) Po přičtení 6.

Obrázek 4: Znázornění rozkladu sčítání do devíti kroků na *soroban*. (a) znázorňuje nastavení prvního sčítance 651 324, a přidáváme druhého sčítance 82 363 456 v osmi krocích (b)–(i), každý krok reprezentuje jednu cifru druhého sčítance.

nám zatím ukazuje, že jsme tedy změnili počet korálků ve spodní části na tyčce pět, nenastala žádná změna ve smyslu přičtení v její horní části, která se nuluje. Zaměříme naši pozornost na tyčku šest, kde chceme přidat jedničku. Šestá tyčka obsahuje hodnotu 9. Přidáním jedničky se dostáváme na 10 a je potřeba dalšího přesunu jedničky. Vynulujeme šestou tyčku a přidáme jedničku na tyčku sedm. Ta obsahuje hodnotu 2 a tu zvedneme na hodnotu 3. Dalších přesunů už není třeba a krok (e) je hotov, přičetli jsme 60 000.

Celý tento proces se opakuje cifra po cifře, dokud není celý rozložený sčítanec připsán. Vidíme, že přičtení cifry sčítance se občas rozloží na dílčí kroky. Podrobněji nebudeme detaily v článku rozvádět, v principu se jedná jen o analogii práce s jednou tyčkou, jen v několika krocích rozloženou. K procvičení si lze vyzkoušet součet $999\,999\,999 + 1$.

Pokud by výklad u zmíněných maker nebyl dostatečný, odkazujeme čtenáře přímo na zdrojový kód `suanpan.mp` z CTAN.ORG.

Makro `add_val` může být použito i bez generování obrázků tím, že se nastaví `false` u proměnné `fig`. V takovém případě proběhne jen základní algoritmus sčítání a součet je uložen v polích.

Je tu samozřejmě možnost, že by součty mohly být simulovány tím, že výpočet bude proveden mimo makro `suanpan` a hodnoty budou vždy nastaveny pro každý příchozí požadavek. Takto by bylo možné využít makro `suanpan` jako nástroj vykreslování pro jiná makra, programy či nástroje.

Přikládáme ukázku součtu, kdy nevznikají obrázky z jednotlivých mezisoučtů.

```
beginfig(200);
  reset_abacus;
  reset_abacus_gray;
  set_abacus_val("82951324");
  draw_abacus;
endfig;
beginfig(201);
  set_abacus_val("82951324");
  add_val(v="60000", iv=100, fig=false);
  draw_abacus;
endfig;
```

Operace sčítání může způsobit problém s přeplněním a nastavila by se proměnná `overflow` typu pravda-nepravda na hodnotu `true`. Před uskutečněním sčítání si makro `add_val` vynuluje tuto proměnnou na `false`.

4.4. Triky na zrychlení výpočtů

Pokud chceme zefektivnit práci s abakem, je užitečné si zapamatovat některé vzory, které se objevují často a dávají nám prostor k jejich automatizaci. Na

jednoduché ukázce si to osvětlíme. Pokud některá z tyček je ve spodní části nastavena na hodnotu 3 a jeden korálek máme přidat, pak nezbývá nic jiného než čtvrtý korálek posunout směrem k příčce. Zde nemáme co vylepšovat, počet stupňů volnosti je nula.

Pokud je však situace drobně upravena, tedy máme-li přidat tři korálky místo jednoho ke stávajícím třem, pak můžeme pravděpodobně u začátečníka očekávat myšlenkový proces založený na výpočtu $3 + 3 = 6$, poté odečtením 5 a nastavením jednoho korálku ve spodní i horní části abaku. Věřte či nikoliv, ale tato situace je neefektivní, poněvadž výpočetní zátěž leží na počtáři.

Uvažujme takto. Víme-li, že tři korálky nemohou být v dané chvíli přemístěny, pak můžeme rovnou zvážit vztah $3 = 5 - 2$ a dostat se ke dvěma úkonům: za prvé, *přidání* jednoho korálku (hodnoty 5) v horní části, a za druhé, *odebrání* dvou korálků v části spodní. Tohle je příjemná matematická zkratka, která nevyžaduje výpočet $3 + 3 = 6$, a to jen díky tomu, že si zavčas všimneme, že nemůžeme ve spodní části přidat ke třem stávajícím korálkům tři další.

Podobně nerealizovatelné úkony si lze představit i v horní části. Můžeme si připravit podobné schéma. Jakmile není možné přidat korálek o hodnotě pět do horní části, uvažme užití formule $5 = 10 - 5$, technicky přidáme jeden korálek (z pohledu aktuální tyčky je hodnoty 10) ve spodní části další tyčky (opět vlevo od aktuální) a ubereme korálek v horní části původní tyčky. A opět se tento postup opakuje tak dlouho, dokud nesečteme celý sčítanec.

Uvažme situaci, kdy chceme přidat pět nebo více korálků na libovolnou tyčku ve spodní části. Tuto situaci můžeme zapsat buď rozepsáním na $5 + a$ nebo $10 - b$, a je-li jedna z nich použitelná, bude aplikována. Konkrétně můžeme mít tři korálky na tyčce v její spodní části a chceme přidat dalších šest. Buď můžeme tuto situaci rozepsat jako $6 = 5 + 1$ nebo $6 = 10 - 4$. Druhý rozklad nelze technicky zrealizovat, protože nemůžeme odebrat čtyři korálky, když máme jen tři. V tomto případě je však možná první situace. Přidáme jeden korálek ve spodní části a pokusíme se přidat jeden korálek v horní části. Pokud by to v horní části nebylo možné, provedeme další rozklad atd.

Náročnější matematické úkony, jako je násobení, dělení, druhá odmocnina atd., mohou být aplikovány rychle a účinně za použití pomocných tabulek, které si předtím musel počtář vštěpit do paměti. Ukázky takových tabulek pro *soroban* předkládá např. Knott [7].

5. Účelová makra pro rozšíření abaku

Chceme-li někomu vysvětlit práci s abakem, je poměrně často výhodné využít nějaké formy značení nebo zvýraznění jednotlivých korálků. Makro `suanpan` nabízí dvě možnosti. Buď je volený korálek vykreslen jinou barvou, konkrétně šedou, nebo je na něj přidán popisek.

Je poměrně snadné využít makra `set_abacus_gray` a vykreslit vybrané korálky v šedé barvě. Makro má tři parametry, zadávané opět ve formě párů *proměnná=žádaná hodnota*. Proměnná `deck` nám upřesní, jestli pracujeme s horní (`upper`) nebo spodní (`lower`) částí abaku. Další dvě proměnné jsou textové řetězce udávající korálky brané zprava. Hodnota `below` odpovídá korálkům vespod navolené části (horní nebo spodní části abaku), opakem je hodnota `above`. Například zapsání hodnoty 2 při `below` znamená, že budou šedou barvou vykresleny dva korálky zespodu. Tedy ty dva, které jsou brány za první kandidáty na posun k příčce (při volbě `lower`), resp. ty dva, které se jako první vrací do základního stavu (při volbě `upper`).

Při `below` to je obdobné, jako kdybychom brali vrcholky stalagmitu; při `upper` to je, jako bychom odebírali spodní špičky stalaktitu.

V současné verzi makra není tento způsob obarvování zautomatizován, i když je to samozřejmě technicky možné. Můžeme si představit celou řadu možných schémat, které jsme do makra nezanesli. Realizace v současné verzi je podpořena jen základními příkazy.

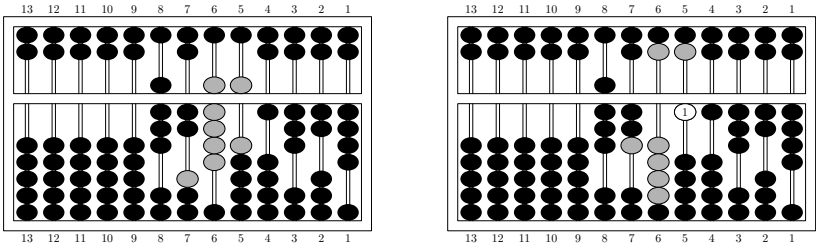
Nechť nám za ukázku poslouží součet z dřívějšího příkladu vyobrazeného na Obrázku 4, konkrétně přechod z kroku (d) do (e), tentokrát však tak, aby byly zvýrazněny všechny posuny korálků. Jednalo se konkrétně o situaci, kdy jsme k mezihodnotě 82 951 324 přičítali 60 000. Výsledek našich snah si můžete prohlédnout na Obrázku 5 a zde přikládáme příslušné zdrojové kódy:

```
beginfig(202);
  reset_abacus;
  reset_abacus_gray;
  set_abacus_val("82951324");
  set_abacus_gray(deck="lower",
     below="1010000", above="0400000");
  set_abacus_gray(deck="upper",
     below="0110000", above="0000000");
  draw_abacus;
endfig;

beginfig(203);
  reset_abacus_gray;
  add_val(v="60000", iv=100, fig=false);
  set_abacus_gray(deck="lower",
     below="0400000", above="1010000");
  set_abacus_gray(deck="upper",
     below="0000000", above="0110000");
  draw_abacus;
  mark_abacus(5,5)(btex 1 etex);
endfig;
```

V daných ukázkách makro `reset_abacus_gray` plní jen ulohu vynulování dříve šedou barvou vyznačených korálků. Bude to pro čtenáře příjemnější, aby viděl právě jen ten jeden přechod.

Dalším nově představeným makrem na vyznačení korálků je `mark_abacus`. Tohle makro umí překreslit korálek tak, aby mohl obsahovat (krátký) popisek. Konkrétně `mark_abacus(5,5)(btex 1 etex)` zapíše cifru jedna na pátý korálek odspodu na pátou tyčku počítanou zprava. Jednou z výhod tohoto přístupu je, že i když se později korálek posune, značka u něj zůstane, aniž bychom příkaz jakkoliv upravovali nebo si jej znovu volali.



Obrázek 5: Zvýraznění všech přesunů v jednom z mezikroků při sčítání 829 513 24 s 60 000. Všechny korálky, které byly přesunuty, jsou vykresleny šedou barvou, a navíc jsme si vyznačili aktuální hodnotu na tyčce, se kterou bylo pracováno. V našem případě to byla hodnota 60 000, tedy pátá tyčka.

6. Využití abaku v jiných číselných soustavách

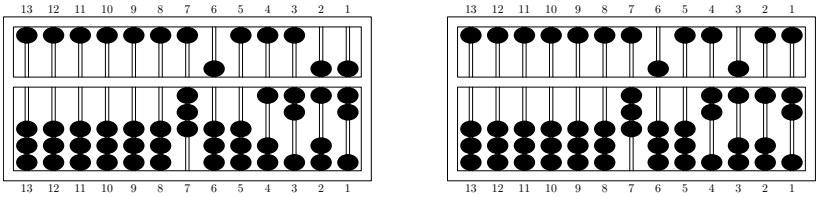
Jak jsme dříve zmínili, čínský abakus může být použit na výpočty v desítkové i v šestnáctkové soustavě, záleží jen na tom, jestli použijeme dva bonusové korálky (jeden v horní části a druhý ve spodní). Po chvíli přemýšlení však můžeme abakus upravit tak, aby sloužil i pro jiné číselné soustavy.

Obrázek 6 nám ukazuje sčítání v osmičkové soustavě. Každá tyčka má čtyři korálky, tři ve spodní části s hodnotou jedna a jeden korálek v horní části s hodnotou čtyři. Tímto způsobem sestavíme hodnoty nula až sedm. Levý obrázek představuje nastavení čísla $3\,401\,256_8$, přičtením 1234_8 dostáváme součet $3\,402\,512_8$, což je vyobrazeno vpravo.

Příprava těchto obrázků je přímočará. Upravíme počet korálků na každé tyčce, dále nastavíme proměnnou `vbu`, která představuje počet korálků v horní části abaku. Takový abakus v osmičkové soustavě má hodnotu horního korálku čtyři, což odpovídá počtu korálků ve spodní části plus jeden. Stačí nám nyní zapsat tento zdrojový kód, abychom dostali náhled před sčítáním a po něm.

Podobně můžeme připravit další číselné podklady. Jen upozorníme zkušené počtáře, že je nutné upravit mnemotechnické pomůcky, které jsme poodhalili v tomto článku dříve tak, aby těmto novým podmínkám vyhovovaly.

```
vbu:=4; % Hodnota kuličky v horní části abaku.
setup_abacus(N=13, NBL=3, NBU=1,
             bead="suanpan", units=0);
beginfig(300);
  reset_abacus;
  reset_abacus_gray;
  set_abacus_val("3401256");
  draw_abacus;
endfig;
beginfig(301);
  add_val(v="1234", iv=100, fig=false);
  draw_abacus;
endfig;
```



Obrázek 6: Sčítání na abaku v osmičkové soutavě. Hodnota vlevo je $3\,401\,256_8$, po přičtení hodnoty 1234_8 dostáváme součet $3\,402\,512_8$, který je k nahlédnutí vpravo.

7. Závěry a možná rozšíření makra `suanpan`

Cílem článku bylo ukázat jednoduché makro kreslící čínský a japonský abak. Snažili jsme se především poskytnout jisté základní stavební kameny. Vylepšení jsou možná jak na straně grafiky, tak na úrovni algoritmů. Abakus může být vykreslen mnohem realističtěji, a co je obzvláště příjemné, další typy korálků lze jednoduše přidat.

Největší prostor pro rozšíření a vylepšení je ale na straně algoritmů. Současná verze makra obsahuje jen jeden postup na sčítání. Mohli bychom si např. připravit algoritmus pro sčítání z pravé části do levé (tj. od jednotek po vyšší řády). Podobně bychom mohli přidat řadu dalších algoritmů.

Náročnější matematické úkony, jako je násobení, dělení, výpočet druhé či třetí odmocniny, ale celá řada dalších [4, 7, 8, 11], by nemělo být náročné do jádra makra `suanpan` implementovat.

Pro každý z takto zvažovaných algoritmů by bylo vhodné jej naprogramovat tak, aby bylo možné na výstupu, graficky a výpisem proměnných, sledovat jednotlivé kroky a mezikroky matematických úkonů.

Makro `suanpan` vám k tomu všemu dává stavební základnu.

Seznam literatury

[1] Delmotte, Alain. Soroban abacus: package pgf-soroban. [Balíček pgf-soroban na kresbu abaku japonského typu soroban přes TEXový balíček PGF.] [online cit. 12. 11. 2007] Balíček je dostupný na URL: `http://ctan.org/tex-archive/graphics/pgf/contrib/pgf-soroban/`

[2] Delmotte, Alain. Soroban abacus: package pst-soroban. [Balíček pst-soroban na kresbu abaku japonského typu soroban přes balíček PSTricks.] [online cit. 3. 9. 2009] Balíček je dostupný na URL: `http://ctan.org/tex-archive/graphics/pstricks/contrib/pst-soroban/`

[3] Goossens, Michel; Mittelbach, Frank; Rahtz, Sebastian; Roegel, Denis; Voß, Herbert. *The LATEX Graphics Companion.* [Velký průvodce grafikou v LATEXu.] 2. vyd. USA, Boston: Addison-Wesley Professional, 2007. ISBN 978-0-321-50892-8.

[4] Heffelfinger, Totton; Flom, Gary. 算盤 Abacus: Mystery of the Bead. [算盤 Abakus: záhada korálků.] [online cit. 16. 1. 2007] Dostupné na URL: `http://webhome.idirect.com/~totton/abacus/`

[5] Hobby, John. METAPOST: A User's Manual. [METAPOST: Uživatelská příručka.] [online cit. 2. 10. 2009] Aktualizovaná verze původního manuálu. Dostupná z URL: `http://www.tug.org/docs/metapost/mpman.pdf`.

[6] Ifrah, Georges. *The Universal History of Computing: From the Abacus to the Quantum Computer.* [Historie výpočetního světa: od abaku až po kvantové počítače.] 1. vyd. USA, New York: John Wiley and Sons, 2001. ISBN 978--0-471-44147-2.

[7] Knott, Cargill Gilston. The Abacus, in Its Historic and Scientific Aspects. [Abakus z pohledu historického a vědeckého.] *Transactions of the Asiatic Society of Japan*, Volume 14: 18–71, 1886. ISSN 0913-4271.

[8] Kojima, Takashi. *Advanced Abacus: Japanese Theory & Practice.* [Abakus pro pokročilé: teorie a praxe v Japonsku.] Japonsko, Tokio: Charles E. Tuttle and Company, 1963. ASIN B0007DNGUQ.

[9] Shu-T'ien, Li. Origin and Development of the Chinese Abacus. [Vznik a rozvoj čínského abaku.] *Journal of the ACM*, Volume 6 (1): 102–110, 1959. ISSN 0004-5411. doi:10.1145/320954.320962

[10] Martzloff, Jean-Claude. *A History of Chinese Mathematics.* [Historie čínské matematiky.] USA, New York: Springer, 2006. ISBN 978-3-540-33782-9.

[11] Moon, Parry. *The Abacus: Its History, Its Design, Its Possibilities in the Modern World.* [Abakus: historie, umělecké provedení a možnosti využití v současnosti.] USA, New York: Gordon and Breach Science Publishers, 1971. ISBN 978-0-677-01960-4.

[12] Needham, Joseph; Wang, Ling. *Science and Civilisation in China, Vol. 3: Mathematics and the Sciences of the Heavens and Earth.* [Věda a civilizace v Číně, svazek třetí: Matematika a vědy nebes a o Zemi.] Velká Británie, Cambridge: Cambridge University Press, 1959. ISBN 978-0-521-05801-8.

[13] Smith, David Eugene; Mikami, Yoshio. *A History of Japanese Mathematics.* [Historie japonské matematiky.] Chicago: The Open court publishing company, 1914. Přetištěno jako Smith, David Eugene; Mikami, Yoshio. *A History of Japanese Mathematics.* 1. vyd. Cosimo Classics, 2007. ISBN 978-1--60206-664-9.

Summary: METAPOST macros for drawing Chinese and Japanese abaci

This article shows how Chinese (算盘, suànpan) and also a Japanese version of abaci (算盤, そろばん, soroban) can be drawn with METAPOST, and illustrates it with the details of a simple algorithm.

The source codes are included as small parts in the article commented in detail. You may find the original English version of the article in *TUGboat*, see `http://www.tug.org/members/TUGboat/tb30-1/tb94roegel-abacus.pdf`.

Keywords: METAPOST, Chinese and Japanese abacus, abaci, abacuses, suànpan, soroban, `suanpan` macro.

Denis Roegel, roegel@loria.fr
http://www.loria.fr/~roegel
LORIA – Campus Scientifique, BP 239
F-54506 Vandœuvre-lès-Nancy Cedex, France

Sazba japonštiny pomocí pTEXu

Timothy Eyre

Contents

Abstrakt

Nástroj pTEX je sázecí systém bohatě využívající možnosti TEXu. pTEX je speciálně navržen pro sazbu japonštiny a je používán především v Japonsku. Článek popisuje jak získat, nastavit si a používat pTEX v každodenním životě s praktickými úlohami, a to s důrazem na správu písem. Článek nás také seznamuje se základy sazby japonštiny obecně a s alternativami vůči systému pTEX.

Klíčová slova: pTEX, pLATEX, W32TEX, sazba japonštiny, kandži, hiragana, kana, katakana, Unicode, ruby, ČJKV.

doi: 10.5300/2010-3/152

Introduction

The program pTEX from ASCII Media Works is an effective tool for typesetting Japanese. Unfortunately I've never been able to find much in the way of English documentation for pTEX. This article gathers together the knowledge I've accumulated on pTEX through web-searching, inspired guesses, hair-pulling, inspecting code and doing my best to make sense of the Japanese documentation.

In this article I assume you are using Microsoft Windows. If you use Linux then you will be sufficiently computer-literate to apply what is written here to your environment. Macintosh users might also find some of the information here useful. I have tried the Macintosh distribution of pTEX and it works well. I also assume that you are familiar with using the MS-DOS command-line interface and basic tools like `gzip` and `tar`.

1. Acquiring and installing pTEX

Point your web browser at `www.w32tex.org` [1]. This is the download site for W32TEX. There is an English version of the page. A good thing about this installation is that the maintainer updates it every few days. Download all the packages from the Basic and Standard Installation sections. If you fancy any of the packages in the Full Installation section then download those too.

One of the things I like about W32TEX is that the packages are just gzipped tar files. The installation includes an installer but you can just `gunzip` all the files and `tar -xvf` them yourself. My W32TEX installation takes up about 250 MB of disk space.

Once you've done this you'll need to add `c:\usr\local\bin` to your path and modify the `texmf.cnf` file to reflect your system. Of course, the details of this are outside the scope of this article.

If you are reading this article then you are probably already familiar with TEX and therefore probably already have a TEX system installed. If you can

get pTEX to work on your existing installation then I'm happy for you. I never managed to do it. For a while I had a TEX Live installation running alongside a pTEX installation. This worked fine; I just had to change my path when I was using Japanese so that I picked up the W32TEX binaries instead of the TEX Live binaries. Eventually I migrated to W32TEX. W32TEX is what most Japanese people seem to use. The good thing about W32TEX is that it handles Japanese without needing any extra configuration.

As an aside, pLATEX seems to be more popular in Japan than LATEX is in the West. In Japan I see a few books on pLATEX in most larger book shops.

Installation on Linux

You need to find the extra packages for your distribution that include pTEX. You'll also need Adobe Reader or `xpdf` with the Japanese support package; a Japanese font, such as `kochi-mincho.ttf`; and `dvipdfmx`. Once all these are installed you can compile documents that are in Shift-JIS format by running

```
ptex -kanji=sjis myfile.sjs
```

This will probably work. However, when you `dvipdfmx myfile.dvi` you'll probably get a failure.

I fixed this by copying the contents of the `cmap` directory in my W32TEX installation to my Linux installation. Then I updated `texmf.cnf` via

```
/etc/texmf.d/<somefile>
```

and

```
update-texmf
```

(this seems to be a feature of teTEX) to point at the directory in my installation that contains my TrueType fonts (`/usr/share/fonts/truetype/`). Finally I updated `x-cid.map` to add the line

```
rml H kochi-mincho
```

`H` refers to something called a CMap resource. You'll find it in the `cmap` directory you copied over.

2. Entering Japanese text

2.1. Encodings

In the world of computers all data is stored as numbers. You will already know that the characters a–z, A–Z, 1–9 and some punctuation marks are represented by numbers between 0 and 127. The number used to represent each character is defined by the ASCII standard[1]. Because we only need the numbers between

[1] IBM mainframes use a character encoding called EBCDIC, which does not represent consecutive letters by consecutive numbers. I've never seen EBCDIC used with TEX.

0 and 127 to represent plain English we can store each character in an English text file as a byte. Languages such as French and Czech that include accented characters can also be represented by text files that use just one byte for each character. However, to represent the accented characters they also make use of the numbers between 128 and 255. You may already know that there is no one standard for the characters represented by the numbers between 128 and 255; the character that is represented by one of these numbers is defined by the *encoding* that is being used.

This state of affairs is reflected in the development of TEX. When Knuth first released TEX each font had 128 character slots. A later version gave each font 256 character slots, thus enabling people to use the full width of a byte to represent character.

Japanese has far more than 256 characters. Therefore we need to use bigger numbers to represent the characters. This is typically achived by using multiple bytes to represent each character. Using two bytes provides us with 65,536 slots to put characters in. This is enough even for Japanese. However, the details of representing complex writing systems in computers is rather more complex than this. Full details are beyond the scope of this article and can be found in [2].

One might think that using multiple bytes to represent the world's most bewildering writing system is complex enough. However, as is usually the case with software, we have another layer of complexity: there is no one standard encoding for representing Japanese characters. The most common ones are Shift-JIS, ISO-2022-JP ('JIS'), EUC-JP and the various encodings of the Unicode character set such as UTF-8, UTF-16 and UTF-32.

Unicode and its encodings are the closest we have to an industry-wide standard for encoding the written word. A disadvantage of the UTF-16 and UTF-32 encodings of Unicode is that if you send Japanese text encoded in one of these formats to a destination that can read ASCII but not Unicode then the recipient cannot read any of the text. This would be particularly unfortunate if there were only a few Japanese characters in the message. This is the advantage of the UTF-8 encoding; it keeps all the ASCII characters as single bytes of ASCII. The disadvantage of UTF-8 is that it uses significantly more bytes than UTF-16 to encode the same number of non-ASCII characters. UTF-32 is inefficient in its use of space and is rarely used.

The JIS encoding was devised in Japan; it stands for *Japanese Industrial Standard*. This system has the same disadvantage as Unicode in that Western characters will not survive if the JIS text is displayed on a JIS-incapable device. Thus Shift-JIS (sometimes written S-JIS or SJIS) was devised. Confusingly, it was devised by a Japanese company called ASCII Media Works in collaboration with Microsoft. Microsoft adopted this encoding (in a slightly modified form) so it is widely used. ASCII Media Works also produced pTEX so, not surprisingly,

the native encoding of pTEX is Shift-JIS. I always use Shift-JIS unless I have a good reason to do otherwise.

I don't know anything about the EUC-JP encoding except that it tends to be used on UNIX systems. For information on the EUC-JP encoding and comprehensive information on handling Chinese, Japanese, Korean and Vietnamese on computers see [2].

2.2. JWPce

Western versions of Microsoft Windows XP and above include a Japanese text entry system; you just need to fiddle with the settings in the Control Panel to get it working. It works with Notepad and if you're lucky it might work with your favourite text editor. Powerful though this input method is, it is more aimed at native Japanese speakers than students of the language.

Much better for people like me is JWPce [3]. It's a free download and comes with plenty of help and documentation. Features that are useful for students include the built-in dictionary and the built-in kanji information look-up. It also has three Japanese fonts built into it.

Download JWPce from [3] and install it. When you save your TEX source use the Shift-JIS encoding (`.sjs`).

2.3. Adobe Reader

You will probably want to view your pTEX creations as PDF files. If you don't already have Adobe Reader installed, install the latest version. Also install the Japanese language pack. The fonts included in this language pack are enough to get you going with pTEX. You don't even need the Windows Japanese fonts.

In Japan people seem to use a DVI previewer called DVIout. It is also possible to run a Japanese-enabled version of `dvips` (see below) and view the results using GhostView.

2.4. Japanese Fonts 日本語の字体

If you want to view your pTEX output as PostScript then you will need to install the Windows Japanese fonts. You can do this from the Control Panel. The fonts are called `msmincho.ttc` and `msgothic.ttc`. Yes, it is counter-intuitive to need TrueType fonts to view a PostScript document.

3. Other Japanese-Capable TEX Systems

There is a LATEX package called CJK that provides another way to typeset Japanese text in TEX. It allows you to typeset Korean and Chinese as well as Japanese. It has documentation in English.

The future of polyglot TEX typesetting appears to lie with XETEX. This system has now been ported from the Macintosh OS X platform to both Linux and Windows. The Windows installation is done as a bolt-on to W32TEX; the W32TEX download site includes a binary package and English installation instructions. I have got both these systems up and running. The Windows installation took a matter of minutes. XETEX is now also available with TEX Live.

There is a Japanese version of a program called Omega, a version of TEX that can handle 16-bit encodings. There seems to be little activity or documentation on this project.

jTEX is an early (c.1987) Japanese-enabled TEX variant created by NTT. It is still available for download but has been largely superseded by pTEX. An article on the development of this package has been published in TUGboat [4].

The UMS package allows you to put Japanese text in a file that is to be compiled by pdfTEX or pdfLATEX. You use it by producing a Shift-JIS source file, running this file through a program called `topdftex` and the sending the result to pdfLATEX with the UMS package included. One reason for doing this rather than using pTEX and `dvipdfmx` is that you might want to use some feature that is specific to pdfTEX.

A sample input file is as follows:

```
\documentclass[12pt]{article}
\usepackage{ums}
\begin{document}
私は魚に興味があります。
\end{document}
```

The commands you need to run to obtain a PDF document from a Shift-JIS format file using pdfLATEX are as follows.

```
topdftex source.sjs tmp.sjs
pdflatex tmp.sjs
```

When you run `topdftex`, the resulting file `tmp.sjs` should look like this:

```
\documentclass[12pt]{article}
\usepackage{ums}
\begin{document}
\UMS{79C1}\UMS{306F}\UMS{9B5A}\UMS{306B}\UMS{8208}...
\end{document}
```

To set up the UMS package you need to run the batch jobs in the following two directories

```
C:\usr\local\share\texmf\fonts\type1\public\omegaj\msmin
C:\usr\local\share\texmf\fonts\type1\public\omegaj\msgoth
```

to create all the `.pfb` files. This in turn requires you to install the W32TEX Omega packages.

pTEX is the most popular solution in Japan and, as such, has plenty of Japanese-specific macros available. Judging from the questions on the TEX newsgroup, `comp.text.tex` the CJK package is the most popular solution outside Japan. XETEX describes itself as experimental software whereas pTEX has had many years of field hardening. Both the CJK package and the XETEX system support writing systems other than Japanese whereas pTEX only supports Japanese in addition to those supported by ordinary TEX.

4. Creating a document

Plain pTEX

A remarkable feature of pTEX is that you can enter Japanese text in-line with Western text without any extra markup. pTEX handles all the font switching internally. Here is a simple document in Plain pTEX. Save the file in Shift-JIS (`.sjs`) format.

```
The Japanese symbol for fish is 魚.
\bye
```

pLATEX

Using pLATEX is no more complex; again pLATEX handles everything for you. The only difference is that if your document is intended to be read as being mainly Japanese you should use

```
\documentclass{jarticle}
```

instead of

```
\documentclass{article}
```

This makes the output caption figures with 図 instead of *Figure* and so on. Here is a simple example:

```
\documentclass{jarticle}
\begin{document}
鯨は魚ではありません。
\end{document}
```

5. Viewing documents

Once you have written your pTEX or pLATEX source file you compile it in the obvious way:

```
c:\work>ptex my_document.sjs
```

or

```
c:\work>platex my_platex_document.sjs
```

The resulting file is called `my_document.dvi`. However, the file format is not standard DVI so the standard versions of `dvips` and `dvipdfm` will not be able to convert it into a viewable format. Because of this pTEX is not strictly speaking a version of TEX at all. pTEX does not pass the `trip.tex` test either [6]; this disqualifies it from being a true TEX. However, you are unlikely to notice any problems in practice.

5.1. dvipdfmx

To convert your `.dvi` file into PDF format, run it through `dvipdfmx`. This program comes with the W32TEX installation and does not need any configuration. Run the following two commands and, assuming you have bound `.pdf` files to Adobe Reader, your document should appear on the screen.

```
c:\work>dvipdfmx my_platex_document
c:\work>start my_platex_document.pdf
```

5.2. dvipsv

The `dvipsv` program is a version of `dvips` enhanced to handle the pTEX `.dvi` format and embed the TrueType fonts in the document. If you want PostScript output then this is probably the one to use. It produces large output files because of the embedding. Obviously, you need GhostScript and GhostView installed to view the output.

```
c:\work>dvipsv my_platex_document
c:\work>start my_platex_document.ps
```

5.3. dvipsk

There is a bit of naming convention confusion here. Radical Eye now call `dvips`: `dvipsk`. However W32TEX calls the executable for the standard, non-pTEX version of `dvipsk`: `dvips.exe`. The executable for the version of `dvipsk` that can handle pTEX output is called `dvipsk.exe`.

The advantage `dvipsk.exe` has over `dvipsv.exe` is that it produces smaller output files and runs more quickly. The disadvantage is that it does not embed the fonts in the output so you need to have the fonts installed on the system where you are going to view the PostScript file. Furthermore, if you install a new Japanese font on your system then you need to modify your GhostScript configuration files before you can view your new document. This is covered in detail in a later section.

W32TEX also includes a program called `udvips`. It appears to produce output identical to `dvipsk`.

6. PDF bookmark entries

You have to do a bit of extra work to get PDF bookmarks to work in Japanese script. The PDF special `tounicode` is the key. For Plain pTEX the source would look like this:

```
\def\bookmark#1{\special{pdf: out 1 << /Title (#1) /Dest
                              [ @thispage /FitH @ypos ] >>}}%
\special{pdf:tounicode 90ms-RKSJ-UCS2}
\bookmark{日本語 1}
\bye
```

and in pLATEX

```
\documentclass{jarticle}
\def\bookmark#1{\special{pdf: out 1 << /Title (#1) /Dest
             [ @thispage /FitH @ypos ] >>}}%
\AtBeginDvi{\special{pdf:tounicode 90ms-RKSJ-UCS2}}
\begin{document}
\bookmark{日本語 1}
Hello
\end{document}
```

This technique works for annotations (sticky notes) in `dvipdfm` too. For Plain pTEX source it would look like this

```
\special{pdf:tounicode 90ms-RKSJ-UCS2}
いろはにほへと
\special{pdf: ann width 3.0in height 36pt
<< /Type /Annot /Subtype /Text
/Contents (日本語) >>}
Sphinx of black quartz, judge my vow.
\bye
```

Determining whether `tounicode` works for other `dvipdfmx` contructs too is left as an exercise for the reader. The following standard hyperref package code placed in the preamble to a document will produce PDF bookmark entries:

```
\special{pdf:tounicode 90ms-RKSJ-UCS2}
\usepackage[dvipdfm,bookmarks=true,%
bookmarksnumbered=true,bookmarkstype=toc,%
colorlinks,linkcolor=blue,urlcolor=blue]{hyperref}
```

7. Installing new kanji fonts

The default installation of W32TEX appears to use Microsoft Mincho and Gothic as its only fonts. However, if you use `dvipdfmx` you actually see the Adobe

Reader fonts; `dvipdfmx` writes the PDF file specifying the Adobe Reader fonts as substitutes. To get the real Microsoft Mincho and Gothic fonts you need to run

```
dvipdfmx -f msembed.map file.dvi
```

This results in a larger PDF file because the font is now embedded within it.

When I first started using pTEX I was grateful to be able to typeset Japanese at all; it seemed greedy to want to use other fonts. However, after using pTEX for a while you might want to use a completely different font. It is possible to install new Japanese TrueType fonts into W32TEX. This section explains how.

7.1. Available fonts

There are dozens of free Kanji fonts out there. Do a web search to find them. Epson in particular have a bundle of several Japanese fonts that they give away. Try searching for `epkyouka.ttf`. One of the most famous free TrueType fonts comes from Netscape and is called Cyberbit. I have created two Kanji fonts: the Kanji Stroke Order Font [6] and the (unmaintained) Choumei font [6], which is simply the Kanji Stroke Order Font with the stroke numbers removed.

If you are learning kanji then it's worth looking at the kyoukasho 教科書(きょうかしょ) fonts. These are the Japanese equivalent of the Western 'Schoolbook' fonts and are designed explicitly for teaching Japanese. A Japanese calligraphy teacher recommended the commercial Iwata Gakusen Kyoukasho (Gーイワタ中太教科書体) font to me. This font costs about ¥12,000 (appr. 2700 CZK; 105 EUR).

7.2. Installing into pTEX

First install the font in Microsoft Windows. Let's call it `epkyouka.ttf`. You should have a file called `c:\windows\fonts\epkyouka.ttf` on your system. In your local `texmf` tree (such as `c:\work\texmf}` copy
`fonts\tfm\dvips\rml.tfm` to `fonts\tfm\dvips\epk.tfm`
copy
`fonts\tfm\ptex\min10.tfm` to `fonts\tfm\ptex\schoolbook.tfm`
and copy
`fonts\vf\ptex\min10.vf` to `fonts\vf\ptex\schoolbook.vf`.

Open `fonts\vf\ptex\schoolbook.vf` in a text editor and change the three letters `rml` to `epk`. What we've done here is to create the `.tfm` files that pTEX uses for a new font and to create a virtual font so we can view it.

Run `mktexlsr` (or equivalent) so that KPSE knows about your new files. You should now be able to run a file like the following through pTEX.

```
\font\schlbk=schoolbook at 12pt
\tenmin 鮭は魚です。
\schlbk 鮭は魚です。
\bye
```

7.3. dvipdfmx

These metric files are not much use unless you can view the output. To do this you must tell `dvipdfmx` about the new font. The best way to do this is to modify `msembed.map`. Copy it into your local `texmf` tree and add the following line

```
epk  H  :0:epkyouka
```

Run `mktexlsr` again and then do

```
dvipdfmx -f msembed.map test.dvi
```

You should get the PDF file. When you open it with Adobe Reader, the document properties should tell you that the Microsoft Mincho and Epson Kyoukasho fonts are both present in the document.

There are some advanced options you can put in the `msembed.map` file. For example

```
epk  H  :0:epkyouka,Bold
```

gives you a bold version of the font. BoldItalic and Italic are also valid keywords here. My experiments indicate that this doesn't work very well; the fonts appear in the modified form in Adobe Reader but do not come out on the printer. This is no great loss; ransom-note typography is best left alone. Some TrueType fonts contain multiple versions of themselves in the same file. You can access the different versions by changing the number between the colons:

```
xyz  H  :1:complexfont
```

The `H` refers to whether the font is for horizontal or vertical typesetting. I haven't tried installing a vertical version of a font.

7.4. dvips

It is also possible to use TrueType kanji fonts with `dvipsv` and `dvipsk`.

For `dvipsv`, locate `psfontsv.map`, take a copy into your local `texmf` tree and add the following line.

```
ekn       r-epson-kyoukasho                <‘r-epson-kyoukasho
```

Next locate `vfontcap` in the main `texmf` tree, save off a backup and modify it where it is (KPSE doesn't seem to find it if I put it in my local `texmf` tree) to add the following lines.

```
r-epson-kyoukasho:\
:ft=freetype:\
:ff=c\:/windows/fonts/epkyouka.ttf:
```

Run `mktexlsr` and then `dvipsv test.dvi` and you should get a PostScript version of your document.

If you want a PostScript file that does not embed the kanji font then you can also configure `dvipsk` to use a new TrueType font. First update `psfonts.map` to include the line

```
ekn          epson-kyoukasho-H
```

Then update the file `cidfmap` in your GhostScript installation (try looking for `c:\gs\gs8.51\lib\cidfmap`) to include the following line (split into two lines here so it will fit on the page)

```
/epson-kyoukasho << /FileType /TrueType /SubfontID 0 /CSI
    [(Japan1) 3] /Path (C:/WINDOWS/fonts/epkyouka.ttf) >> ;
```

I find that documents created this way take a long time to open in GhostView. Furthermore, one document with dozens of different fonts in it that I tried crashed GhostScript. Therefore I can't recommend this method.

8. Vertical typesetting

Traditionally Japanese is written from top to bottom and from right to left. One of the strengths of pTEX is that it has native support for this format.

To typeset a document vertically in Plain pTEX use `\tate` at the start of the document and declare the font you want to use (`\tentmin` is the only one I know works):

```
\tate\tentmin
私はイギリス人です。
\bye
```

Then convert the `.dvi` file to a landscape PDF as follows

```
dvipdfmx -l sample
```

In Japanese *tate* 縦(たて) means *vertical*.

As you would expect from Plain TEX, the rest of the document formatting needs work before you can use this method for a real document. However, using pLATEX you get everything done for you. All you have to do is change the

```
\documentclass{jarticle}
```

in the preamble to

```
\documentclass{tarticle}
```

For example

```
\documentclass{tarticle}
\begin{document}
鯨は魚ではありません。
\end{document}
```

gives you something like Figure 1.

Again you need to convert the `.dvi` file to a landscape PDF as follows

```
dvipdfmx -l sample
```

That's right, you can take your horizontally-orientated document and convert it to vertical format by changing just one character.

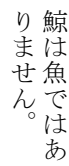


Figure 1: Vertical Japanese.

9. Ruby

Ruby is the typographical name for furigana. These are small kana characters written near a kanji to clarify its reading, like this: 魚(さかな).

9.1. Ruby in pLaTeX

To use ruby in your pLaTeX document include

```
\usepackage{sfkanbun}
\usepackage{furikana}
```

in your pLaTeX document's preamble. There are two macros and a variety of options. The following code yields the example in Figure 2.

```
\documentclass[12pt]{tarticle}
\usepackage{sfkanbun}\usepackage{furikana}
\begin{document}
\kana{私新聞魚犬}{わたししんぶんさかないぬ}\par
\kana[0]{私}{わたし}\kana[0]{新聞}{しんぶん}\kana[0]{魚}{さかな}%
\kana[0]{犬}{いぬ}\par
\kana[1]{私}{わたし}\kana[1]{新聞}{しんぶん}\kana[1]{魚}{さかな}%
\kana[1]{犬}{いぬ}\par
\kana[2]{私}{わたし}\kana[2]{新聞}{しんぶん}\kana[2]{魚}{さかな}%
\kana[2]{犬}{いぬ}\par
\kana[3]{私}{わたし}\kana[3]{新聞}{しんぶん}\kana[3]{魚}{さかな}%
\kana[3]{犬}{いぬ}\par
\kana[4]{私}{わたし}\kana[4]{新聞}{しんぶん}\kana[4]{魚}{さかな}%
\kana[4]{犬}{いぬ}\par
\Kana{私, 新, 聞, 魚, 犬}{わたし, しんぶん, さかな, いぬ}\par
\Kana[0]{私, 新, 聞, 魚, 犬}{わたし, しんぶん, さかな, いぬ}\par
\Kana[1]{私, 新, 聞, 魚, 犬}{わたし, しんぶん, さかな, いぬ}\par
\Kana[2]{私, 新, 聞, 魚, 犬}{わたし, しんぶん, さかな, いぬ}\par
\Kana[3]{私, 新, 聞, 魚, 犬}{わたし, しんぶん, さかな, いぬ}\par
\Kana[4]{私, 新, 聞, 魚, 犬}{わたし, しんぶん, さかな, いぬ}\par
\end{document}
```

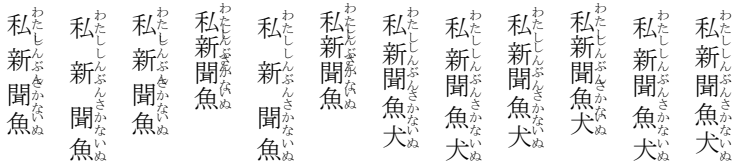


Figure 2: Demonstration of ruby macro syntax.

9.2. Ruby in Plain pTEX

It is possible to modify the file `furikana.sty` to allow you use its ruby macros in Plain pTEX. Here is a summary of what you need to do (I don't recommend this for beginners):

– Copy `furikana.sty` to `plain_furikana.tex`. Do the edits that follow on the latter file;
– Remove the lines that start `\typeout`;
– Remove the paragraph before the line that reads `\let\rubykatuji=\tiny`;
– Change `\@s@sf` to `\asasf` throughout;
– Remove the `\kana` macro;
– Change the `\k@na@` macro so that it always takes five parameters and rename it to `\kana`. Parameter `#1` is the ruby style, parameters `#4` and `#5` define the font and size used for the ruby. Add

```
\font\tiny=#4 at #5pt\def\@rubykatuji{\tiny}\def\rubykatuji{\tiny}
```

to the macro after the line `\xkanjiskip=0pt`; and
– Remove `\endinput` at the end of the file.

You can then use ruby in a Plain pTEX document by adding

```
\input plain_furikana.tex
```

near the start of the document and entering things like

```
\kana{1}{私}{わたし}{epkyo}{6}
```

This works in both horizontal and vertical modes. You may want to define your own two-parameter macro that sets the other parameters automatically. `\Kana` remains a pLATEX-only macro.

9.3. Ruby in Plain non-pTEX

The Plain pTEX version of the `furikana.sty` macro described in the previous section requires that you use pTEX. The following macro allows users to use ruby in any version of TEX. An obvious application is to typeset ruby when using XETEX. The `furikana.sty` macro does not work in pLATEX `\section{}` headings, so this macro could also be useful when typesetting with pLATEX.

However, this macro results in corrupted PDF bookmarks when combined with the hyperref package. This macro could even be used with jTEX or Plain TEX (fonts permitting).

```
\font\tinyjapanese=min10 at 6pt%
\def\furigana#1#2{\leavevmode%
\setbox0=\hbox{#1}\setbox1=\hbox{\tinyjapanese#2}%
\ifdim\wd0>\wd1\dimen0=\wd0\else\dimen0=\wd1\fi%
\hbox{\vbox{\hbox to\dimen0{\tinyjapanese\hfil#2\hfil}%
\nointerlineskip\hbox to\dimen0{\hfil#1\hfil}}}}
```

The furikana.sty macros produce more elegant output and provide more formatting flexibility than this macro:

No ruby: 日本で働いた
furikana.sty: 日本(にほん)で働(はたら)いた
\furigana{}{}: 日本(にほん)で 働(はたら) いた

You are likely to want to tweak the macro described in this section to suit your particular application.

10. Circled characters

Japanese text (especially reference works) sometimes makes use of characters with circles around them. The macros \psCirclebox and \pscirclebox provided by the PSTricks macro package work well for producing Japanese characters with circles around them. The disadvantage is that you then have to produce your document as a PostScript file rather than a PDF file. The best way to handle this is to use GhostScript to convert your document to PDF like this:

```
gswin32c -sDEVICE=pdfwrite -sOutputFile=circle.pdf
                           -dNOPAUSE -dBATCH -q circle.ps
```

The output looks bad in Adobe Reader but looks fine when you print it.

11. dvipdfmx and PSTricks effects

The fancy text colouring and rotation dvipdfm(x) and PSTricks provide work just as well with Japanese text as they do with Western text. The following sample code produces the garish example in Figure 3.

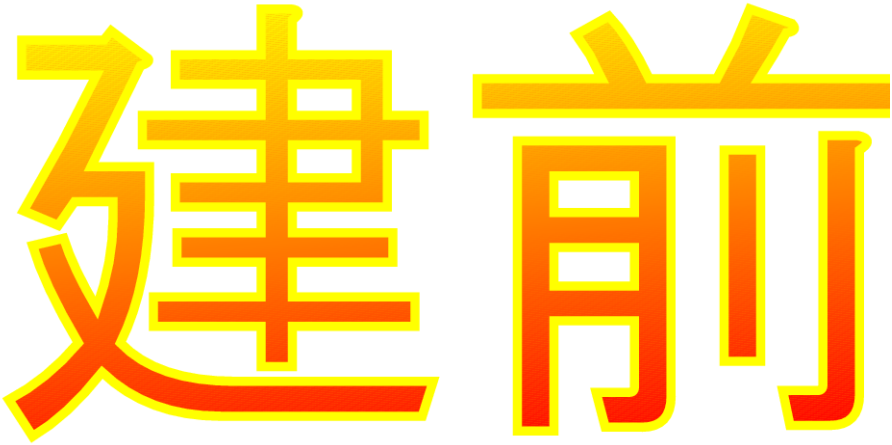


Figure 3: Using PSTricks on Kanji.

```
\documentclass[11pt]{article}
\usepackage{pstricks}
\usepackage{pst-grad, pst-plot, pst-text, pst-char}
\begin{document}
\begin{pspicture}(0,-1)(8,2)
\pscharpath[linecolor=Yellow,fillstyle=gradient,%
gradbegin=Yellow,gradend=Red,gradmidpoint=1,gradangle=5]%
{\font\tmp=goth10 at 1.5cm\tmp 建前}
\end{pspicture}
\end{document}
```

12. Mixing vertical and horizontal text

This might seem like an exotic requirement and I have to admit I'm unlikely to use it myself. However, Japanese newspapers often mix vertical and horizontal text. You can do this in pTEX using `minipage`:

私は魚が好きです。私は魚が好きです。私は魚が好きです。私は魚が好きです。私は魚が好きです。私は魚が好きです。

鯨は魚ではありません。鯨は魚ではありません。
鯨は魚ではありません。鯨は魚ではありません。

私は魚が好きです。私は魚が好きです。私は魚が好きです。私は魚が好きです。私は魚が好きです。私は魚が好きです。

Figure 4: Vertical Japanese within horizontal.

```
\documentclass{jarticle}
\usepackage{plext}
\begin{document}
私は魚が好きです。私は魚が好きです。私は魚が好きです。
私は魚が好きです。私は魚が好きです。私は魚が好きです。

\begin{minipage}<t>{16zw}
鯨は魚ではありません。鯨は魚ではありません。
鯨は魚ではありません。鯨は魚ではありません。
\end{minipage}

私は魚が好きです。私は魚が好きです。私は魚が好きです。
私は魚が好きです。私は魚が好きです。私は魚が好きです。
\end{document}
```

Which yields something like Figure 4. The full syntax for `minipage` is described in the `platex.tex` format file as follows:

```
\begin{minipage}<dir>[pos]{width}...\end{minipage}
  dir: t ... force tate mode.
       y ... force yoko mode.
       z ... rotate 90 degree (ignored at yoko mode).
```

Yoko means horizontal mode. This syntax encourages the following test:

```
\documentclass{tarticle}
\usepackage{plext}
\begin{document}
私は魚が好きです。私は魚が好きです。私は魚が好きです。
私は魚が好きです。私は魚が好きです。私は魚が好きです。

\begin{minipage}<y>{16zw}
鯨は魚ではありません。鯨は魚ではありません。
鯨は魚ではありません。鯨は魚ではありません。
\end{minipage}

私は魚が好きです。私は魚が好きです。私は魚が好きです。
私は魚が好きです。私は魚が好きです。私は魚が好きです。

\begin{minipage}<z>{16zw}
鯨は魚ではありません。鯨は魚ではありません。
鯨は魚ではありません。鯨は魚ではありません。
\end{minipage}

私は魚が好きです。私は魚が好きです。私は魚が好きです。
私は魚が好きです。私は魚が好きです。私は魚が好きです。
\end{document}
```

Which yields something like Figure 5. Note that a `zw` is a new unit of width introduced by pTEX.

13. Kanji font selection in LATEX

One way to make the default kanji text font in a LATEX document be a new one is brute force: `\font\normal=epkyo at 13pt \normal`. However, LATEX has a system for defining font sizes consistently and it is more architectural to use that. I think what follows is pretty much the same as you'd do for a new Western font except that `\rmdefault` is replaced by `\mcdefault`.

What you need to do is copy `jy1mc.fd` and `jt1mc.fd` from your installation pTEX tree to your local `texmf` tree (I put mine in `ptex\platex\base`) and rename them as `jy1ep.fd` and `jt1ep.fd`, where `ep` represents your new font.

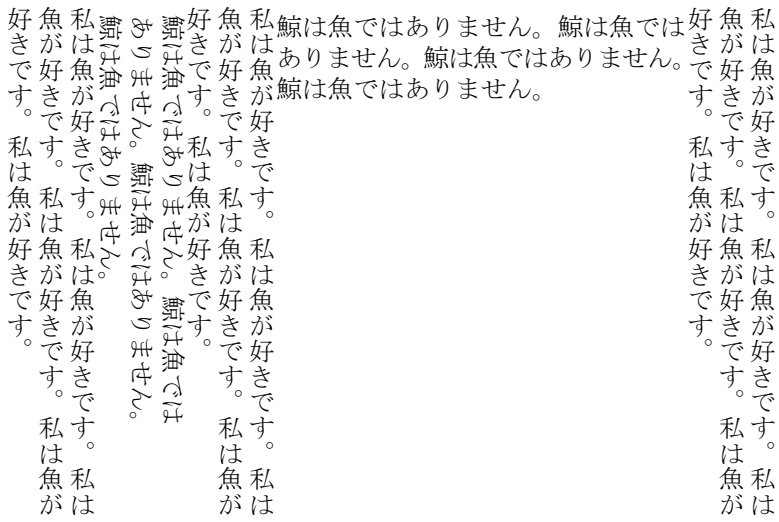


Figure 5: Horizontal Japanese within vertical.

Change all the instances of `mc` in the files to `ep`. Then you need to change the following code

```
\DeclareFontShape{JY1}{mc}{m}{n}{<5> <6> ... <10> sgen*min
    <10.95><12><14.4><17.28><20.74><24.88> min10
    <-> min10
    }{}
```

to

```
\DeclareFontShape{JY1}{ep}{m}{n}{<-> s*[1.3] epkyo}{}
```

Obviously you should replace `ep` here with your own font's name. The `[1.3]` is the magnification over the design size of the font (deliberately set high here at 130% because the Epson Kyoukasho font is rather small). The `s*` means, 'I don't care what size this ends up being so LaTeX should not warn me when it sees sizes it does not recognize'. The `epkyo` is the name of your font, which should exist as a `.tfm` file. All those numbers in pointy brackets and the `sgen*min` in the original are to do with fized sizes of the font and appear to be unneccesary in modern installations.

Once you've done this, create a file called `myfont.sty` in your local `texmf` tree and put something like the following code in it.

```
\ProvidesPackage{epkyo}
\renewcommand{\mcdefault}{ep}
\endinput
```

Then run `mktexlsr`, put `\usepackage{epkyo}` in the preamble to your document and you should get your new font. If you want to change your gothic font rather than your mincho font then the above should work substituing *gt* for *mc* throughout. I haven't tried it.

14. Missing font shapes

You may find that pLaTeX complains about missing font shapes when compiling documents. The messages are benign because the pLaTeX font code sensibly substitutes other fonts in their place. You can prevent these warnings by adding the missing shapes to the files `jt1ep.fd` and `jy1ep.fd` described in Section 13 so that they look like this:

```
\DeclareFontShape{JT1}{ep}{m}{n}{<->s*[1.3] epkyo}{}
\DeclareFontShape{JT1}{ep}{m}{sc}{<->ssub*ep/m/n}{}
\DeclareFontShape{JT1}{ep}{bx}{n}{<->ssub*gt/m/n}{}
```

and this:

```
\DeclareFontShape{JY1}{ep}{m}{n}{<-> s*[1.3] epkyo}{}
\DeclareFontShape{JY1}{ep}{m}{sc}{<-> ssub*ep/m/n}{}
\DeclareFontShape{JY1}{ep}{bx}{n}{<-> ssub*gt/m/n}{}
```

It's the middle line in each of these that is new. pLaTeX also complains about the gothic kanji font shapes that one might have thought it would have defined itself (the file that does this seems to be `plfonts.dtx`). The `myfont.sty` file is a convenient (albeit unarchitectural) place to fix this up. You can do so by adding the following two lines so it looks something like this:

```
\ProvidesPackage{epkyo}
\renewcommand{\mcdefault}{ep}
\DeclareFontShape{JT1}{gt}{m}{it}{<->ssub*gt/m/n}{}
\DeclareFontShape{JY1}{gt}{m}{it}{<->ssub*gt/m/n}{}
\endinput
```

If you are not setting up your own fonts and just want to suppress warnings in a document compiled using the default fonts then another approach (the one used in this document) is to define a new style file that declares the font shapes that pLaTeX is complaining about. Mine is called `noswarn.sty`:

```
\ProvidesPackage{noswarn}
\DeclareFontShape{JT1}{gt}{m}{it}{<->ssub*gt/m/n}{}
\DeclareFontShape{JY1}{gt}{m}{it}{<->ssub*gt/m/n}{}
\DeclareFontShape{JT1}{mc}{m}{sc}{<->ssub*gt/m/n}{}
\DeclareFontShape{JY1}{mc}{m}{sc}{<->ssub*gt/m/n}{}
\endinput
```

Put `\usepackage{noswarn}` in your document's preamble and run `mktexlsr`. Next time you compile your documents the warnings should have stopped. If you still get some warnings then try adding the shapes that pLaTeX is complaining about to the style file.

15. Underlined Japanese text

Though underlining text is considered bad typography, it is easy to do in pTeX if you are in horizontal mode. Just use

`{$\underline{\hbox{鯨を食べないで下さい}}$}`

to get 鯨を食べないで下さい.

I have seen pLaTeX packages that do underlining but all the documentation was in Japanese. I think they handle vertical format text.

16. Warichu

Warichu or 割り注 is a form of inserted notes within Japanese text. The characters are half the height of the main text. This facility is available in pLaTeX using the `warichu.sty` package. Figure 6 shows an example of text that includes *warichu*. I generated the first block of text in Figure 6 from the code:

田中先生は`\warichu{本}{教科書だけです。小説がきらいです。}`が好きです。

In general, the syntax is

`\warichu{X}{Y}Z`

where `X` denotes the last ordinary character before the notes, `Y` denotes the notes themselves and `Z` denotes the characters that come after the notes.

This typographical device is more commonly used in vertical format. In *tate* mode one uses the macro `\twarichu` rather than `\warichu`. I generated the second block of text in Figure 6 using the above code with `\twarichu` substituted for `\warichu`.

The `warichu.sty` macros `\warigaki` and `\twarigaki` are similar to the macros `\warichu` and `\twarichu` except that they only take one parameter and they omit parentheses from the result. The third and fourth blocks of text in Figure 6 give examples. I generated the third block of text in Figure 6 from the following code:

田中先生は本`\warigaki{教科書だけです。小説がきらいです。}`が好きです。

There are two limitations to the `warichu.sty` package. First, it does not support line-breaking for *warichu*; all your notes have to be on a single line. Only painstaking manual tweaking could produce multi-line *warichu*. Second, by default the font used for the *warichu* text is simply a scaled-down version of

the main text font. Ideally the weight of the strokes in the *warichu* text would match that in the main text. A good knowledge of LaTeX and pLaTeX font management would probably allow an expert user to get around this problem. This could be done by switching to a weightier version of the main text font for the *warichu* text. がんばって。

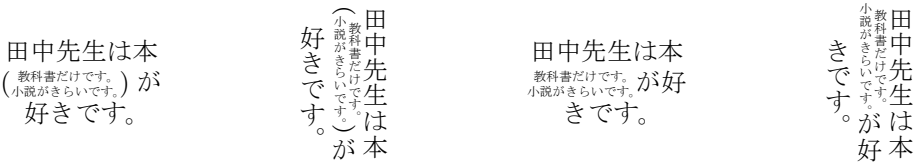


Figure 6: Examples of the use of the warichu macros. From left to right: `\warichu`, `\twarichu`, `\warigaki`, `\twarigaki`.

References

[1] W32TeX download page in English: `http://www.w32tex.org/`

[2] Lunde, Ken. *CJKV Information Processing.* Second Edition, O'Reilly Media, 2009. ISBN 978-0-596-51447-1.

[3] `http://www.physics.ucla.edu/~grosenth/jwpce.html`

[4] Saito, Yasuki. *Report on jTeX: A Japanese TeX.* In *TUGboat,* Volume 8, Number 2, pp. 103–116, July 1987. ISSN 0896-3207. Available at `http://tug.org/TUGboat/Articles/tb08-2/tb18saito.pdf`

[5] `http://oku.edu.mie-u.ac.jp/~okumura/texfaq/japanese/ptex.html`

[6] `http://nihilist.org.uk/`

Summary: Typesetting Japanese with pTeX

pTeX is a TeX-like typesetting system that is specifically designed for typesetting Japanese and is widely used in Japan. This article describes how to acquire, set up and use a pTeX system in practice, with an emphasis on font management. It also provides basic background information on Japanese text processing and alternatives to pTeX.

Keywords: pTeX, Japanese, kanji, kana, Unicode, ruby, CJKV.

Timothy Eyre, mail@nihilist.org.uk
CSTUG c/o FEL ČVUT, Technická 2
Prague, CZ-166 27, Czech Republic

Pravidla sazby japonštiny v XeTeXu

Kazuomi Kuniyoshi

Abstrakt

Tato krátká zpráva komentuje důvod vzniku balíčku genzi, který nastavuje a upravuje pravidla sazby japonštiny v XeTeXu. Komentáře, balíček i několik ukázek si lze stáhnout z webové stránky autora, viz `http://kuniyoshi.fastmail.fm/xetex/`. Balíček zatím není součástí CTAN.ORG.

Klíčová slova: ČJK, XeTeX, japonština, balíček genzi.

doi: 10.5300/2010-3/174

In principle, XeTeX lets you mix as many writing systems as you wish in a document. With an appropriate OpenType font being installed in the operating system, you can freely use all the letters which the font contains. Moreover, you can put letters from different fonts onto a document at the same time. This is a marvellous feature of XeTeX.

I am a multilingual writer, or to be more precise, multi-*script*-al writer, who loves the power of TeX. The writing systems I often mix are Japanese and Latin. It is no big deal to do so in today's word processors, but fine-tuning the balance of these letters, let alone producing an overall beautiful layout, is not an easy task in such an environment. So the advent of XeTeX was an exciting news for me. Yet, on trying it for the first time, I immediately faced a problem: XeTeX does not know the formatting rules for Japanese writing well.

Some websites give an impression that XeTeX can produce good Japanese documents only by touching `\XeTeXlinebreaklocale` and `\XeTeXlinebreakskip`. This is simply false; everyone who knows how to write in Japanese should be able to see quickly that this is like a paradise (mis)represented in a summer travel brochure. Indeed, XeTeX can print Japanese letters; however it ignores the formatting rules to the extent that it cannot be used in serious publications. It seems to me that there is some confusion over printing letters and formatting them among developers and users.

In the world of computing, there is a convenient word 'CJK'. This word is convenient as long as letters are concerned, but it turns out to be inconvenient as soon as document formatting becomes the topic of discussion. Perhaps it is right to say that 'C', 'J' and 'K' are similar enough to be treated together for engineering purposes. Chinese and Japanese share a lot of letters despite subtle differences, and it is unnecessary to come up with different engineering methods to deal with them. I totally agree with this.

XeTeX is, however, not only a tool to print letters onto PDF files (and sometimes to sheets of paper), but also a powerful formatting tool. In any event, that is why people are drawn into TeX most of the time. So it is important to separate printing letters and formatting them; for printing letters is an engineering issue, but formatting them is a cultural issue. And, 'C', 'J' and 'K' do differ greatly when it comes to their formatting rules.

The current state of XeTeX is almost perfect on the printing side. There is wonderful freedom in picking up any of 15,444 letters from an Adobe-Japan1-4 font in XeTeX. But whenever I write a document in Japanese or (Classical) Chinese, XeTeX does not automatically format text correctly. This is an area where more work should be done, and it is interestingly an interdisciplinary one spanning software engineering, linguistics and literature.

The genzi package I wrote is a preliminary attempt to bridge this gap.

`http://kuniyoshi.fastmail.fm/xetex/`

Those who use other writing systems in XeTeX, especially very 'exotic' ones, may also be feeling that cultural differences should be considered more. As TeX, which was exclusively American in the beginning, has gone out of the Euro-American world and stepped into the dangerous 'polyscript' terrain of East and Southeast Asia, more humanism should melt into its engineering than ever before.

Summary: Japanese formatting rules for XeTeX

This is a two-page report informing about the reasons and the existence of the genzi package which sets Japanese formatting rules for XeTeX. The package, samples and more comments can be viewed and downloaded from the author's website, see `http://kuniyoshi.fastmail.fm/xetex/`.

Keywords: CJK, XeTeX, Japanese, genzi package.

Kazuomi Kuniyoshi, Brussels, Belgium
kuniyoshi@fastmail.fm
CSTUG c/o FEL ČVUT, Technická 2
Prague, CZ-166 27, Czech Republic

Kazuraki: tutoriál k japonskému OTF písmu

Ken Lunde

Abstrakt

Článek představuje klíčové technické kroky tvorby proporcionálního japonského písma nazvaného Kazuraki. Toto písmo je oproštěno od konvenčních omezení přetrvávajících v Japonsku po desetiletí. Design je inspirován kaligrafií z dvanáctého století umělce, spisovatele a jednoho z největších básníků v japonské historii Fujiwara-no-Teika. Písmo bylo poprvé oficiálně představeno 7. prosince 2009 v předváděcí místnosti firmy Adobe (North America Type Showroom).

Písmo by se mohlo stát inspirací a vzorem pro další tvůrce a písmolijny, především k vyjádření individuálního a osobitého stylu, silných výrazových prostředků a decentnějšího stylizování.

V ukázkách a přílohách tohoto článku je konkrétně použito OpenType písmo KazurakiStd-Light. Užité nástroje *makeotf*, *tx*, *mergeFonts*, *rotateFont* jsou součástí volně dostupné softwarové kolekce AFDKO (Adobe® Font Development Kit for OpenType), stáhnutelné z webových stránek `http://www.adobe.com/devnet/opentype/afdko/`.

Klíčová slova: písmo Kazuraki, proporcionální glyfy, japonské písmo, písmo OpenType, AFDKO, Adobe, *makeotf*, *tx*, *mergeFonts*, *rotateFont*.

doi: 10.5300/2010-3/176

1. Introduction

This tutorial is designed to guide Japanese font developers in building special-purpose OpenType® Japanese fonts, using KazurakiStd-Light (to be referred to as simply Kazuraki® from this point forward) as an example of how to build a genuinely and completely proportional Japanese font. The techniques, tools, and control files that are described or referenced in, or attached to, this document are tightly coupled to tools that are included in AFDKO (Adobe® Font Development Kit for OpenType) Version 2.0 or greater, which is available, at no charge, at the following URL: `http://www.adobe.com/devnet/opentype/afdko/`

Please pay attention to Section 6 on page 186 of this article, which includes information—relevant as of this writing—about compatibility with applications.

If you have any questions regarding the content of this article, please do not hesitate to contact its author, Ken Lunde (`lunde@adobe.com`).

2. Kazuraki Design Motivations

With the exception of the vertical-only hiragana ligatures, all glyphs in Kazuraki have corresponding horizontal and vertical forms. That is, for each character, there are two glyphs in the font. The glyphs themselves are the same, but their advances and default positioning along both X- and Y-axes are different. All glyphs needed to be replicated for vertical use, due to limitations in the ability to shift glyphs in both X- and Y-axis directions in the OpenType 'vmtx' table, coupled with the strong desire to make expected behavior the default—without an application depending on, or activating, any GSUB or GPOS features.

The glyph complement includes glyphs for all standard kana characters, but includes a limited set of kanji, 1,082 to be exact, suitable for creating Japanese greeting cards, menus, and other specialized uses. Kazuraki also includes a basic set of proportional Latin glyphs to aid in keyboard input. At a minimum, we recommend including glyphs that correspond to ASCII (U+0020 through U+007E). While Kazuraki is fully-functional in this limited context, it also serves as an example for building comparable fonts.

2.1. Genuine Proportional Glyphs

All horizontal/vertical glyph pairs in Kazuraki are the same, with the exception of small kana, some punctuation, and parenthetical symbols, which require different glyphs for horizontal and vertical use in conventional Japanese fonts, due to their position or orientation. Post-processing of the glyph data is used to derive the vertical versions, which are positioned along the Y-axis differently, and have different metrics.

2.2. Vertical-Only Hiragana Ligatures

Kazuraki includes a small number of vertical-only hiragana ligatures, fifty to be exact. Three are four-character ligatures, twelve are three-character ligatures, and the remaining thirty-five are two-character ones. These are activated through the use of the 'liga' GSUB feature. The advantage of using 'liga' is that most applications that perform an adequate level of typesetting also tend to automatically invoke this GSUB feature, or at least allow users to activate it through a standard UI. This allows vertical-only hiragana ligatures to be used by default in many applications.

2.3. CID- Versus Name-Keyed Structure

Kazuraki was built as a CID-keyed OpenType font. Its 'CFF' table is built from a CIDFont resource. Although Kazuraki could have been built as a name-keyed font, CID-keyed fonts have advantages for Japanese fonts. The primary

advantage is that a CID-keyed structure supports multiple hint dictionaries. Each hint dictionary ideally covers glyphs for specific glyph classes, and each hint dictionary can have its own hinting parameters. Using multiple hint dictionaries thus offers significant rendering advantages.

3. OpenType Table Settings & Overrides

Many of the OpenType tables require special settings for Kazuraki. This section describes the special settings, one table at a time, along with information that demonstrates how the table overrides are specified in the "features" file as used as input for AFDKO's *makeotf* tool.

3.1. BASE

There is no special treatment necessary for the 'BASE' table, other than the usual ICF (Ideographic Character Face) and baseline values that should be normally specified. It is very important to include a 'BASE' table in all OpenType fonts. The following is the 'BASE' table override in the "features" file of Kazuraki:

OpenType fonts. The following is the 'BASE' table override in the "features" file of Kazuraki:

```
table BASE {
  HorizAxis.BaseTagList              icfb icft ideo romn;
  HorizAxis.BaseScriptList DFLT ideo -117 877  -120 0,
                           hani ideo -117 877  -120 0,
                           kana ideo -117 877  -120 0,
                           latn ideo -117 877  -120 0;
  VertAxis.BaseTagList               icfb icft ideo romn;
  VertAxis.BaseScriptList  DFLT ideo 3    997  0    120,
                           hani ideo 3    997  0    120,
                           kana ideo 3    997  0    120,
                           latn ideo 3    997  0    120;
} BASE;
```

Note that only the 'DFLT', 'hani', 'kana', and 'latn' scripts are declared in the 'BASE' table override, based on the glyph complement of Kazuraki.

3.2. CFF

The original source data for Kazuraki is a name-keyed OpenType font containing exactly 1,450 glyphs, each with 1000-unit set widths. The same source font also contains 'palt' and 'vpal' GPOS features that specify the desired glyph metrics (horizontal and vertical set widths, and X- and Y-axis shifting values,

respectively), and these GPOS features are used as the source of the default metrics for the horizontal and vertical glyphs in the final font, which is an Adobe-Identity-0 CID-keyed OpenType font containing 2,973 glyphs (CIDs 0 through 2972).

Three AFDKO tools are used to process the original set of 1,450 glyphs: *tx*, *mergeFonts*, and *rotateFont*.

The first tool, *tx*, simply extracts the 'CFF' table from the source OpenType font, and converts it into a name-keyed Type 1 font, using the following command line:

```
% tx -t1 Kazuraki_2.5.otf > font.pfa
```

The second tool, *mergeFonts*, is used to convert the glyph names into CIDs, and to simultaneously synthesize the vertical glyphs from the original horizontal versions, using the following command line:

```
% mergeFonts -cid cidfontinfo cidfont.raw
    h.map font.pfa v.map font.pfa
```

The end result is an Adobe-Identity-0 CIDFont resource, named "cidfont.raw," containing 2,673 glyphs in a single hint dictionary.

The third tool, *rotateFont*, serves to set the widths for the horizontal glyphs, and to also shift them along the X-axis. The set width and X-axis shifting values are in the 'palt' GPOS feature of the source name-keyed OpenType font. The following command line is used:

```
% rotateFont -t1 -rtf shift.map cidfont.raw cidfont-prop.raw
```

The end result is an Adobe-Identity-0 CIDFont resource, named "cidfont-prop.raw," containing 2,673 glyphs, the horizontal versions of which now have proportional metrics. 300 proportional Latin glyphs are added to bring the glyph complement up to 2,973 glyphs.

As an example, let us explore the treatment of the horizontal glyph for the hiragana character "shi" (し). The glyph in the original name-keyed font is named "CID864" (named after CID+864 of the Adobe-Japan1-*x* character collection). Its 'palt' GPOS feature settings were as follows in the source name-keyed OpenType font:

```
position \CID864 <-223 0 -485 0>;
```

After processing by the *mergeFonts* tool, this (horizontal) glyph becomes CID+224. The 'palt' data shown above is used to generate the following *rotateFont* directive for the "shift.map" mapping file:

```
224 224 515 -223 0
```

The calculation is simple: the value "−223" is used as-is as the X-axis shifting value, and the default set-width value of 1000 becomes 515 after the value "−485" is added to it (a subtraction operation).

The set widths and Y-axis shifting for the vertical glyphs are specified in a 'vmtx' table override definition that is inserted into the "features" file. The

handling of vertical glyphs, in terms of specifying their set widths and Y-axis positions, is covered later in this document.

Once these tools have been run, and the glyphs are assigned to CIDs, and the horizontal glyphs have been set to their default set widths and X-axis positions (that is, made proportional), the CIDFont is then hinted as usual. The process of hinting also involves creating multiple hint dictionaries, ideally only one for each glyph class.

Note: The process of establishing multiple hint dictionaries in a CIDFont requires files and tools that are not included in AFDKO*, and their description is intentionally (and appropriately) omitted from this article. However, mergeFonts techniques described in Adobe Tech Note #5900 ("*AFDKO *Version 2.0 Tutorial: mergeFonts, rotateFont & autohint"), which is among the documentation bundled with* AFDKO*, can be used to establish multiple hint dictionaries. Multiple mergeFonts mapping files, in which the first line each each file names a hint dictionary, is the appropriate technique. And, multiple mergeFonts mapping files can serve to specify glyphs for single hint dictionaries. In fact, the proprietary tool that was used to establish multiple hint dictionaries for Kazuraki uses mergeFonts to perform this task.*

The Special-Purpose Adobe-Identity-0 Character Collection

Because the glyph complement of Kazuraki does not adhere to the Adobe-Japan1-6 character collection, and because it makes little sense to extend Adobe-Japan1-6 to accommodate such special-purpose fonts, the special-purpose Adobe-Identity-0 character collection is advertised in the CFF. Although "Adobe-Identity-0" does not explicitly specify that Kazuraki is a Japanese font, other table settings, along with proven heuristics, are used to make clear the fact that it is a Japanese font.

In essence, the advantage of using the Adobe-Identity-0 character collection is that there are no preconceived notions of language or script, making it possible to build CIDFonts based on dynamic glyph sets, much like TrueType and name-keyed OpenType fonts. The technique of using the Adobe-Identity-0 character collection should not be used to build general-purpose OpenType Japanese fonts. The Adobe-Japan1-*x* character collection should be used instead.

File Size Issues

Because the horizontal/vertical glyph pairs are identical, in terms of their outlines, the subroutinization ability of AFDKO's *makeotf* tool makes the resulting CFF table only slightly larger than that of the original source data, which contained roughly half the number of glyphs. The subroutinized 'CFF' table thus became approximately fifty percent the size of the unsubroutinized version.

Hinting Issues
Hinting, in terms of stem widths, is applied as usual for Kazuraki. Alignment zones, however, are another matter. The hint dictionaries for non-Latin glyph classes, such as kana and kanji, typically use the following `/BlueValues` array:

```
/BlueValues [-250 -250 1100 1100] def
```

However, due to the larger (taller) than usual bounding boxes of the vertical-only hiragana ligatures, the hint dictionary for the kana glyphs require different values, in order to ensure that there are no alignment zones in contact with its glyphs. The "Kana" hint dictionary of Kazuraki uses the following `/BlueValues` array:

```
/BlueValues [-1250 -1250 2000 2000] def
```

Furthermore, the "Dingbats" and "Kanji" hint dictionaries of Kazuraki uses the same `/BlueValues` array, due to the extent to which the shapes of their glyphs extend above and below the 1000×1000 em-box.

The following is the `/FontBBox` for Kazuraki:

```
/FontBBox {-326 -1179 1573 1939} def
```

It was thus critical to select `/BlueValues` values less than −1179 (the Y-axis low point) and greater than 1939 (the Y-axis high point), especially for the "Kana" hint dictionary.

3.3. GPOS

The only GPOS features that are included in Kazuraki are 'kern' and 'vkrn', for horizontal and vertical kerning, respectively. The 'palt' and 'vpal' GPOS features in the original source data served to drive the production process, to specify the horizontal/vertical set widths and X- and Y-axis shifting values. These GPOS features are not in the final form of the font, because they are not necessary. Their values were used to define the default glyph metrics.

3.4. GSUB

Kazuraki contains only four GSUB features: 'fwid', 'vert', 'vrt2', and 'liga'. Although the conventional ordering of these features is 'liga' followed by 'vert' and 'vrt2', this font's vertical-only hiragana ligatures necessitates a different ordering, specifically 'vert' and 'vrt2' followed by 'liga'. As a general rule, the ordering of GPOS and GSUB features in the "features" file is important, because the same ordering is reflected in the 'GPOS' and 'GSUB' tables that are generated by AFDKO's *makeotf* tool.

The 'vert' GSUB feature substitutes the horizontal forms with their vertical versions. This feature covers the majority of the font. Once the 'vert' GSUB feature has been applied, the vertical-only hiragana ligatures can then be applied via the 'liga' GSUB feature.

OpenType-savvy applications that support vertical writing automatically invoke the 'vert' (or 'vrt2', if present) GSUB feature. These same applications also invoke the 'liga' GSUB feature by default, which then serves to activate (or make default) the vertical-only hiragana ligatures.

3.5. OS/2

Because the special-purpose Adobe-Identity-0 character collection is used for Kazuraki, several 'OS/2' table fields must be more carefully specified, such as the OS/2.unicodeRange and OS/2.codePageRange fields. For Kazuraki, these settings are specified in the "features" file as the following 'OS/2' table overrides:

```
XHeight 423;
CapHeight 645;
UnicodeRange 0 1 2 5 31 33 35 36 38 48 49 50 59 62 65 68;
CodePageRange 1252 932;
```

Note that the "XHeight" and "CapHeight" values are set to values that correspond to the proportional Latin glyphs that are in its glyph complement.

The "UnicodeRange" values correspond as follows:

0	Basic Latin
1	Latin-1 Supplement
2	Latin Extended-A
5	Spacing Modifier Letters
31	General Punctuation
33	Currency Symbols
35	Letterlike Symbols
36	Number Forms
38	Mathematical Operators
48	CJK Symbols And Punctuation
49	Hiragana
50	Katakana
59	CJK Unified Ideographs
62	Alphabetic Presentation Forms
65	Vertical Forms
68	Halfwidth And Fullwidth Forms

The "CodePageRange" value of 1252 corresponds to "Latin 1," and 932 corresponds to "JIS/Japan." These 'OS/2' table settings help to explicitly identify Kazuraki as a Japanese font.

3.6. VORG

The 'VORG' table is automatically generated when using AFDKO's *makeotf* tool, and is derived from the settings and overrides of the 'vmtx' table. See the section for the 'vmtx' table for more information on 'vmtx' table settings and overrides.

3.7. cmap

The 'cmap' table for CID-keyed OpenType fonts is built using one or more CMap resources. For Kazuraki, because it is based on the special-purpose Adobe-Identity-0 character collection, special-purpose CMap resources are necessary. Because the vertical glyphs are accessible through the 'vert' and 'vrt2' GSUB features, only the horizontal glyphs are mapped from Unicode code points.

3.8. name

The 'name' table is built as usual, setting English and Japanese strings, as appropriate. The only exception is the name.ID=20 string, which is not necessary due to the special-purpose nature of Kazuraki. The specification of 'name' table strings is performed in the "FontMenuNameDB" and "features" files. Care must be taken to explicitly set Japanese as the script and language, for as many of the strings as possible, as appropriate.

For more information about specifying 'name' table strings for OpenType Japanese fonts, please refer to Adobe Tech Note #5149 ("OpenType-CID/CFF CJK Fonts: 'name' Table Tutorial"), available at the following URL:

`http://www.adobe.com/devnet/font/pdfs/5149.OTFname_Tutorial.pdf`

3.9. vmtx

The 'vmtx' table plays an absolutely crucial role in building fonts such as Kazuraki, because it is in this table that the vertical set widths are specified, along with any Y-axis shifting. Anything specified in the 'vmtx' table becomes default behavior. Thus, OpenType-savvy applications that support vertical writing can use such fonts without modification.

Kazuraki's "features" file contains a very large number of "VertAdvanceY" and "VertOriginY" statements in its 'vmtx' table overrides. Nearly every vertical glyph required treatment by one or both of these 'vmtx' overrides.

As an example, let us explore the treatment of the vertical glyph for the hiragana character "shi" (し). The glyph in the original name-keyed font is named "CID864" (named after CID+864 of the Adobe-Japan1-x character collection). Its 'vpal' GPOS feature settings were as follows:

```
position \CID864 <0 -26 0 331>;
```

After processing by *mergeFonts*, this (vertical) glyph became CID+1682. The 'vpal' data shown above was used to generate the following 'vmtx' table overrides for the "features" file:

```
VertOriginY \1682 906;
VertAdvanceY \1682 1331;
```

The calculation is simple: the value "−26" is subtracted from 880 (a fixed value that represents the top of the em-box) to become 906, which is the new origin, and the value "331" is added to the default 1000-unit width to become 1331.

4. Special Tools

A single special-purpose tool was written, in Perl, to generate all of the control files and data in a single execution. The mapping files that controlled the execution of the *mergeFonts* and *rotateFont* tools were generated by this tool, as was the "features" files containing 'vmtx' table overrides and all GSUB and GPOS feature definitions. The raw data to build the Unicode (UTF-32) CMap resources was also generated by this tool. Due to the large number of glyphs, and the complex relationships between them, it was important to create a tool to do this work, because doing so by hand would have been tedious, and also prone to error.

When writing a comparable tool, I found that it was very useful to maintain a mapping from the glyph names in the source font to the final CIDs. This made generating the raw data for the CMap resource a much easier task. It also made other tasks easier.

5. OpenType Control Files & Data

Once the name- to CID-keyed conversion is complete, the usual control files and data, required by AFDKO's *makeotf* tool, must be generated or supplied. These control files and data are detailed in the following sections.

5.1. CIDFont Resource

Kazuraki's CIDFont resource is constructed as usual, with an appropriate number of hint dictionaries, ideally one for each glyph class, and with Adobe-Identity-0 as its advertised ROS (`/Registry`, `/Ordering`, and `/Supplement`, which are the three entries of the /CIDSystemInfo dictionary). As stated earlier in this document, Kazuraki's CIDFont resource contains 2,973 glyphs, specifically CIDs 0 through 2972. Kazuraki contains exactly six hint dictionaries, named as follows, and with the number of glyphs in each in parentheses:

- KazurakiStd-Light-Dingbats (102 glyphs)
- KazurakiStd-Light-Generic (one glyph)
- KazurakiStd-Light-Kana (406 glyphs)
- KazurakiStd-Light-Kanji (2,164 glyphs)
- KazurakiStd-Light-Proportional (150 glyphs)
- KazurakiStd-Light-ProportionalRot (150 glyphs)

5.2. The "features" File

The "features" file plays an important role, in that overrides to specific tables can be made, and GPOS and GSUB features can be defined. Kazuraki contains two GPOS features, 'kern' and 'vkrn', to specify horizontal and vertical kerning pairs, respectively. Four GSUB features—'fwid', 'vert', 'vrt2', and 'liga'—are also included, whose relative order is important, as described earlier in this document. Lastly, the 'vmtx' table overrides, which are also used to build the 'VORG' table, serve to specify the default vertical metrics.

5.3. The "FontMenuNameDB" File

The English and Japanese menu names that are recorded in the 'name' table of an OpenType font are specified in the "FontMenuNameDB" file. Kazuraki's "FontMenuNameDB" entry is shown below:

```
[KazurakiStd-Light]
   f=3,1,0x411,\304b\3065\3089\304d Std
   s=3,1,0x411,L
   c=3,1,0x411,\304b\3065\3089\304d Std L
   f=1,1,11,\82\a9\82\c3\82\e7\82\ab Std
   s=1,1,11,L
   c=1,1,11,\82\a9\82\c3\82\e7\82\ab Std L
   f=Kazuraki Std
   s=L
   c=Kazuraki Std L
```

It is important to stress that English menu names must be set—in addition to the obvious Japanese menu names—in case such fonts are used in applications whose heuristics may cause a failure to properly use the Japanese menu names.

5.4. CMap Resources

For special-purpose fonts such as Kazuraki, only a Unicode CMap resource is necessary. Even if there are no mappings outside the BMP, a UTF-32 CMap resource is still recommended as input to AFDKO's *makeotf* tool. For Kazuraki, the Unicode CMap resource was named "UniKazurakiStd-UTF32-H" to make

it tightly coupled with the font. This CMap resource is used solely as input to AFDKO's *makeotf* tool, to build the Unicode 'cmap' subtables of the resulting OpenType font.

For more information about building CMap resources, please refer to Adobe Tech Note #5099 ("Building CMap Files for CID-Keyed Fonts"), available at the following URL:

`http://www.adobe.com/devnet/font/pdfs/5099.CMapFiles.pdf`

6. Testing & Compatibility Considerations

Kazuraki works as expected in Adobe InDesign® CS2 and greater. The horizontal and vertical metrics are respected, and proper vertical layout is supported, including the vertical-only hiragana ligatures.

Kazuraki functions in Adobe Illustrator® CS2 and Adobe Photoshop® CS2 with some limitations, specifically that the vertical-only hiragana ligatures do not function, even if the 'liga' GSUB feature is turned on.

In addition, these and other applications may not display Kazuraki's name in Japanese in their font menus. Kazuraki's name may instead display in English, using the English-language menu name strings that are specified in the 'name' table. Kazuraki works very well with CS3 and CS4 applications, specifically InDesign, Illustrator, and Photoshop. In fact, we recommend that CS3 and later applications be used for Kazuraki and comparable fonts.

Due to its unique (and limited) glyph complement, Kazuraki is not recommended for use as a component in these applications' Composite Font functionality.

When developing special-purpose OpenType Japanese fonts, it is prudent to rigorously test the font with a variety of OSes and applications, to include entire document authoring workflows.

7. Glyph Synopsis

The following eleven pages provide a complete glyph synopsis for the 2,973 glyphs of Kazuraki, arranged by CID. The following list provides information about specific CID ranges:

0001–0150	Horizontal glyphs—proportional Latin
0151–1462	Horizontal glyphs—Japanese
1463–1612	Pre-rotated forms of CIDs 1–150
1613–2920	Vertical forms of CIDs 151–1462
2921–2972	Vertical-only hiragana ligatures and pre-composed double kana iteration marks

8. The Latest News

Licenses to use Kazuraki became available for sale for $35 on Adobe's North America Type Showroom on December 7, 2009, and on their Japan, France, and Germany Type Showrooms on December 21, 2009.

Summary: OpenType Japanese Font Tutorial: Kazuraki

Adobe System's Type Engineering & Design team in Japan has developed a ground-breaking and innovative new typeface design that breaks the mold that has constrained Japanese typefaces for decades. The typeface design, created by Adobe's own Ryoko Nishizuka, was inspired by the calligraphy of the 12th century Japanese calligrapher and writer Fujiwara-no-Teika, and its final production to produce a functional OpenType font leveraged three powerful AFDKO (Adobe Font Development Kit for OpenType) tools, *tx*, *mergeFonts*, and *rotateFont*, to implement its complex metrics.

Kazuraki is unique among other mainstream Japanese typefaces in that it is fully proportional, in both writing directions. Some glyphs are wider than they are tall, and some are taller than they are wide, and this is reflected in their metrics. For this reason, and because subtle shifting is required for correct positioning of each glyph, there are separate glyphs for both writing directions. In other words, for the 1,082 kanji that are supported in the current version, the font contains 1,082 glyphs for horizontal use, and 1,082 glyphs for vertical. In addition, Kazuraki also includes a significant number of two-, three-, and four-character hiragana ligatures for vertical use.

The tutorial that is reprinted here in its entirety is designed to guide font developers in building special-purpose OpenType fonts, using Kazuraki as an example of how to build a fully-proportional Japanese font. The current version can always be accessible here:

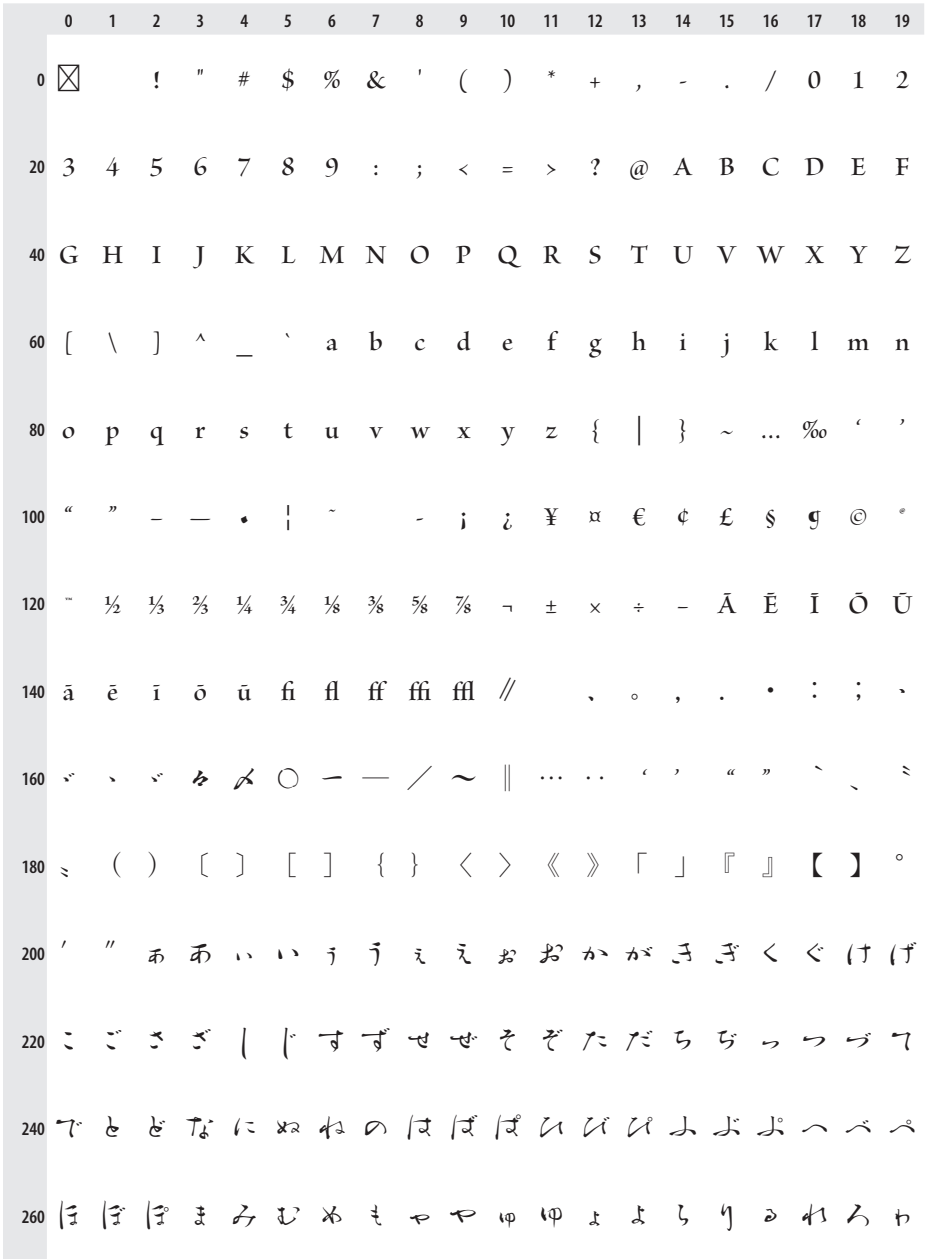
`http://www.adobe.com/devnet/font/pdfs/5901.Kazuraki_Tutorial.pdf`

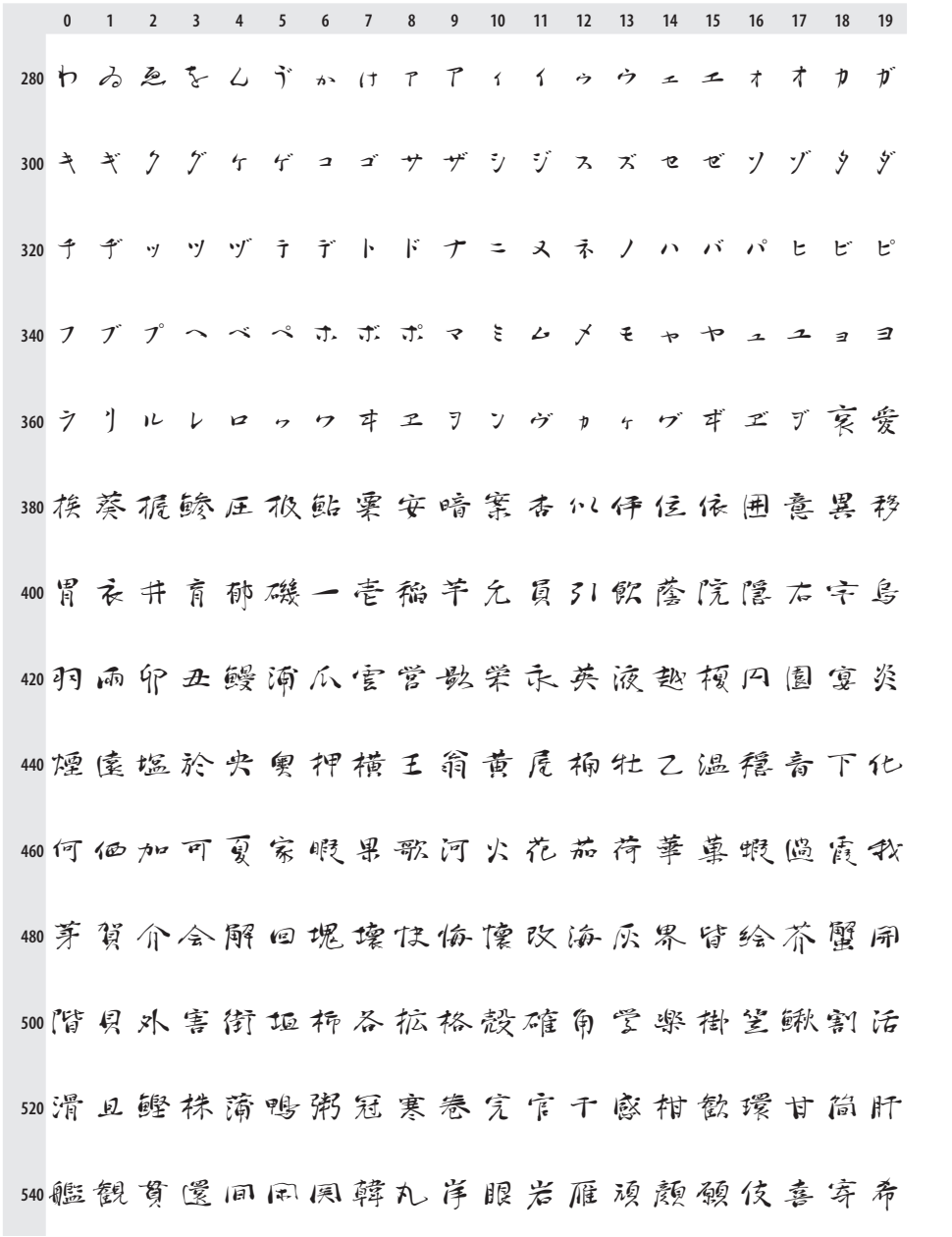
The Kazuraki specimen book, which demonstrates how this font can be used, is available here:

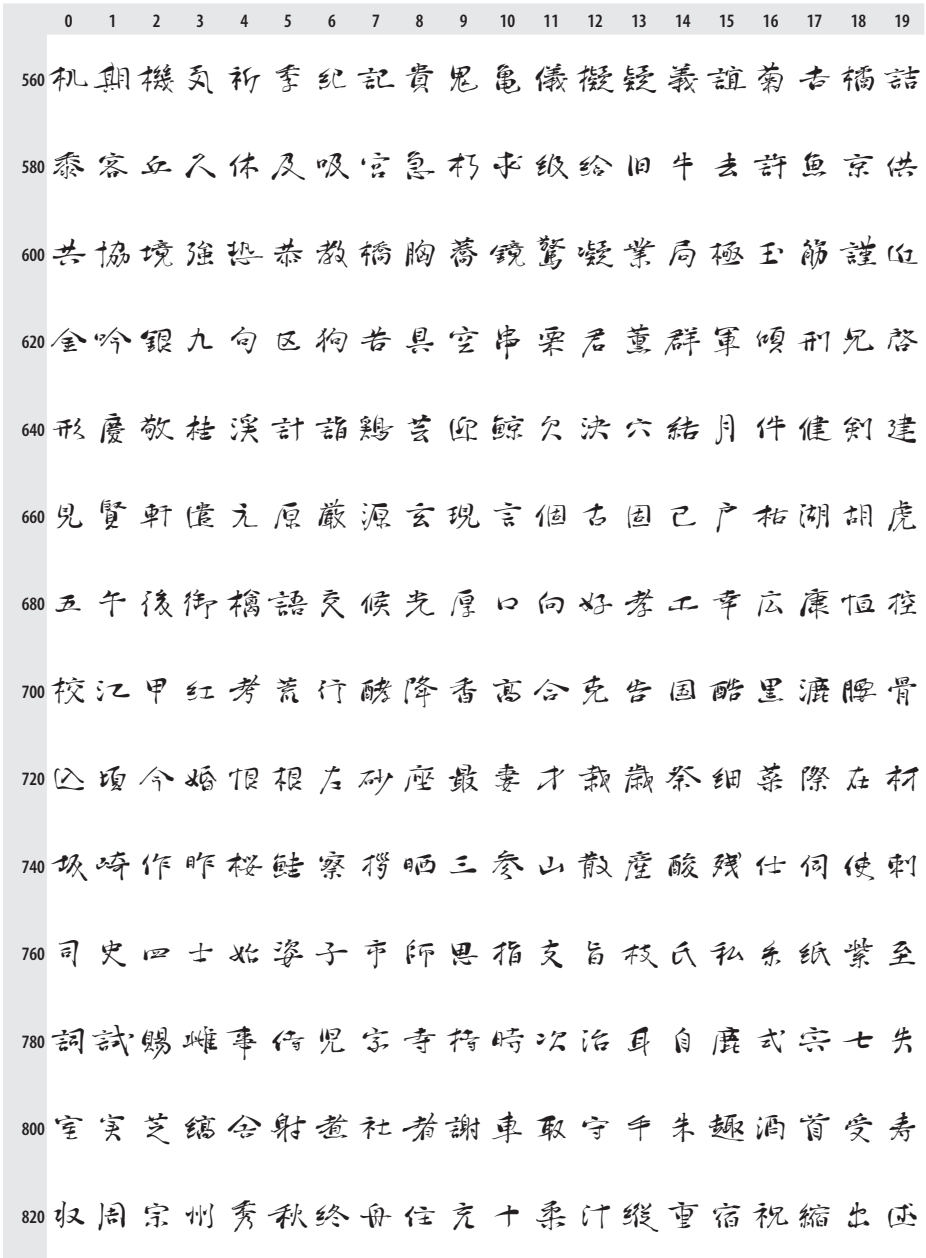
`http://store4.adobe.com/type/browser/pdfs/Kazuraki_SPN.pdf`

Keywords: Kazuraki font, Proportional glyphs, Japanese font, OpenType font, AFDKO, Adobe, *makeotf*, *tx*, *mergeFonts*, *rotateFont*.

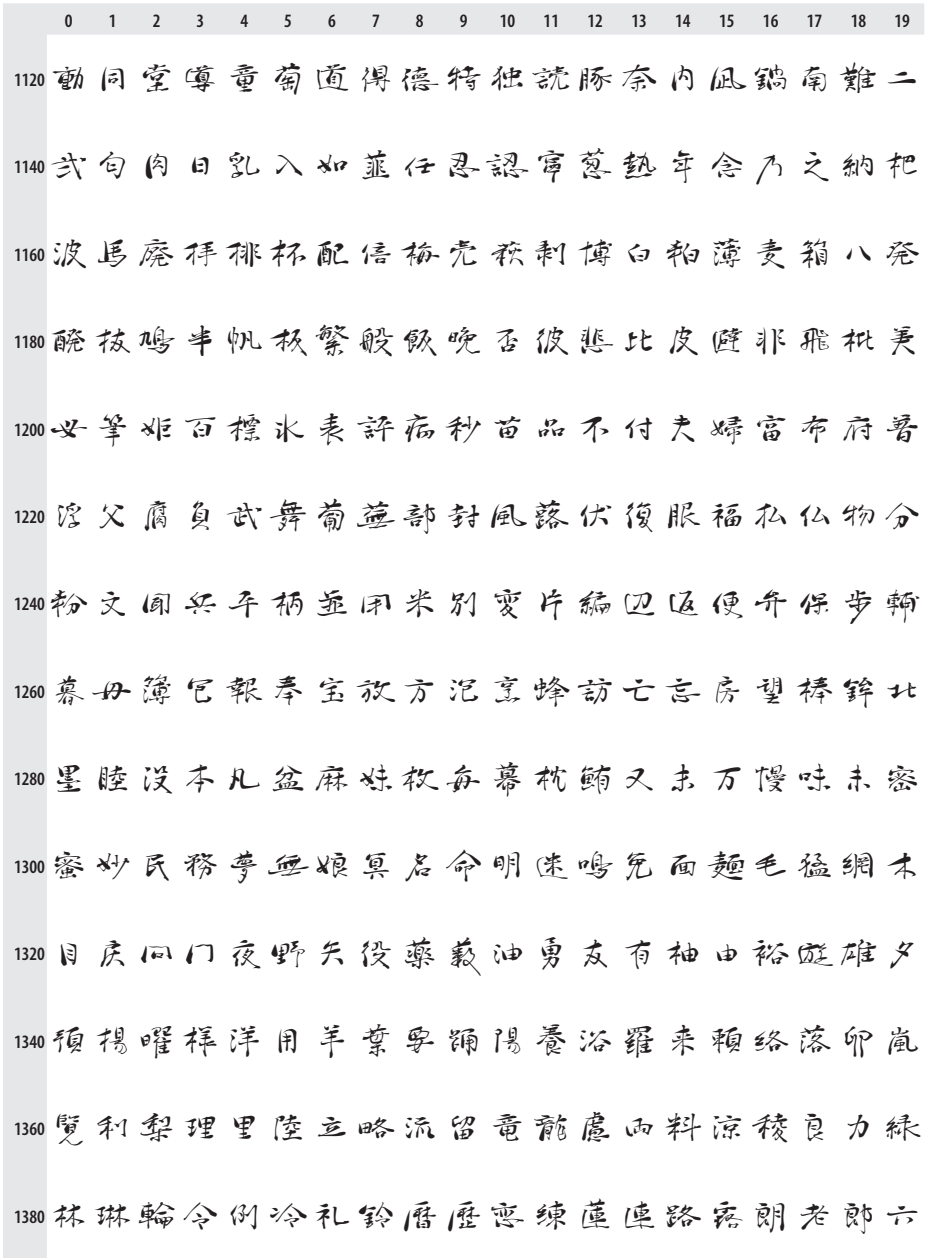
Dr. Ken Lunde (`lunde@adobe.com`)
Senior Computer Scientist, CJKV Type Development
Adobe Systems Incorporated, 345 Park Avenue
San Jose, CA 95120-2704 USA


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
0 ☒ ! " # $ % & ' ( ) * + , - . / 0 1 2
20 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F
40 G H I J K L M N O P Q R S T U V W X Y Z
60 [\ ] ^ _ ` a b c d e f g h i j k l m n
80 o p q r s t u v w x y z { | } ~ ... ‰ ‘ ’
100 “ ” – — • ¦ ˜ - ¡ ¿ ¥ ¤ € ¢ £ § ❡ © ª
120 ¨ ½ ⅓ ⅔ ¼ ¾ ⅛ ⅜ ⅝ ⅞ ¬ ± × ÷ – Ā Ē Ī Ō Ū
140 ā ē ī ō ū fi fl ff ffi ffl // 、 。 , . • : ; 、
160 〻 ゝ 〻 々 〆 〇 ー — ／ 〜 ‖ … ‥ ‘ ’ “ ” 〳 〴 〵
180 〵 ( ) 〔 〕 [ ] { } 〈 〉 《 》 「 」 『 』 【 】 °
200 ′ ″ ぁ あ ぃ い ぅ う ぇ え ぉ お か が き ぎ く ぐ け げ
220 こ ご さ ざ し じ す ず せ ぜ そ ぞ た だ ち ぢ っ つ づ て
240 で と ど な に ぬ ね の は ば ぱ ひ び ぴ ふ ぶ ぷ へ べ ぺ
260 ほ ぼ ぽ ま み む め も ゃ や ゅ ゆ ょ よ ら り る れ ろ わ

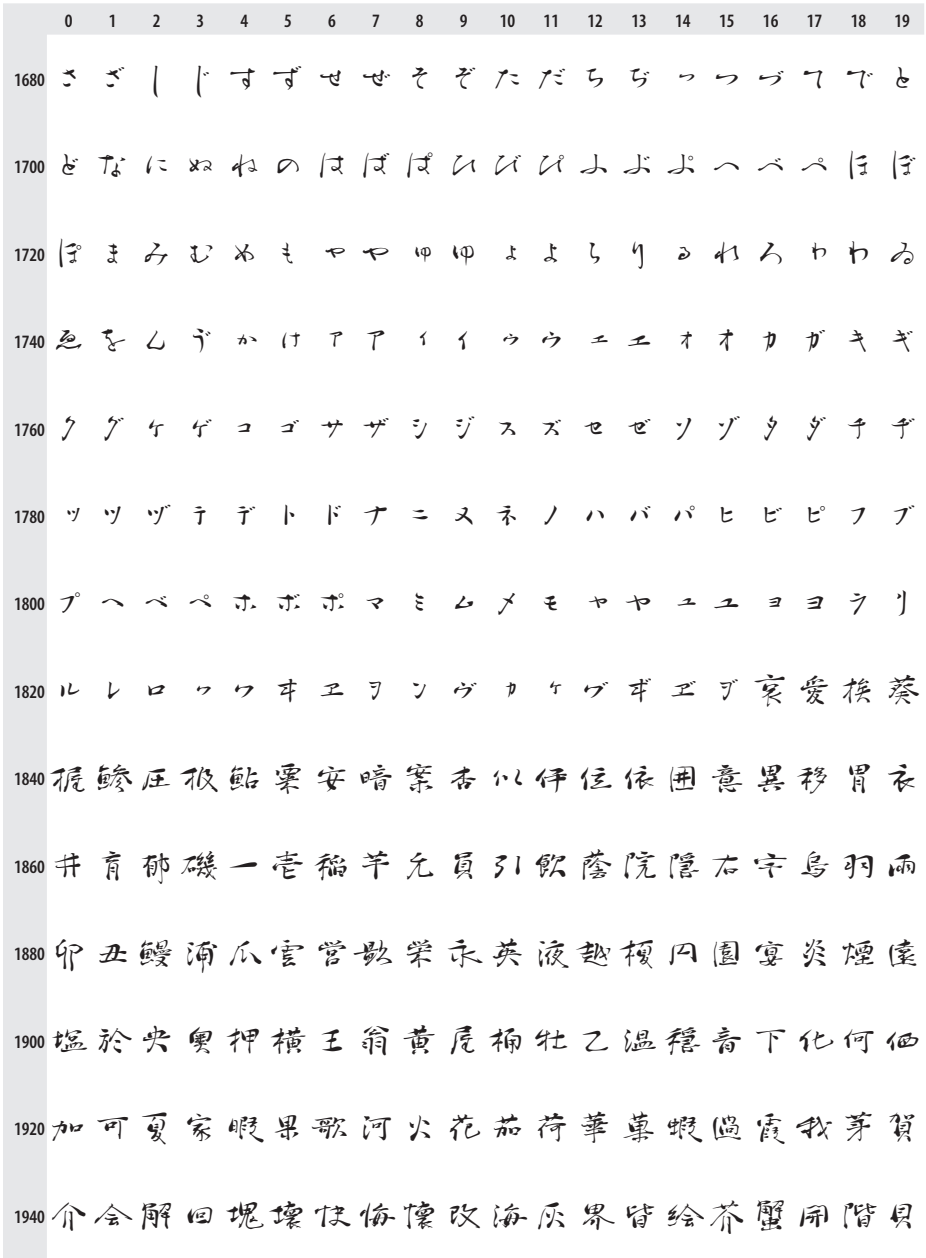

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
280 わ ゐ ゑ を ん ゔ か け ァ ア ィ イ ゥ ウ ェ エ ォ オ カ ガ
300 キ ギ ク グ ケ ゲ コ ゴ サ ザ シ ジ ス ズ セ ゼ ソ ゾ タ ダ
320 チ ヂ ッ ツ ヅ テ デ ト ド ナ ニ ヌ ネ ノ ハ バ パ ヒ ビ ピ
340 フ ブ プ ヘ ベ ペ ホ ボ ポ マ ミ ム メ モ ャ ヤ ュ ユ ョ ヨ
360 ラ リ ル レ ロ ヮ ワ ヰ ヱ ヲ ン ヴ ヵ ヶ ヷ ヸ ヹ ヺ 哀 愛
380 挨 葵 茜 穐 鯵 圧 扱 鮎 粟 安 暗 案 杏 ㈱ 伊 位 依 囲 意 異 移
400 胃 衣 井 育 砺 磯 一 壱 稲 芋 允 員 引 飲 蔭 院 陰 右 宇 烏
420 羽 雨 卯 丑 鰻 浦 瓜 雲 営 曳 栄 永 英 液 越 援 円 園 宴 炎
440 煙 遠 塩 於 央 奥 押 横 王 翁 黄 屋 柿 牡 乙 温 穏 音 下 化
460 何 価 加 可 夏 家 暇 果 歌 河 火 花 茄 荷 華 菓 蝦 過 霞 我
480 茅 賀 介 会 解 回 堺 壊 快 悔 懐 改 海 灰 界 皆 絵 芥 蟹 開
500 階 貝 外 害 街 垣 柿 各 拡 格 殻 確 角 堂 楽 掛 窒 鰍 割 活
520 滑 且 鰹 株 蒲 鴨 粥 冠 寒 巻 完 官 干 感 柑 歓 環 甘 簡 肝
540 艦 観 貫 還 間 関 陥 韓 丸 岸 眼 岩 雁 頑 顔 願 伎 喜 寄 希

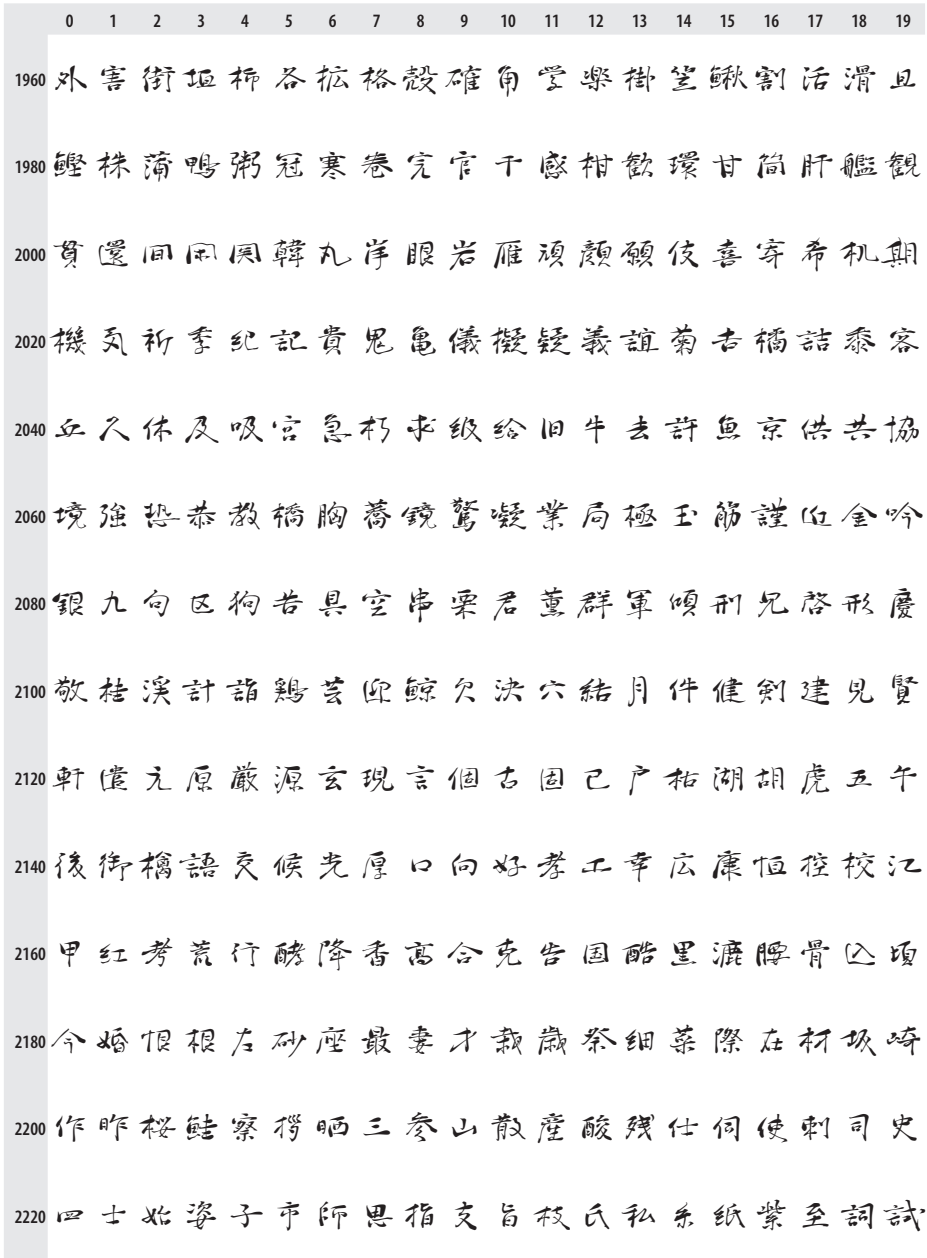

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
560
580
600
620
640
660
680
700
720
740
760
780
800
820

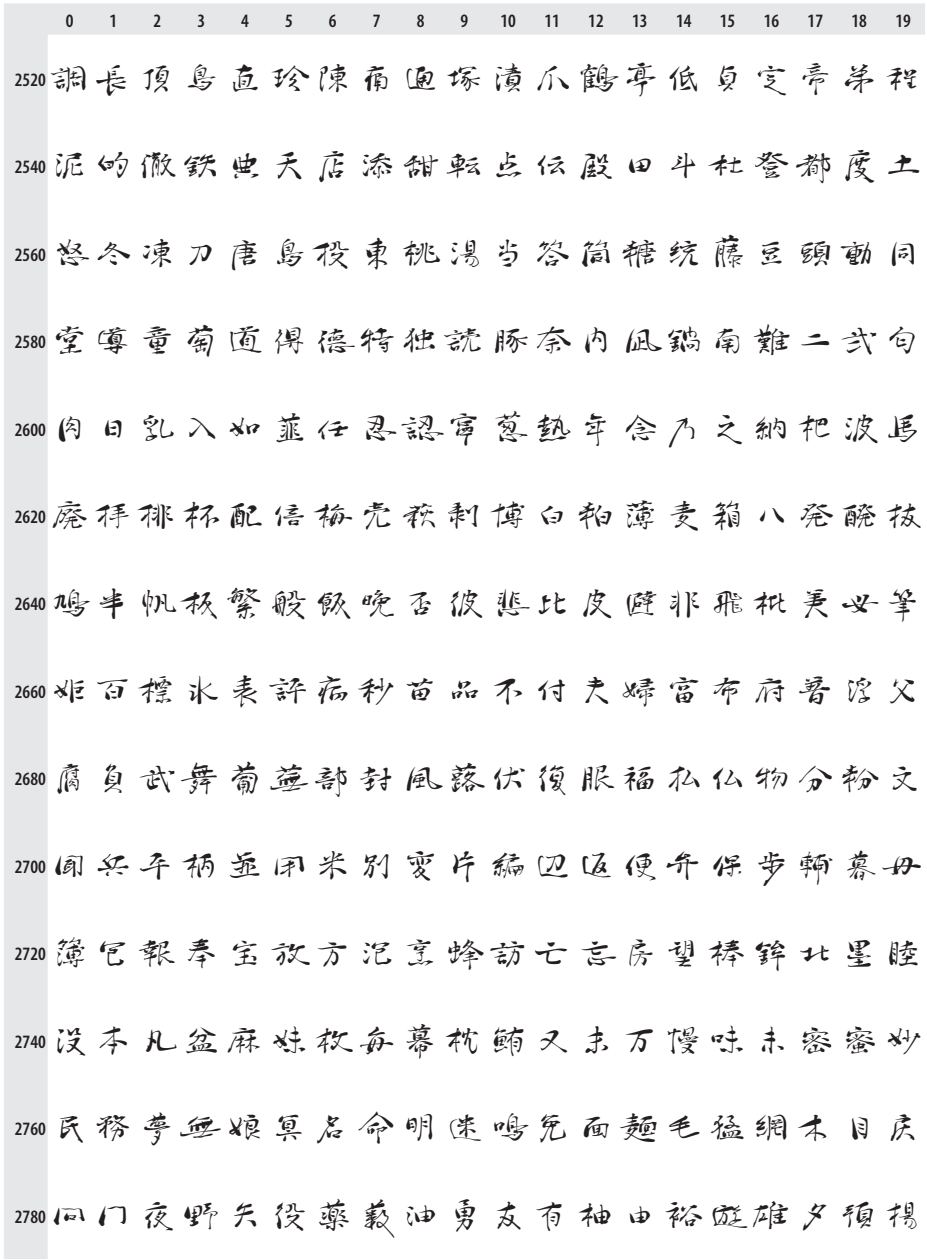
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
840	俊	春	旬	洞	純	順	処	初	所	暑	署	書	諸	助	女	除	傷	商	将	小
860	少	抄	昭	晶	松	消	焼	照	証	賞	鐘	鐘	障	上	大	乗	城	場	嬢	常
880	情	条	状	蒸	醸	植	色	食	信	心	振	新	深	申	真	神	吾	親	身	辛
900	進	針	人	仁	塵	壬	尋	尽	酢	吹	水	炊	酔	随	数	杉	雀	世	瀬	制
920	勢	性	成	星	晴	正	清	生	盛	精	声	製	西	請	青	税	席	惜	昔	石
940	赤	跡	切	折	設	節	雪	絶	舌	蝉	仙	先	千	占	専	川	戦	扇	泉	浅
960	洗	染	選	鮮	前	然	全	禅	膳	祖	素	組	僧	喪	壮	掃	早	曹	巣	相
980	草	藻	装	走	送	窟	増	臓	蔵	贈	造	息	束	俗	賊	族	統	存	尊	他
1000	多	太	打	体	対	待	戴	苔	袋	鯛	代	台	大	第	題	沢	宅	只	達	谷
1020	鱈	丹	単	担	旦	淡	炭	短	胆	団	壇	弾	暖	檀	段	男	値	知	地	智
1040	池	置	匠	竹	筑	茶	着	中	仲	忠	柱	注	虫	酎	猪	丁	帳	張	朝	潮
1060	町	腸	調	長	頂	鳥	直	珍	陳	痛	通	塚	漬	爪	鶴	亭	低	貞	定	帝
1080	弟	程	泥	的	徹	鉄	典	天	店	添	甜	転	点	伝	殿	田	斗	杜	登	都
1100	度	土	怒	冬	凍	刀	唐	島	投	東	桃	湯	当	答	筒	糖	統	藤	豆	頭

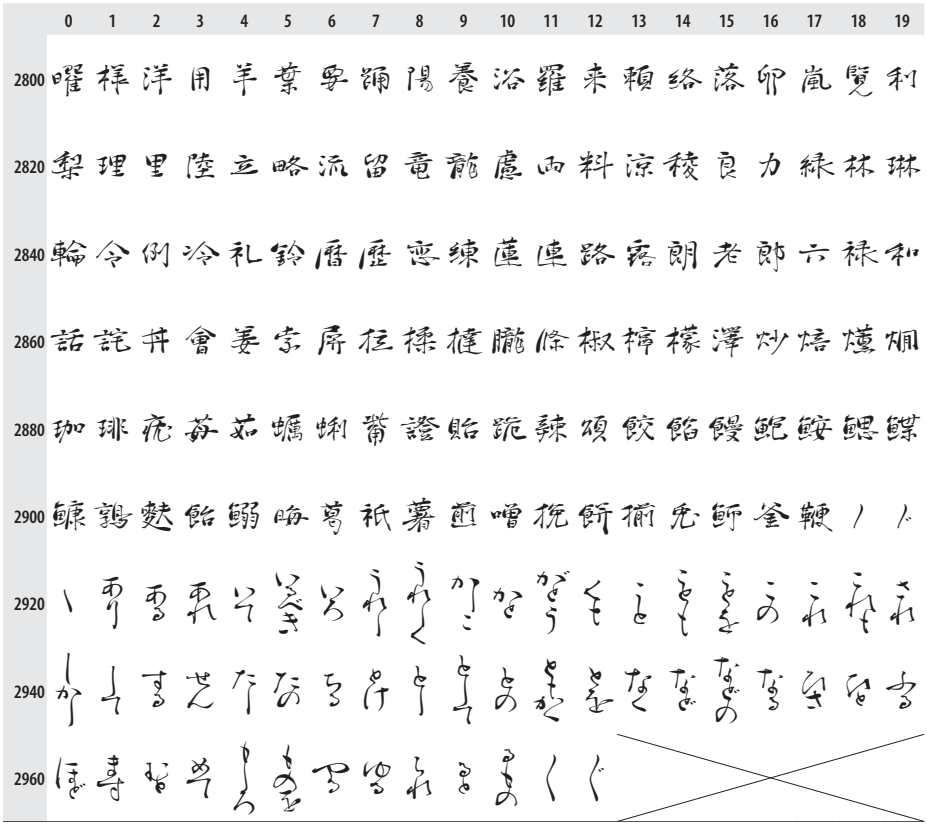

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
1120
1140
1160
1180
1200
1220
1240
1260
1280
1300
1320
1340
1360
1380

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1400	祿	和	話	詫	幷	會	蓋	宗	房	征	棟	橿	臟	條	椒	楠	檬	澤	炒	焙
1420	燻	烱	珈	琳	疣	苺	茹	蠣	蜊	螯	證	貼	蹠	辣	頌	餃	餡	饅	鯢	鯱
1440	鰓	鰈	鰊	鶲	麩	餡	鰯	晦	鴬	祇	薯	亜	噌	祝	舛	禰	兎	鮃	釜	鞭
1460	ノ	ゞ	ヽ		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	0
1480	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	@	A	B	C	D
1500	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
1520	Y	Z	[	\	]	^	_	`	a	b	c	d	e	f	g	h	i	j	k	l
1540	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	…	‰
1560	‘	’	“	”	–	—	•	‼	‾		‐	¡	¿	¥	¤	€	¢	£	§	¶
1580	©	ª	¨	¯	½	⅓	⅔	¼	¾	⅛	⅜	⅝	⅞	±	×	÷	−	Ā	Ē	Ī
1600	Ō	Ū	ā	ē	ī	ō	ū	ff	fi	fl	ffi	ffl	〃		‘	。	’	・	•	:
1620	;	、	゛	゜	々	〆	〇	ー	—	／	＼	＝	：	：	，	‘	〃	（		
1640	）	〔	〕	〔	〕	{	}	〈	〉	≪	≫	「	」	『	』	【	】	。	，	〃
1660	あ	あ	い	い	う	う	え	え	お	お	か	が	き	ぎ	く	ぐ	け	げ	こ	ご


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
1680
1700
1720
1740
1760
1780
1800
1820
1840
1860
1880
1900
1920
1940




0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
2520
2540
2560
2580
2600
2620
2640
2660
2680
2700
2720
2740
2760
2780


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
2800
2820
2840
2860
2880
2900
2920
2940
2960

Jak na výrobu písma kandži s pořadím tahů

Timothy Eyre

Abstrakt

Tahy u každého jednotlivého kandži by měly být psány v přesně daném pořadí. Projekt Aaaa (A亜あ) Ulricha Apela sestavil sadu SVG souborů, které obsahují informace o tazích a jejich pořadí. Díky tomu byl autor článku schopen sestavit písmo tak, že zobrazuje správné pořadí tahů u 6373 kandži. Vzniklé písmo usnadňuje přípravu studijních pomůcek, které tuto informaci využívají.

Klíčová slova: japonština, kandži, pořadí tahů, tvorba písma, formát TrueType, formát SVG, program FontForge.

doi: 10.5300/2010-3/199

Introduction

The strokes of each Japanese character (kanji) should be written in a certain order. This correct stroke order is called *hitsujun* (筆順) in Japanese and helps with legibility and memorization. Students of kanji are examined on kanji stroke order in exams such as the Japanese Kanji Aptitude Test (日本漢字能力検定試験).

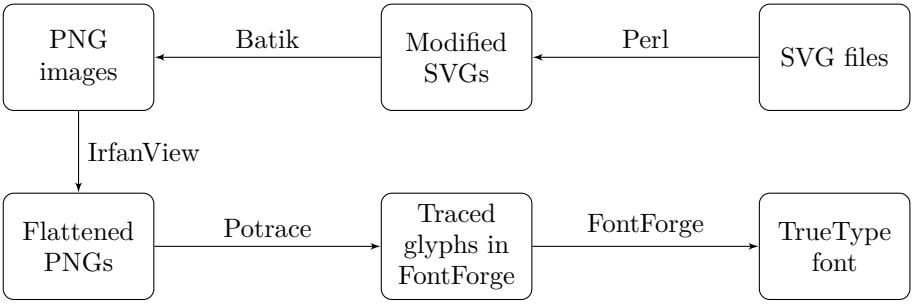
Although there are some general rules about the order in which the strokes of a kanji should be written, it is useful to have a reference available. There are indeed numerous printed and electronic kanji stroke order resources available. However, the coverage of these resources is not always as comprehensive as one might like and the presentation is not always as helpful as it might be. Various means of presenting of kanji stroke order diagrams appear. Step-by-step diagrams are popular in printed resources, such as [1] and animated diagrams are popular in electronic resources such as [2].

Dr. Ulrich Apel and Dr. Julien Quint have produced a large set of SVG files [3], [4] that present the stroke order of several thousand kanji in the static form used in [4]. This impressive data resource certainly excelled as a means of storing the information but retrieval required the use of the A亜あ(Aaaa)-project's web interface and Adobe's SVG browser plugin.

Dr. Apel was kind enough to send me the set of SVG files. A few years earlier it had crossed my mind that a simple way to present kanji stroke order diagrams would be to build them into a Unicode font. The SVG files from the A亜あ-project provided me with the data I needed to create such a font. This paper describes how I converted the SVG data into the Kanji Stroke Order Font and how I maintain the font now it exists.

1. Method of Creation

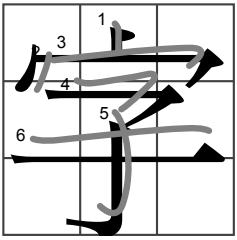
The following flowchart summarizes the steps involved in creating the Kanji Stroke Order Font.



The subsections below describe each of these steps in detail.

1.1. Modifying the SVG files

The original SVG files contained graphical information that was not necessary for the font. Here is an example of a raw SVG image:



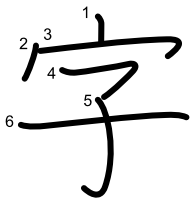
As you can see, while the pen stroke version of the kanji and numbers are required for the font, the grid and printed version of the kanji are not. By means of a Perl script, I processed the contents of each SVG file to remove the superfluous elements and change the colour of the pen strokes from grey to black. The fact that SVG is an XML-based file format and therefore plain text made this processing simple: I eliminated the large print kanji simply by changing its font size from 108 to 0, changed the colour of the grid from dark grey to white and the colour of the pen strokes from mid grey to black. A simplified version of the Perl script looks like this:

```
while ($line = <STDIN>)
{
  $line =~ s/#808080/#000000/;
  $line =~ s/#404040/#FFFFFF/;
  $line =~ s/font-size:108/font-size:0/;
  print $line;
}
```

The `while` loop reads the standard input until the last line of the redirected file is reached. The three `$line =~ s/foo/bar/;` lines substitute `foo` for `bar`. This means, for example, that `$line =~ s/#808080/#000000/;` replaces light grey with black. This script would be executed as follows:

```
perl preproc.pl < infile.svg > outfile.svg
```

However, in practice we are handling several thousand files so some extra Perl globbing code is needed. After this simple processing, the SVG image is more suitable for inclusion in a font, looking as it does like this:



1.2. Conversion to PNG using Batik

The Batik SVG Toolkit [6] is a Java library that can be used to render SVG graphics. Rendering the several thousand SVG files created in the previous step simply required multiple executions of commands like the following:

```
java -jar batik-rasterizer.jar -d . -m image/png
                -h 2000 -dpi 2000 outfile.svg -bg 255.255.255.255
```

The parameters for this command are explained as follows:

- The parameter `-d .` simply specifies that the local directory should be used for the output.
- The parameter `-m image/png` is a MIME specification, indicating that the output should be PNG rather than any of the other formats that Batik can produce.
- The parameter `-h 2000` specifies the height of the output in pixels.
- As a result of the previous parameter, the parameter `-dpi 2000` ensures that the resulting PNG image is exactly one inch tall.

- The parameter `outfile.svg` specifies the SVG file to be rendered; it is called `outfile.svg` here because it is the output from the step described in section 1.1.
- The parameter `-bg 255.255.255.255` specifies the background fill colour, including the alpha (transparency) parameter.

Batik renders the SVG files relatively slowly, so rendering all of them requires an overnight run. At the end of this process we will have several thousand PNG files, one for each character that is going to go into the font.

1.3. Flattening the PNGs

Batik produces 32-bit PNG files. This generous image depth has two disadvantages for our purpose here. Firstly, it makes the PNG files needlessly large, an important consideration when there are literally several thousand of them. Secondly, Batik uses subpixel rendering (the technique used by CoolType and ClearType) to improve the appearence of the output. The grey fuzz around each kanji looks pleasing to the human eye but gives extra work to Potrace. Therefore, it is desirable to reduce the number of bitplanes in the PNG files to 1. The IrfanView [7] image viewing and processing utility has a GUI tool that makes this bitplane reduction straightforward, albeit difficult to document.

1.4. Tracing the PNGs

At last we are ready to start using FontForge [8], the BSD-licensed outline font editor that runs on Linux, Mac and even Microsoft Windows with the help of Cygwin [9]. FontForge includes a trace facility, which calls out to the Potrace [10] bitmap-to-vector tracing tool under the covers. FontForge can also use Autotrace [11] to perform the same function.

When doing bulk tracing into a Unicode font, FontForge requires the filename of the source image file to be of the form `uniXXXX.png`, where `XXXX` represents the 4-digit hexadecimal number corresponding to the Unicode number for the character. This fits in well with our workflow so far.

Invoking the trace feature only requires two menu option selections in FontForge. The first option is File→Import with the Format option set to 'Image Template'. This creates a background image for each of the characters. We then tell FontForge to trace these background images by selecting the option Element→Autotrace.

The tracing process takes some hours and uses copious resource on the host machine. One of the challenges I encountered in this project was finding a PC that was powerful enough to handle this task.

Once the tracing is complete the font can be exported from FontForge in whatever format you choose. I distribute the Kanji Stoke Order Font in the TrueType format.

2. Polishing the Font

As with all real-life projects, the font created by the workflow above is imperfect. The following subsections describe the polishing that I did to produce a usable font. At no stage in the creation of the font did I check each glyph for the accuracy of the stroke orders; with several thousand characters this was not practical.

2.1. Character Sizes

The most obvious glitch was that some of the kanji were too large in comparison with the other characters in the font. Correcting the rogue characters was simply a matter of invoking a scaling transformation for the character in FontForge. Finding the rogue characters was more of a challenge.

I tracked down the oversized characters in two ways, both using XeTeX and a list of the codepoints covered by the font.

The first chunk of XeTeX code prints the height of each character to screen, allowing for redirection to file, sorting and subsequent identification of outliers.

```
\font\cf="KanjiStrokeOrders" at 40pt
\def\boxit#1{\hbox{\vbox{\hrule\hbox%
  {\vrule\vbox{#1}\vrule}\hrule}}}
\def\charfont#1{\setbox0=\hbox{\boxit{\hbox{\cf#1}}}%
                \immediate\write15{\the\ht0#1}}
\input test_data.tex\bye
```

The file `test_data.tex` is simple, containing just several thousand lines of the following form:

```
...
\charfont{\char"9051}{9051}%
\charfont{\char"9052}{9052}%
\charfont{\char"9053}{9053}%
...
```

The second chunk of XeTeX code prints the characters themselves, which enabled me to view the characters in bulk and identify ones that looked wrong.

```
\font\cf="KanjiStrokeOrders" at 40pt
\nopagenumbers\pdfpagewidth=331mm\pdfpageheight=207mm
\hsize=300mm\vsize=200mm
\parindent=0pt\hoffset=-0.5in\voffset=-0.9in\baselineskip=0pt
\def\boxit#1{\hbox{\vbox{\hrule\hbox%
  {\vrule\vbox{#1}\vrule}\hrule}}}
\def\charfont#1#2{\setbox0=\hbox{\boxit{\hbox{\cf#1}}}\box0\ }
\input test_data.tex\bye
```

Figure 1: Kanji Stroke Order Font test page

Figure 1 shows a typical page generated by this XeTeX fragment. There are 28 test pages in total, which means that looking through them for mis-formatted characters is a tractable task.

2.2. Making the character widths consistent

All Japanese kanji notionally occupy a square of constant width. Therefore it was necessary to adjust the width of the characters to all be the same. FontForge provides a scripting language, which makes tasks like this easy.

```
SelectWorthOutputting()
SelectFewer(0u0000,0u2e8b) # Remove non-Japanese chars
SelectFewer(0uFF61,0uFF9F) # Omit half-width katakana
foreach
  Print ("Doing next Jp ", GlyphInfo("Unicode"))
  SetWidth(1024);
  CenterInWidth();
endloop
```

Note the line of code that omits resizing the half-width katakana. These are a relic from the early days of Japanese-language computing [12]. I use a similar block of code with a call to `SetWidth(512)` to set the width of these characters.

2.3. Cleaning up the outlines

An effect of tracing the characters from image files is a surplus of points in the character outlines. FontForge has a simplification function that cuts down the number of points, thereby reducing the file size and improving maintainability.

FontForge also has a validation function, which enabled me to clean up some problems with the TrueType font. However, so powerful is FontForge's validation feature that it was not practical to fix every validation error.

2.4. Add missing characters

The source SVG dataset omitted some characters that are commonly used but not relevant to stroke order analysis. To make the Kanji Stroke Order Font more usable I merged it with another font to fill in the gaps; FontForge has a built-in feature for doing exactly this. Originally I used a public domain font called Tuffy [13] but once I had created the Unicode font Choumei [14] I used that to fill in the gaps in the Kanji Stroke Order Font. I describe the Choumei font in a separate section below.

Creation of the Kanji Stroke Order Font was now complete so I was able to publish it on my website [15].

3. Maintenance

Maintaining the font is fiddly but simple. I rely on reports from users to track down errors in the stroke order diagrams. When a user reports an error I verify that the proposed revision to the stroke order is correct and then use FontForge to change the affected character. The stroke order numbers are stored as shapes in the font, with no references or underlying structure. This means that the only way of changing the order of the stroke numbers is to cut and paste them.

At the time of writing, the current version of the Kanji Stroke Order Font is v2.014. This contains 146 corrections to the initial revision. By the time this paper is published I shall have produced v2.015 of the Kanji Stroke Order Font, which will contain several more corrections.

User feedback is important for the Kanji Stroke Order Font: it is not practical for me to check all the glyphs myself and most of the users of the font are undoubtedly more skilled in kanji than I am. Moreover, accuracy is important because the ease of use and broad coverage of the Kanji Stroke Order Font means there is a risk that some users might treat it as being authoritative, which it most certainly is not.

4. Future Possibilities

Dr. Apel has moved on to create the KanjiVG project [16], which uses a defined SVG format to store information about drawing kanji. It would be theoretically possible to automatically create the Kanji Stroke Order Font from this data using a script based on the technologies described in this paper.

A major advantage of this approach would be that the two projects could be kept synchronized and corrections to either would be picked up by both. My experience with creating the Kanji Stroke Order Font showed that font creation requires a significant amount of manual tweaking, so it is not clear that such a process would work in practice. This is an area for future study.

5. Choumei Unicode font

Free (as in speech or beer) fonts that cover a large number of kanji are relatively few in number. This is to be expected because creating a quality kanji font requires a large amount of effort. However, with the source data for the Kanji Stroke Order Font already to hand, it was comparatively easy to create a comprehensive kanji font (albeit one of dubious quality) by omitting the stroke order numbers from the glyphs in the Kanji Stroke Order Font. I achieved this by running through the Kanji Stroke Order Font workflow steps described in this paper with a single change: I set the font size of the kanji stroke numbers to zero in the Perl script at the SVG modification stage. Thus, with minimal additional effort, I was able to publish a free kanji font [14]. Given its simple appearence, I named it Choumei for Kamo no Choumei (鴨長明, 1155?–1216), author of the essay *Life in a Ten Foot Square Hut*.

Having created this initial draft of the Choumei font, I used FontForge to add to its glyph compliment. I did this mostly by cutting, pasting and modifying existing glyphs rather than drawing glyphs afresh. The characters I added were mostly symbols and accented Latin letters. Having expanded Choumei significantly, I merged it with the Kanji Stroke Order Font to expand the latter font's glyph coverage, albeit without stroke numbers.

I now treat the Choumei font as unmaintained.

Conclusion

A font can be generated from SVG data using a workflow, each step of which uses free (as in speech or beer or both) tools. Each of these steps is scalable and therefore it is practical to use the workflow for a set of several thousand SVG files. This scalability made it possible to convert the A亜アあ-project's kanji stroke order SVG files into a TrueType font.

References

[1] Halpern, Jack: Kôdansha Kanji Learner's Dictionary, Kôdansha International, December 2001. ISBN 978-4-770-02855-6.

[2] Kanji Café website: `http://www.kanjicafe.com/`

[3] Quint, Julien; Apel, Ulrich: *Does Learning How to Read Japanese Have to Be So Difficult And Can the Web Help?*, Proceedings of the WWW 2005 conference, Chiba, Japan, May 2005. ISBN 1-59593-052-3. URL: `http://www2005.org/cdrom/docs/p1152.pdf`

[4] Quint, Julien; Apel, Ulrich: *Teaching and Reference Material on Japanese Kanji in SVG Stroke Order, Animated Drawings of Characters, Kanji Components and their relationships.* Proceedings of the SVG Open 2004 conference, Tokyo, Japan, September 2004. URL: `http://www.svgopen.org/2004/papers/svgopen/`

[5] O'Neill, P. G.: *Essential Kanji: 2,000 Basic Japanese Characters Systematically Arranged for Learning and Reference*, Weatherhill Inc., 1 Jan 1974. ISBN 978-0-834-80222-3.

[6] Batik SVG Toolkit: `http://xmlgraphics.apache.org/batik/`

[7] IrfanView: `http://www.irfanview.com/`

[8] FontForge: `http://fontforge.sourceforge.net/`

[9] Cygwin: `http://www.cygwin.com/`

[10] Potrace: `http://potrace.sourceforge.net/`

[11] Autotrace: `http://autotrace.sourceforge.net/`

[12] Hensch, Kurt: Research and Development in IBM, History of Far Eastern Languages in Computing, 2nd private edition, Roehm TYPOfactory GmbH, Sindelfingen, Germany, 2004. ISBN 3-937267-03-4.

[13] Tuffy font: `http://tulrich.com/fonts/`

[14] Choumei Unicode Font: `http://www.nihilist.org.uk/`

[15] Kanji Stroke Order Font: `http://www.nihilist.org.uk/`

[16] KanjiVG project: `http://kanjivg.tagaini.net/`

Summary: Creating a Kanji Stroke Order Font

This article describes how a font that displays Kanji Stroke Orders can be created from thousands of SVG files containing this information.

Keywords: Kanji, Stroke Order, Font, TrueType, SVG, FontForge.

Timothy Eyre, `mail@nihilist.org.uk`
CSTUG c/o FEL ČVUT, Technická 2
Prague, CZ-166 27, Czech Republic

PDFdiff: skript srovnávající PDF soubory

Timothy Eyre

Abstrakt

Článek představuje skript naprogramovaný v Pythonu, který na vstupu očekává dva PDF soubory a automaticky je zpracuje nástroji pdftk, Ghostscript, ImageMagick a XeTeX tak, že vznikne nový PDF soubor, který ukazuje rozdíly mezi oběma vstupy.

Klíčová slova: Srovnání souborů, Python, Ghostscript, pdftk, ImageMagick.

doi: 10.5300/2010-3/208

Introduction

Tools that show the difference between two files are a mainstay of software development. They are most commonly used to view the difference between two versions of a source code file; you might even use such a tool when you are developing TeX code. Less common, but still potentially useful, are tools that show the difference between two binary files.

This article describes the motivation for and implementation of a tool that shows the differences between two PDF files.

1. Motivation

Here are three examples where a PDF file difference-finding tool would be useful:

(1) You have two versions of a paper in PDF format. You know that the differences between them are slight but you need to know exactly what the differences are.

(2) You have created a document using the Kanji Stroke Order Font [1] and want to create an equivalent version but without the stroke order numbers. A colleague tells you that the Choumei font (also available from [1]) is simply the Kanji Stroke Order Font with the stroke numbers removed. You suspect that the Choumei font may have different metrics to the Kanji Stroke Order Font and will therefore change the layout of your document. You need a way to test this.

(3) You have re-written a TeX macro used by a large publication to make it more maintainable or to add a new feature. You are fairly sure that the new version of the macro will not change the publication's layout in any way but need a way to verify this.

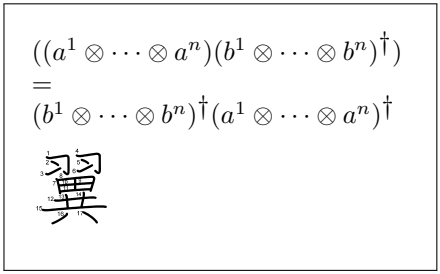


Figure 1: Pre-modification version of the PDF file

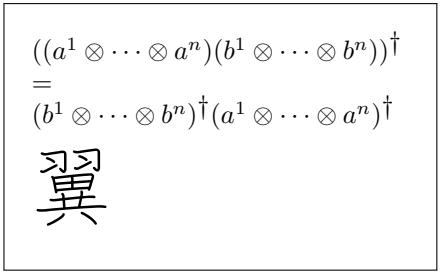


Figure 2: Post-modification version of the PDF file

Figures 1 and 2 illustrate the first two of these three scenarios. The change to the mathematical expression between Figures 1 and 2 is easy to see, but such changes are harder to spot in a a larger article. It is not obvious whether the change from the Kanji Stroke Order Font to the Choumei font would change the spacing of the document.

In both cases, the output of PDFdiff shown in Figure 3 makes the changes easy to see.

2. Implementation

I implemented PDFdiff using Python to stitch together the steps summarized in Figure 4.

This section describes each of these steps in turn. I use the words 'pre' and 'post' to denote the versions of a document before and after some sort of modification that we invoke PDFdiff to determine. In practice it may not be the case that there is a well-defined notion of 'pre' and 'post' version but it is a useful shorthand for identifying the two documents.

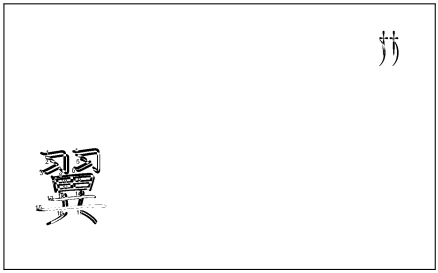


Figure 3: Difference between the two PDF files

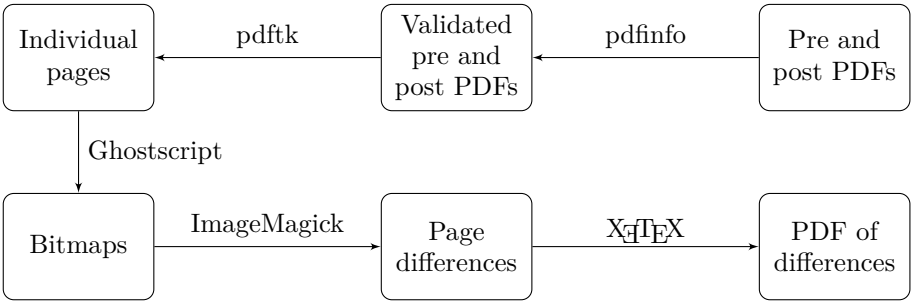


Figure 4: PDFdiff workflow

2.1. Validation

To be able to compare the pre and post versions of the document sensibly, the two versions of the document must have the same dimensions. Therefore the first thing that PDFdiff does is to call out to the tool `pdfinfo` [2] to extract the dimensions of the document under scrutiny.

If the dimensions of the pre and post versions of the document differ then PDFdiff reports this and exits.

2.2. Burst

To simplify the process of generating the bitmaps with Ghostscript, PDFdiff first bursts the input PDF files into multiple PDF files, each containing a single page of either the pre or post version of the document. PDFdiff does this by calling out to the tool `pdftk` [3] as follows:

```
pdftk pre.pdf burst
pdftk post.pdf burst
```

After each call to pdftk, PDFdiff renames the files. This is necessary because pdftk chooses its own output filenames and so we need to prevent the output of the second burst from overwriting the output of the first.

PDFdiff then checks that the pre and post versions of the document have the same number of pages. If the two documents differ in length then PDFdiff will inform you of the fact. If you are then determined to compare the two versions of your document regardless of their differing number of pages then it is a simple matter to concatenate a few extra pages to the end of the shorter document to even up the lengths.

At the end of this burst process, we should have $2n$ new PDF files, where n is the number of pages in the document under scrutiny.

2.3. Pages to Bitmaps

PDFdiff is now ready to convert the pages it has extracted into bitmap files. It does this using Ghostscript [4], as follows:

```
gswin32 -dNOPAUSE -r600 -dBATCH -sDEVICE=bmp256
                                    -sOutputFile=xxx.bmp xxx.pdf
```

Here `xxx` denotes the name of a single-page PDF file that PDFdiff produced at the burst stage. `-dNOPAUSE` tells Ghostscript that it should not wait for user input. `-r600` tells Ghostscript to use a resolution of 600dpi, which I judge to be a reasonable compromise between quality and resource use. `-dBATCH` tells Ghostscript to exit after processing the supplied PDF file rather than leaving the Ghostscript environment open. `-sDEVICE=bmp256` tells PDFdiff to write an 8-bit colour bitmap file. This allows some degree of colour to be preserved in the final difference file.

2.4. Subtract Bitmaps

Having called out to Ghostscript $2n$ times to produce $2n$ bitmaps, PDFdiff calls out to the ImageMagick software suite [5] to subtract the bitmaps from one another, as follows:

```
composite -compose difference pre_i.bmp post_i.bmp diff_i.bmp
```

In this command, `pre_i.bmp` denotes the bitmap generated from page i of `pre.pdf`, `post_i.bmp` denotes the bitmap generated from page i of `post.pdf` and `diff_i.bmp` represents the file generated by the `composite` command of Imagemagick that shows the difference between the two input bitmap files. In all three cases, $1 \leq i \leq n$.

However, the `diff_i.bmp` files produced by `composite` are not ideal. They are large and the text (if any) will appear as white on a black background. To solve both these problems, PDFdiff runs the bitmap files through another ImageMagick command, as follows:

```
convert -negate diff_i.bmp diff_i.png
```

This produces a much smaller PNG version of the bitmap file and the `-negate` option reverses the colours so that we have black on a white background again.

The ImageMagick processing is the most time-consuming part of the workflow, typically requiring around two-thirds of the total time taken to create the difference file.

2.5. Regenerate Document

Now we have n `.png` files, it is a simple matter to create a new PDF file of n pages with each page containing a single bitmap file as a graphics inclusion. I chose to use XeTeX for this purpose simply because that is the flavour of TeX I use most of the time rather than because I needed a specific feature of XeTeX.

To achieve this, PDFdiff generates some TeX source of the following form:

```
\pdfpagewidth=155pt\pdfpageheight=93pt
\hsize=155pt\vsize=93pt\voffset=-1in\hoffset=-1in
\hbox{\XeTeXpicfile "diff_i.png" width 155pt height 93pt}
\vfill\eject
\bye
```

Naturally, there is an `\XeTeXpicfile` inclusion for each $i \in \{1, \ldots, n\}$.

PDFdiff then calls out to XeTeX with the auto-generated TeX source file as the parameter, as follows:

```
xetex diff.tex
```

Once XeTeX completes its processing, we have the desired final PDF file that shows the differences between the pre and post versions of the document.

3. Deficiencies

There are four deficiencies that users may notice with this script: speed, size of output, resolution and input parameters.

3.1. Speed

To run the script to compare the 8 pages of this paper takes 215 seconds on my laptop. It would be nice if the script ran faster. Testing shows that most of the time is taken at the ImageMagick stage, so there is not much scope for increasing the speed.

3.2. Size of Output

The original version of this document is about 280 kB in size. Running PDFdiff over two versions of this document produced an output file 2 MB in size. In general,

the output file produced by PDFdiff is much larger than the input files. This is simply because the page is described as a raster graphic rather than using the font mechanism provided by the PDF standard. Given the size of modern disks this is only likely to be a problem when sending the output of a large difference over a communications link.

3.3. Resolution

PDFdiff relies on raster graphics to create the difference output. Therefore, by definition, the resolution of the output must be finite, unlike the smooth lines of the original vector graphics. Of course, if you are using the old TEX `.pk` files you will be using rasters anyway.

It is easy enough to modify PDFdiff to produce output of arbitarily high resolution (subject to the limitations of Ghostscript and Imagemagick) but this is inevitably at the expense of longer run times and larger output files.

3.4. Input Parameters

Purely out of laziness, I have written the script to assume that the 'pre' version of the PDF document is called `pre.pdf` and the 'post' version of the document is called `post.pdf`. It would be simple to change this but I prefer the discipline of being sure which file is which and the simplifying effect on the Python code.

Conclusion

Using readily-available tools, it is possible to create a workflow to compare PDF documents. Comparing PDF documents is relatively resource-hungry but can be invaluable. Not only is PDFdiff useful for developing TEX documents, TEX and related tools also make up important parts of the workflow.

References

[1] Kanji Stroke Order Font and Choumei Font: http://www.nihilist.org.uk/

[2] pdfinfo: http://www.glyphandcog.com/

[3] pdftk: The PDF Toolkit: http://www.accesspdf.com/pdftk/

[4] Ghostscript: http://pages.cs.wisc.edu/~ghost/

[5] ImageMagick: http://www.imagemagick.org/

Summary: PDFdiff: A PDF File Comparison Script

A Python script can be used to take two PDF files and automatically process them with pdftk, Ghostscript, ImageMagick and XeTeX to produce a PDF file that shows the differences between the two input files.

Keywords: PDF, file difference, Python, Ghostscript, pdftk, ImageMagick.

Timothy Eyre, mail@nihilist.org.uk
CSTUG c/o FEL ČVUT, Technická 2
Prague, CZ-166 27, Czech Republic

Sazba čínštiny v TEXu: historie a současnost

Jjgod Jiang

Abstrakt

Článek seznamuje čtenáře s přehledem sazby čínštiny v TEXu od prvopočátků až do současnosti.

Klíčová slova: CCT, TianYuan, CJK, CTEX, XETEX, `xcp.py`, zhspacing, xeCJK, ctex, ctex-kit, ctex-doc, sazba čínštiny, Google code.

doi: 10.5300/2010-3/215

Introduction

Knuth developed TEX as a 7-bit system at the time it was invented, internationalization became a tough issue ever since then, yet CJK (Chinese, Japanese and Korean) typesetting is probably the toughest part. The biggest problem is that Eastern language systems like Chinese has a lot of characters compare to western languages like Latin, English and French.

There are more than 100 thousands of ideograph based characters and more than 10 thousand of which are used on daily basis. It means the typesetting system should be able to support such a large range of characters and use fonts accordingly. Unfortunately under a 7-bit or 8-bit system like Knuth-TEX, there is no trivial or elegance solution to support that.

CCT and TianYuan

Since the original TEX system does not provide such a capability, workarounds are invented. Early TEX adopters in China are mostly mathematicians, in 1990s, Zhang Linbo invented the CCT (`http://lsec.cc.ac.cn/~zlb/`) system while Xiao Gang and Chen Zhijie invented TianYuan system (TY for short; `http://wims.math.ecnu.edu.cn/ty/`).

Both of them are based preprocessors – using a preprocessing tool to convert from the original TEX source file in raw Chinese to a version that TEX can recognize, most importantly, the DVI generated in such methods can be further processed to use Chinese fonts. This mechanism was not very easy to use but worked fairly well. However, there are still many limitations:

1. preprocessing based solutions are cumbersome, you need an extra wrapper to the TEX executable otherwise files will be messed up;

2. font support is very limited, in both systems mentioned above, you can only use the fonts predefined, it is very difficult for them to support arbitrary fonts.

CJK package

Things changed a lot since Werner Lemberg introduced the CJK macro package (`http://cjk.ffii.org/`), its a pure TEX macro package so you don't need to preprocess the TEX source. Werner also defined a font system for CJK font selection, based on LATEX font selection scheme. To support fonts that contains thousands of glyphs, this system will need users to convert TTF files into multiple "sub-font" files (since TEX can support 256 font files at most, each of them can have 256 glyphs, so theoretically we can use 65536 different glyphs in one single TEX document), each one covers 256 different glyphs, combining this sub-font scheme and a PS/PDF generation driver that supports this scheme, we can correctly generate the resulting PDF (or PostScript). At that moment, DVIPDFM*x* (`http://project.ktug.or.kr/dvipdfmx/`) is such an PDF driver.

A typical CJK document looks like this:

```
\documentclass{article}
\usepackage{CJK}

\begin{document}
\begin{CJK}{GBK}{song}
你好，这里是一段中文内容。
\end{CJK}
\end{document}
```

As you can see, the text wrapped between `\begin{CJK}`... and `\end{CJK}` can be CJK characters, but everything outside cannot be. Obviously that is not very convenient since you will like to define macros containing CJK characters in the preamble from time to time.

This method works pretty well and proved to be quite stable, so more and more Chinese typesetters adopted it. Zhang Linbo also converted CCT to use a similar method, compare to CJK, CCT have more features focused on Chinese typesetting, such as white space adjustment, punctuation width adjustment, etc.

CTEX

As one memorable mark of TEX typesetting in China, Wang Lei and Wu Lingyun created a community called "CTEX" (`http://www.ctex.org/`), which later be-

came the most popular and most active community for TEX learning and development in China. CTEX was originally a mailing list, but converted to a Web forum later.

With the help of many others, Wu Lingyun packaged all mature Chinese TEX typesetting solutions with the most popular TEX distribution on Windows – MiKTEX – to an individual distribution called CTEX. CTEX became the most popular TEX distribution in China ever since then.

XƎTEX

However, with the above solutions, installing fonts for TEX usage is still very much a pain. It requires the users to convert fonts from TTF to TFM and edit mapping file very carefully. Thus, new TEX users consider font installation a very difficult part and try to avoid it as much as possible.

Something revolutionary happened since the introduction of XƎTEX created by Jonathan Kew (`http://scripts.sil.org/xetex`): processing Unicode documents became so simple and straightforward, especially for CJK documents: you can use any CJK characters in Unicode, not just limited to certain local character sets like GBK or Big5 or Shift-JIS. Best of all, you can use any system font directly, without doing any clumsy "subfont" conversion or carefully creating mapping files for PS/PDF drivers.

However, XƎTEX was only available to Mac users when it was first developed, because only a few TEX users use Mac at that time, it was not widely recognized. Soon after Jonathan Kew ported it to Linux and Akira Kakuto provided the Windows port (through `w32tex` and TEX Live), XƎTEX became very popular among Chinese users.

The low-level facility provided by XƎTEX is enough to typeset regular Chinese documents, but typesetters always want better results: switch fonts automatically between CJK and Latin text, adjust the width of punctuations to achieve better line breaking, better support for vertical layout are some of the common requests.

The `xcp.py` script and **zhspacing** package

Among these request, automatic font switching is especially urgent. At that time, most Latin glyphs from Chinese fonts are so badly designed that it is almost impossible to use. Naturally a typesetter would like to use the high quality Latin fonts for non-Chinese texts.

At first, Jiang Jiang designed a preprocessing script called `xcp.py` (`http://code.google.com/p/xcp/`), it simply wrap all the text in CJK with a macro to switch font. This solution is not very convenient but worked amazingly well for most Chinese documents, yet does not impose much performance penalty.

Since more and more users are requesting this feature, Jonathan Kew and Miyata Shigeru worked together to implement it as a build-in of XeTeX. Basically, you can assign a "character class" to each Unicode code point with `\XeTeXcharclass`, then define the command to invoke when switching between two specific class with `\XeTeXinterchartoks`.

With the sample code provided by Miyata Shigeru, Yin Dian implemented a package called zhspacing (`http://code.google.com/p/zhspacing/`), which has complete support to assign and switch fonts automatically for Chinese documents. zhspacing is a big step forward compare to `xcp.py`, and it is well documented in both English and Chinese.

xeCJK

Soon after the release of zhspacing, Sun Wenchang provided another package called xeCJK (`http://ctan.org/pkg/xecjk/`) with exactly the same purpose. xeCJK is more like a direct descendant of the CJK package by Werner Lemberg, users can port their CJK documents into xeCJK without much efforts, yet features like `CJKnumber` and `CJKpunct` remains compatible. xeCJK soon became the most popular solution for Chinese typesetting.

A typical document in xeCJK looks like this:

```
\documentclass{article}
\usepackage{xecjk}
\setmainfont{Palatino}
\setCJKmainfont{SimSun}

\begin{document}
你好，这里是一段中文内容。
\end{document}
```

While it is totally fine using xeCJK in your documents directly, it is still a bit inconvenient if you need to include other functions like captions ("Table of Contents", "Chapter", "Bibliography", and so on) in Chinese, punctuations in Chinese, hyperref with CJK bookmarks, etc. Different packages may have different options, which should be carefully tuned for each TeX engine and output drivers (pdfTeX / XeTeX, `dvipdfmx`, `dvips`).

ctex, ctex-kit and ctex-doc

To simplify this process, Wu Lingyun initiated a higher level package called ctex (`http://ctan.org/pkg/ctex/`), which contains several document classes and macro packages designed for Chinese typesetting with LaTeX.

ctex has multiple backends to support traditional CJK and pdfTEX based engine, or dvipdfmx based PDF output, while it also supports xeCJK and XHLATEX based engine. Fonts are also pre-defined in several schemes like "winfonts" and "adobefonts". A typical ctex document looks like this:

```
\documentclass{ctexart}

\begin{document}
你好，这里是一段中文内容。
\end{document}
```

The most recent effort of CTEX community is a project called ctex-kit (http://code.google.com/p/ctex-kit/), we organized all Chinese related TEX packages into a code repository on Google code, then we package and submit these packages to CTAN regularly. With this totally open source approach, these packages are now maintained in a more active state, and users can retrieve them easily from CTAN or through the package manager of TEX Live 2009.

Besides ctex-kit, there is another project called ctex-doc (http://code.google.com/p/ctex-doc/), which organized all of the translations (lshort-cn for instance) or original documents related to TEX, LATEX, ConTEXt and graphics tools like METAPOST.

Conclusion

With the collaboration by the whole Chinese TEX community, we have every reason to expect an even more bright future for Chinese TEX typesetting.

Summary: Chinese TEX Typesetting: Past and Present

The article introduces and overviews Chinese TEX typesetting from its early beginnings to the present day.

Keywords: CCT, TianYuan, CJK, CTEX, XHTEX, xcp.py, zhspacing, xeCJK, ctex, ctex-kit, ctex-doc, Chinese typesetting, Google code.

Jjgod Jiang, gzjjgod@gmail.com
6 South Kexueyuan Road
Beijing, 100080, China

Sudoku s vepsanými kandži: integrace čínských glyfů s grafikou na úrovni METAPOSTu

Denis Roegel

Abstrakt

Článek představuje metodu, jak lze technicky za pomoci METAPOSTu vykreslit sudoku s čínskými glyfy (kandži). Makra nejsou rozsáhlá a jsou v úplné podobě součástí tohoto článku.

Na vykreslení sudoku jen s arabskými číslicemi můžete použít balíček sudoku; na generování, vykreslení a řešení sudoku nahlédněte, prosím, na balíček sudokubundle. Oba balíčky jsou dostupné z CTAN.ORG.

Sudoku, česky též magický čtverec, je logická hra, kterou vymyslel Howard Garns v roce 1979. Své obliby se dočkala především v Japonsku, ale prakticky se hraje po celém světě. Cílem hry je doplnit chybějící čísla 1 až 9 v předem dané předvyplněné tabulce. K dříve vyplněným číslům je potřeba dopsat další čísla tak, aby platilo, že v každé řadě, v každém sloupci a v každém z devíti čtverců byla použita vždy všechna čísla jedna až devět. Pořadí čísel není důležité. Čísla se nesmí opakovat v žádném sloupci, řádku nebo ve vnitřních malých čtvercích o devíti polích. Kdo by si to rád zkusil, nechť navštíví např. `http://profuvsvet.ic.cz/data/sudoku.html` nebo `http://sudokuonline.cz/`.

Přehledně o typech sudoku, jejich algoritmech a matematickém pozadí, viz např. Wikipedie [1]. Za zajímavost stojí, že první mistrovství světa v sudoku v roce 2006 vyhrála Češka Jana Tylová z Ústí nad Labem. Turnaj v sudoku můžete absolvovat např. na mezinárodním festivalu Czech Open. Ustavující Guinnessův rekord drží z 23. února 2008 Pavel Jaselský z Vojtovce na Slovensku.

Klíčová slova: METAPOST, hra sudoku, balíček CJK, balíček latexmp, kódování UTF-8, čínština, čínské glyfy, kandži.

doi: 10.5300/2010-3/220

1. Úvodem

Není tomu zas tak dávno, co jsem potřeboval ke svému povídání o čínském kalendáři využít výborný balíček Wernera Lemberga CJK. K této příležitosti jsem chtěl připravit obrázky v METAPOSTu s čínským značením. Práce to byla téměř bezproblémová.

This article is a translation of the article "Kanji-Sudokus: Integrating Chinese and Graphics", which first appeared in *TUGboat*, Volume 29 (2), pp. 317–319, 2008. Reprinted with permission. Translation and Czech abstract by Pavel Stříž. The author took the opportunity of this translation in order to make a few minor changes for the sake of clarity, on the suggestion of the translator.

O té bezproblémovosti to ve skutečnosti není zcela pravda, pravdou však je, že v současnosti je balíček CJK stále lépe a lépe zabudováván do verzí TEX Live. Sazba dokumentů s čínštinou, japonštinou a korejštinou (ČJK) se stává v rychle se měnícím TEXovém světě téměř nadpozemskou záležitostí. Zdaleka tomu tak nebylo v polovině roku 2007. Stále je potřeba nainstalovat různé, řekněme, linuxové balíčky a i poté hrozí, že nám drobná, ale nenahraditelná záplata chybí. Například se podívejme na mou verzi Ubuntu z poloviny roku 2008. TEX Live nebyl v té době tak kompletní jako dnes, chyběla mu korejská písma a byl jsem nahraný. Počítám však s tím, že to je již zapomenutá bolavá minulost a tato konkrétní situace již byla dávno vyřešená.

Abych shrnul své postřehy k polovině roku 2008, tak za pomoci TEX Live 2007, možná s několika doplňkovými linuxovými balíčky, a zároveň s poslední verzí editoru `Emacs` jste naprosto soběstační k přípravě prvotřídních dokumentů s ČJK. Sazba ČJK se stala ještě snazší záležitostí, protože téměř vše můžeme sepisovat v kódování UTF-8, bez potřeby využívat konverzí vstupního souboru, např. pomocí maker editoru `Emacs`. Tento postup se používal ke generování souboru `.cjk`, který mohl být následně LATEXem zpracován. Teď je celý proces ještě rychlejší; soubor, který připravíte, je zároveň tím, který se zpracovává.

Pro tvorbu obrázků přes METAPOST bylo toto zpracování ČJK připraveno taktéž příjemněji. Ještě donedávna, když jsme chtěli zahrnout čínštinu do METAPOSTu, tak jsme museli vygenerovat soubor `.cjk`, ten však bohužel nemohl být přímo METAPOSTem zpracován. Do souboru `.cjk` se muselo drobně zasáhnout, protože makra `Emacs`u formát METAPOSTu nebrala v úvahu.

Situace se mohla řešit na úrovni maker `Emacs`u, nu, ale ve skutečnosti, když už je tento druh konverzí do `.cjk` souboru téměř kompletně zapomenutou historií, tak i tento druh problému je bezpředmětný. Tedy má rada je nejen vstoupit do světa Linuxu, TEX Live v poslední verzi, ale také psát dokumenty s ČJK v kódování UTF-8. Šlape to jak švýcarské hodinky!

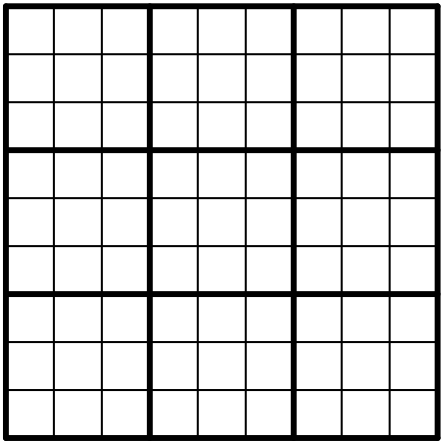
2. Motivační ukázka

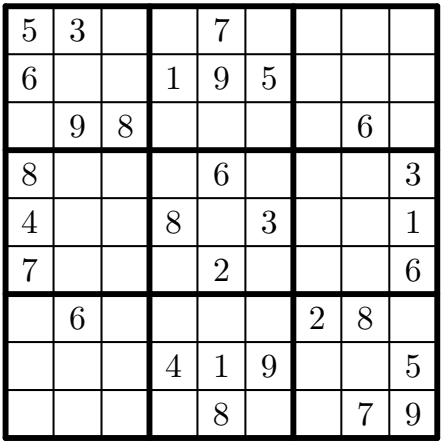
Osvětlím vám technické pozadí integrace čínštiny a METAPOSTu na následující drobné ukázce. Nakreslím si hrací pole s mřížkou, v polích však ne s arabskými číslovkami, ale s čínskými glyfy. Číslovky jsou následující 一 (1), 二 (2), 三 (3), 四 (4), 五 (5), 六 (6), 七 (7), 八 (8) a 九 (9).

Prázdná mřížka (ve všech proměnných `sol` nastavit nuly a v makru `fillgrid` přepnout `false` na `true`) a typická ukázka zadání ze hry sudoku vypadá jako následující příklad (tento konkrétní problém je převzat z Wikipedie [1]).

2.1. Vykreslení mřížky

Zadání sudoku může být nakresleno v METAPOSTu takto:






```
beginfig(1);
  string sol[];
  drawgrid(1.5pt,.5pt);
  % První řádek je ve spodní části mřížky.
  % Poslední řádek je naopak v její horní části.
  sol1="000080079"; sol2="000419005";
  sol3="060000280"; sol4="700020006";
  sol5="400803001"; sol6="800060003";
  sol7="098000060"; sol8="600195000";
  sol9="530070000";
  fillgrid(sol)(false);
endfig;
```

Makro `drawgrid` je přímočaře naprogramované a vykreslí potřebné horizontální a vertikální linky (s parametrem `u` nastaveným například na 1 cm):

```
def drawgrid(expr tha,thb)=
  pickup pencircle scaled thb;
  for i=0 upto 9:
    draw (i*u,0)--(i*u,9u);
    draw (0,i*u)--(9u,i*u);
  endfor;
  pickup pencircle scaled tha;
  for i=0 upto 3:
    draw (3i*u,0)--(3i*u,9u);
    draw (0,3i*u)--(9u,3i*u);
  endfor;
enddef;
```

2.2. Vyplnění polí

Předtím, než se nám podaří vypsat hodnotu do políčka, si musíme zajistit posun do pozice `(i,j)`, kde `i` je pořadové číslo sloupce a `j` je hodnota řádku. Počáteční hodnota je v obou směrech nastavena na jedničku a počátek tvoří levá spodní buňka. Následuje příprava makra, které si načte argumenty `i`, `j` a popisek umístí do středu příslušné buňky. V našem konkrétním případě je popisek zvětšen na 200 %, s tím, že aktuální změna závisí na velikosti hracího pole a velikosti písma.

```
def pos(expr i,j,l)=
  label(l scaled 2,((i-.5)*u,(j-.5)*u));
enddef;
```

2.3. Vysázení hodnot

Abychom zajistili vysázení, řekněme, hodnoty tři na pozici (2,9), můžeme zapsat:

```
pos(2,9,btex 3 etex);
```

My samozřejmě budeme chtít situaci zevšeobecnit a vysázet jednotlivé hodnoty z textového řetězce `sol`. Touto úvahou mohou některé nástroje a balíčky nabídnout řešení, stejně tak i potenciální problém. Abychom této situaci s přebráním problémů jiných nástrojů předešli, rozšiřme naši úvahu na:

```
pos(2,9,TEX(s));
```

kde `s` je textový řetětec k načtení makrem `TEX`. Zmíněné makro je načteno balíčkem `TEX`:

```
input TEX;
```

Vypadá to hezky, ale bohužel tento postup není příliš vhodný, důvod je ten, že TEX je celkově volán jedenaosmdesátkrát. A vedle toho nemáme stále zajištěnu podporu čínských glyfů, jakmile dojde na její volání. Hledejme ještě chvíli postup, který by byl více použitelný a lépe nám vyhovoval.

To, co jsme usilovně hledali, je balíček METAPOSTu nazvaný latexmp. Dříve zmíněný popisek na pozici (2,9) zapíšeme takto:

```
pos(2,9,textext(s));
```

Jednou z výhod balíčku latexmp je, že potřebuje jen dvě spuštění LaTeXu místo toho, aby si jej volal pro každý popisek zvlášť. Balíček navíc umožňuje snadné načtení dalších LaTeXových balíčků, především těch pro čínštinu. Dostáváme se k tomu, jak by mohly vypadat první řádky zdrojového kódu v METAPOSTu:

```
input latexmp;
setupLaTeXMP(class="article",
  packages="CJKutf8",
  preamble=(
    "\let\N\newcommand"
```

```
    &"\N\0{}\N\1{一}\N\2{二}\N\3{三}"
    &"\N\4{四}\N\5{五}\N\6{六}\N\7{七}"
    &"\N\8{八}\N\9{九}"
    &"\AtBeginDocument{"
    &   "\begin{CJK}{UTF8}{bsmi}}"
    &"\AtEndDocument{\end{CJK}}"));
```

Funguje to tak, že si hlavička načte balíček CJKutf8, který potřebujeme na práci se vstupními soubory formátu UTF-8. Následuje definování příkazů `\0` (prázdné pole), `\1` (čínská číslovka 1), `\2` (čínská číslovka 2), postupně až do příkazu `\9`.

Pro tyto číslovky zařídíme načtení nezbytného prostředí pro ČJK hned ze začátku těla dokumentu takto:

```
\AtBeginDocument{
  \begin{CJK}{UTF8}{bsmi}}
```

Otevřené prostředí LaTeXu uzavřeme obdobně:

```
\AtEndDocument{\end{CJK}}
```

2.4. Složení dílčích částí do fungujícího celku

V této chvíli máme připravenou definici jednotlivých buněk, můžeme si vykreslit hrací pole v podobě mřížky a máme připravená makra na čínské číslovky. Co nám zbývá? Ještě bychom potřebovali dvě maličkosti.

Za prvé bychom si přáli, abychom mohli vykreslit zadání versus řešení. Pro naše účely je zadání jen pole hodnot buněk, kdy některé jsou rovny nule. Hodnoty nula budou vypadat jako prázdné buňky. Když už jsme u tohoto přepínače, tak bychom si přáli, aby se nenulové hodnoty zobrazily buď v podobě arabských číslic nebo právě číslic čínských.

Naše zavedená konvence na hodnoty buněk je vskutku jednoduchá, neboť použijeme `4`, chceme-li výstup v podobě arabských číslic, nebo `\4`, chceme-li číslice čínské. Je to jedna z možností. Pak už si jen dáváme pozor, máme-li před číslovkou zpětné lomítko, či nikoliv. Zvláštní pozornost věnujme nule, protože nula u arabských číslic bude prázdné pole, ovšem u čínských číslic může a nemusí být pole vyplněno glyfem pro nulu. Záleží na stylu.

Přepínač je naprogramován pod makrem `zerospace`. Makro si bere znak `s` a nahradí jej mezerou, jen pokud se jedná o nulu a režim není nastaven na čínské číslovky.

```
def zerospace(expr chinese,s)=
  if not chinese and (s="0"): " " else: s fi
enddef;
```

Dostáváme se do finále, neboť mřížka je vyplněna makrem `fillgrid`. Nechť je prvním parametrem název pole s textovým řetězcem a druhý nechť je přepínač mezi číslovkami čínskými a arabskými. Příkaz `substring` je použit na oddělení jednotlivých znaků.

```
def fillgrid(text grid)(expr chinese)=
  for i=1 upto 9: for j=1 upto 9:
    pos(j,i,textext(if chinese: "\" & fi
         zerospace(chinese,
           substring(j-1,j) of grid[i])));
  endfor; endfor;
enddef;
```

Výsledky naší práce následují. První obrázek je zadání sudoku (levý obrázek) a druhý je jeho jednoznačné řešení (pravý obrázek), vše se zapnutým režimem pro čínštinu.

五	三			七				
六			一	九	五			
	九	八					六	
八				六				三
四			八		三			一
七				二				六
	六					二	八	
			四	一	九			五
				八			七	九

五	三	四	六	七	八	九	一	二
六	七	二	一	九	五	三	四	八
一	九	八	三	四	二	五	六	七
八	五	九	七	六	一	四	二	三
四	二	六	八	五	三	七	九	一
七	一	三	九	二	四	八	五	六
九	六	一	五	三	七	二	八	四
二	八	七	四	一	九	六	三	五
三	四	五	二	八	六	一	七	九

3. Závěrem

Příklad nám ukazuje, jak je v současnosti snadné včlenit čínštinu do METAPOSTu. Zbývá nám již jediné, a to napojit na představená makra dostatečně obecného řešitele sudoku. To nechť zůstane čtenářům jako otevřený a v článku neřešený problém.

Seznam literatury

[1] Wikipedia, the free encyclopedia. [Wikipedie, otevřená encyklopedie.] [online cit. 15. 1. 2010] Typy sudoku, o algoritmech a matematickém pozadí. Dostupné z webových stránek na URL:
V angličtině: http://en.wikipedia.org/wiki/Sudoku
V angličtině: http://en.wikipedia.org/wiki/Algorithmics_of_sudoku
V angličtině: http://en.wikipedia.org/wiki/Mathematics_of_Sudoku
V češtině: http://cs.wikipedia.org/wiki/Sudoku

Summary:
Kanji-Sudokus: Integrating Chinese and Graphics

The article illustrates the integration of Chinese and METAPOST on a small example. The author draws a Sudoku grid, not with Hindu-Arabic numerals, but with Chinese numerals. This article demonstrates how straightforward the integration of Chinese and METAPOST has become.

The source codes are included as small parts in the article commented in detail. You may find the original English version of the article in *TUGboat*, see http://tug.org/TUGboat/Articles/tb29-2/tb92roegel.pdf.

Keywords: METAPOST, the game of sudoku, CJK package, latexmp package, UTF-8, Chinese glyphs, kanji.

Denis Roegel, roegel@loria.fr
http://www.loria.fr/~roegel
LORIA – Campus Scientifique, BP 239
F-54506 Vandœuvre-lès-Nancy Cedex, France

Balíček sudokubundle

Peter Wilson

Abstrakt

Zdrojové kódy v balíku SUDOKU BUNDLE poskytují sadu maker pro zobrazování, řešení a generování her Sudoku. Tento článek popisuje některé pohledy na algoritmy a fungování těchto maker.

Klíčová slova: Sudoku, balíček sudokubundle.

doi: 10.5300/2010-3/227

Introduction

I developed the SUDOKU BUNDLE in response to a challenge presented in the PracTEX journal. It is available from CTAN.ORG with a complete User Manual [8] and is also described in the *LaTeX Graphics Companion* [3, Chapter 10].

In December 2005 the *PracTEX Journal* [2] set a competition about Sudoku puzzles. Depending on their experience with TEX, contestants were asked to (a) typeset a particular puzzle, (b) typeset a puzzle described in a 'Sudoku' file, (c) create a solver for Sudoku puzzles. I entered the competition with a printer and solver. Following from this it was no great effort to develop a matching Sudoku puzzle generator. These form the SUDOKU BUNDLE.

A Sudoku puzzle consists of a 9 by 9 grid of cells with some of the cells containing a number between 1 and 9, such as is shown in Figure 1. The problem is to place a number between 1 and 9 in each cell such that no number appears more than once in each row and in each column and in each minor 3 by 3 grid. The solution to the example puzzle is shown later in Figure 4 on page 237. The puzzle and answer have been typeset using the PRINTSUDOKU package.

Among many other sources the *Sudoku Online* [5] website provides much information on Sudoku puzzles and their solutions, as does the *Sudoku Solver by logic* website [6].

A Sudoku puzzle may be represented as a simple text file consisting of nine rows of numbers and dots, nine numbers and dots in each row. The numbers are the clues to the puzzle and the dots represent blanks in the grid. A Sudoku file for the example puzzle is given in Figure 2.

A harder puzzle is in Figure 3. You may like to try and solve it. Whether you do or not, the solution as determined and displayed by the SUDOKU BUNDLE is towards the end of the article in Figure 6 on page 240.

		4	8	3			7	2
	1	2					8	
		5	2		1	3		
				6	2		9	1
7			5		9			3
9	4		7	8				
		3	9		7	4		
	5					6	1	
	8			4	6	9		

Figure 1: Example of simpler Sudoku puzzle

```
..483..72
.12....8.
..52.13..
....62.91
7..5.9..3
94.78....
..39.74..
.5....61.
.8..469..

Example of simpler puzzle
(anything can come after the nine puzzle lines)
```

Figure 2: A Sudoku file for the example simpler puzzle (centered)

	3		7			2	9	
2	5	8			1	7		
					5			
		9				8		
			4	2	3			
		2				3		
			8					
		5	6			9	3	7
	9	6			4		8	

Figure 3: A harder puzzle

The SUDOKU BUNDLE only handles sudoku puzzles that consist of 9 by 9 arrays of the numbers 1 through 9. Other puzzles, such as those consisting of 16 by 16 arrays of numbers and letters are outside the scope of this paper.

The SUDOKU BUNDLE consists of three packages:

1. PRINTSUDOKU which prints a puzzle that is contained in an external file or writes out a file that is specifed by a macro in the document.
2. SOLVESUDOKU which solves puzzles up to a certain level of difficulty; it requires the PRINTSUDOKU package to read the puzzle from a file and print it and also to print the solution and write it out to an external file.
3. CREATESUDOKU which generates puzzles that can be solved by SOLVESUDOKU; it requires the SOLVESUDOKU package.

I am not going to describe the packages in detail as you can find that elsewhere [8]. Rather, I shall discuss some of the algorithms that I used in the packages and how I implemented them in LaTeX.

Typesetting

This is the province of the PRINTSUDOKU package, which provides two basic functions:

1. The `\sudoku{`⟨*file*⟩`}` macro reads a Sudoku game from ⟨*file*⟩ and typesets the grid and clues;
2. The `\writepuzzle{`⟨*line1*⟩`}{`⟨*line2*⟩`}...{`⟨*line9*⟩`}[`⟨*text*⟩`]` macro writes the nine lines of a puzzle to the `\puzzlefile` external file, where the default file is: `\newcommand*{\puzzlefile}{puzz.sud}`

The code for `\writepuzzle` is pretty simple, just opening an output file and writing the 9 arguments to the file a line at a time:

```
\newwrite\s@dwrite
\newcommand*{\writepuzzle}[9]{%
  \immediate\closeout\s@dwrite
  \immediate\openout\s@dwrite=\puzzlefile
  \immediate\write\s@dwrite{#1}%
  ...
  \immediate\write\s@dwrite{#9}%
  \writes@dpuzzend}
```

`\writes@dpuzzend` provides an option to write text after the puzzle data is written, with `\sudpuzznewline` as a new line macro (`\\` will not work here).

```
\newcommand*{\writes@dpuzzend}[1][\@empty]{%
  \ifx\@empty #1\else
    \immediate\write\s@dwrite{ }% a blank line
    \immediate\write\s@dwrite{#1}%
  \fi
```

```
  \immediate\closeout\s@dwrite}
\newcommand*{\sudpuzznewline}{^^J}
```

The \writepuzzle and \writes@dpuzzend pair of macros is a particular instance of a general technique for making it appear that a macro takes more than the 9 argument limit imposed by TEX.

More complicated is the code for the \sudoku{⟨*file*⟩} macro, which reads a puzzle from the ⟨*file*⟩ and typesets it.

```
\newcommand*{\sudoku}[1]{%
  \reads@dgame{#1}% open the file for reading
  \s@dgame}
```

The macro \s@dgame uses the picture environment to draw the grid. It then inserts the clues into the grid.

```
\newcommand*{\s@dgame}{%
  \setlength\unitlength\halfs@dcell % units of half cell size
  \begin{picture}(18,18)(0,-18)
%% code to draw the grid
  \adds@dclues
  \end{picture}}
```

The macro \adds@dclues (not shown) reads the puzzle file line by line, each time setting \firsts@dcluetrue, and then for each line calls \dos@dcols to insert its clues into the grid.

```
\newcommand*{\dos@dcols}{%
\bgroup
  \loop%        over the 9 clues
    \ifnum\s@dncol<10\relax
%% calculate grid location coordinates (\s@dcolpos,-\s@drowpos)
    % put the clue into the grid
    \put(\s@dcolpos,-\s@drowpos){\makebox(0,0){\gets@dclue}}%
    \advance\s@dncol 1\relax% increment clue/column count
  \repeat
\egroup}
```

The macro \gets@dclue retrieves the next clue (character) from the line of clues and presents it for printing. To do this it uses \splitoff{⟨*string*⟩}, which gets the next character in a string, making it available as \istchar and leaves the remainder of the string as \restchars. The technique is based on TEX's delimited arguments [4, chapter 10]. I have talked about this in more detail in two of my *Glisterings* columns [7, 9], the second of which also contains a long example of using the \loop...\repeat construct.

```
\def\gettwo#1#2\nowt{%
  \gdef\istchar{#1}\gdef\restchars{#2}}
\def\splitoff#1{\gettwo#1\nowt}
```

Finally, this is the code for `\gets@dclue`. If the clue is a number, `\gets@dclue` provides it for printing or if it is a '.' then `\gets@dclue` skips on. At the beginning the string is the line as read from the file (`\s@dline`); after that the string is `\restchars`.

```
\gdef\s@dfstop{.}
\newcommand*{\gets@dclue}{%
  \iffirsts@dclue%  initially set by \adds@dclues
    \expandafter\splitoff\expandafter{\s@dline}%
    \global\firsts@dcluefalse
  \else
    \expandafter\splitoff\expandafter{\restchars}%
  \fi
  \ifx\s@dfstop\istchar%  a '.' return nothing
  \else%                  return clue number
    \istchar
  \fi}
```

Solving

The `\sudokusolve{`⟨*file*⟩`}` macro in the SOLVESUDOKU package attempts to solve the puzzle contained in the ⟨*file*⟩. It first prints the puzzle as specified in ⟨*file*⟩, then solves it as best it can, and lastly typesets the (partial) solution.

The following facts are used to generate a solution.

1. Initially the potential solution for any cell is in the set of digits 1...9.
2. In a solved puzzle a digit must be unique within its row, its column, and its 3 by 3 block. So, if a solution, say N, is known for a cell, then N can be deleted from all the potential solutions in the other cells of the row, column and block. I have called this a *simple reduction*.
3. If among all the cells in a row (column, block) there is a digit that occurs only once among all the potential solutions, then that digit is the solution for its cell. I have termed this a *loner*.
4. If among all the cells in a row (column, block) there are two digits which occur only twice in the potential solutions, each time as a pair (e.g., 39 and 39), then one or other of the two digits must be a solution for a cell in which the pairs occur. This means that the two digits cannot occur anywhere else in the row (column, block) and thus can be eliminated from all the other potential solutions. I call this *pair reduction*.
5. There are other facts which are more difficult to apply and I have not considered them because of their complications and the difficulty of embodying them in LaTeX code.

The solution procedure is:

1. Populate the puzzle grid by assigning to each cell either the clue (digit) given in the puzzle ⟨*file*⟩ or the set of potential solutions when the puzzle provides no value.
2. For each clue perform a simple reduction, which may produce loners.
3. Continue the simple reductions until there is no change in the solution state. This is either because the full solution has been obtained or that more sophisticated methods are needed.
4. Examine the partial solution for pairs and if one is found perform the pair reduction. After a pair reduction go back and look for loners (and subsequent simple reductions (and subsequent pair reductions). At the end either a complete solution is found or the solver gives up.
5. The process stops when either all 81 cells have been solved or there is no change in any potential solution after going through all the reductions.

The major problem was in deciding on a convenient datastructure for the problem. In the end I used a 9 digit 'binary solution set' for the representation of a cell's potential solution (e.g., [111111111] ⇔ 123456789 and [101010101] ⇔ 13579). The solution, say N, for a cell is represented as the 'set' $[-N]$; that is, for example, a potential solution '3' is represented as [001000000] and the actual solution '3' is represented as $[-3]$. I will use the term *9-set* to indicate a set with a maximum of 9 members, where a member is a digit d in the range $1 \le d \le 9$.

Following from this was the question of how to implement the datastructure? There are 81 cells in the Sudoku grid and I needed to maintain a potential or actual solution for each cell. It was convenient to use a `\count` for each cell's solution set which was accessible via the cell's number (1...81).

```
\newcommand*{\newknt}[1]{\expandafter\newcount\csname #1\endcsname}
\newcommand*{\useknt}[1]{\csname #1\endcsname}
```

`\newknt{`⟨*id*⟩`}` creates a new `\count` called ⟨*id*⟩, where ⟨*id*⟩ can include analphabetic characters (like digits), and `\useknt{`⟨*id*⟩`}` expands to the ⟨*id*⟩ `\count` created previously by `\newknt`.

```
% make the potential solution sets
\newcommand*{\makesudsets}{%
  \global\s@lcnta=1\relax
  \loop
    \ifnum\s@lcnta<82\relax
    \newknt{s@lans\the\s@lcnta}%
    \global\useknt{s@lans\the\s@lcnta}=111111111\relax
    \advance\s@lcnta 1\relax
  \repeat}
```

`\makesudsets` creates 81 `\count`s named `\s@lans1` through `\s@lans81` and sets them all to 111111111.

Now some macros are needed to manipulate a 9-set. These are principally based on the fact that TEX only provides integer arithmetic. For instance, with integer arithmetic

$$19/10 = 1,\ 20/10 = 2 \text{ and } 21/10 = 2,$$

which is a method for getting the first digit of a two-digit number (and similarly for numbers with more digits). Further,

$$(19/10) \times 10 = 10 \text{ while } 19 - (19/10) \times 10 = 9$$

which provides a method for obtaining the last digit of a two-digit number. As a more complicated example, to determine the number of thousands in a number, say 13247546 where the answer is 7,

$$\begin{aligned} (13247546/1000) &= 13247 \\ (13247/10) \times 10 &= 13240 \\ 13247 - 13240 &= 7 \end{aligned}$$

`\settonum{`⟨*set*⟩`}{`⟨*cnt*⟩`}` converts a potential binary solution 9-set ⟨*set*⟩ to the corresponding list of digits, e.g., `[11...1] -> 12...9`. The result is assigned to the `\count` ⟨*cnt*⟩ which must be supplied by the calling macro. If the set is negative then the result is that number (e.g., `[-3] -> -3`). If the set contains only a single non-zero entry, that is converted to the negative of the corresponding digit (e.g., `[100] -> -7`).

```
\newcommand*{\settonum}[2]{%
  \settonumcnt=#1\relax
  \tempcnty=0\relax
  \tenscnt=1\relax
  \ifnum\settonumcnt<0\relax % just return the number
    \tempcnty=\settonumcnt
    #2=\tempcnty
  \else
    \ifodd\settonumcnt  % set is [ddddddd1] so 9 flagged
      \tempcntz=9\relax
      \multiply\tempcntz \tenscnt
      \advance\tempcnty by \tempcntz
      \multiply\tenscnt 10\relax
    \fi
    \divide\settonumcnt by 10\relax % set reduced to [ddddddd]
    \ifodd\settonumcnt  % reduced set is [ddddddd1] so 8 flagged
      \tempcntz=8\relax
      \multiply\tempcntz \tenscnt
      \advance\tempcnty by \tempcntz
```

```
    \multiply\tenscnt 10\relax
  \fi
  \divide\settonumcnt by 10\relax % set reduced to [ddddddd]
  \ifodd\settonumcnt  % reduced set is [dddddd1] so 7 flagged
    ...
  \ifodd\settonumcnt  % reduced set is [1] so 1 flagged
    \tempcntz=1\relax
    \multiply\tempcntz \tenscnt
    \advance\tempcnty by \tempcntz
  \fi
  \ifnum\tempcnty<10\relax
    \ifnum\tempcnty>0\relax % single digit
      \tempcnty = -\tempcnty
    \fi
  \fi
  #2=\tempcnty
\fi}
```

`\numofnuminset{`⟨*dig*⟩`}{`⟨*set*⟩`}{`⟨*cnt*⟩`}` sets the `\count` ⟨*cnt*⟩ to the number of times the digit ⟨*dig*⟩ is represented in the 9-set ⟨*set*⟩. For example the number of the digits in the 9-set `[200000013]` are `1->2, 2->0, ..., 8->1` and `9->3`.

```
\newcommand*{\numofnuminset}[3]{%
  \tempsetctr=#2\relax
  \tempsetansrctr=\tempsetctr
  \ifnum\tempsetctr<0\relax % a solution, not a set
    \tempsetansctr=0\relax
  \else
    \ifcase #1\relax
    \or     % 1
      \divide\tempsetansctr by 100000000\relax
    \or     % 2
      \divide\tempsetansctr by 10000000\relax
      \tmpsetctr=\tempsetansctr
      \divide\tmpsetctr 10\relax \multiply\tempsetctr 10\relax
      \advance\tmpsetansctr -\tmpsetctr
    \or    % 3
      \divide\tempsetansctr by 1000000\relax
      \tmpsetctr=\tempsetansctr
      \divide\tmpsetctr 10\relax \multiply\tempsetctr 10\relax
      \advance\tmpsetansctr -\tmpsetctr
    \or
      ...
    \or   % 9
      \tmpsetctr=\tempsetansctr
      \divide\tmpsetctr 10\relax \multiply\tempsetctr 10\relax
```

```
      \advance\tmpsetansctr -\tmpsetctr
    \else % error
      \tmpsetansctr=0\relax
    \fi
  \fi
  #3=\tmpsetansctr}
```

The macro `\deletenumfromset{`⟨*dig*⟩`}{`⟨*set*⟩`}{`⟨*cnt*⟩`}` removes the digit ⟨*dig*⟩ from the potential binary solution 9-set, putting the modified set in `\count` ⟨*cnt*⟩. If the digit was removed then the boolean `\ifsetchanged` is set TRUE.

```
\newcommand*{\deletenumfromset}[3]{%
  \global\setchangedfalse
  \tmpsetctr=#2\relax
  \tmpsetansctr=#2\relax
  \ifnum\tmpsetctr<0\relax% represents a solved number, do nothing
  \else
    \ifcase #1\relax
    \or%              1
      \divide\tmpsetctr by 100000000\relax
      \ifodd\tmpsetctr%  it's there
        \advance\temsetansrctr -100000000\relax
        \global\setchangedtrue
      \fi
    \or%             2
      \divide\tmpsetctr by 10000000\relax
      \ifodd\tmpsetctr%  it's there
        \advance\temsetansrctr -10000000\relax
        \global\setchangedtrue
      \fi
    \or
      ...
    \or%             8
      \divide\tmpsetctr by 10\relax
      \ifodd\tmpsetctr%  it's there
        \advance\temsetansrctr -10\relax
        \global\setchangedtrue
      \fi
    \or%             9
      \ifodd\tmpsetctr%  it's there
        \advance\temsetansrctr -1\relax
        \global\setchangedtrue
      \fi
    \fi
  \fi
  #3=\tmpsetansctr}
```

Given these macros it is now just a case of using them in a lot of tedious code to solve the puzzle using the procedure described earlier.

Going through the puzzle as presented, for any cell for which a clue is given, its potential binary solution set is replaced by the actual solution. After this initialisation, `\deletenumfromset` is used for the simple reductions.

The macro `\numofnuminset` is used to find loners — the potential binary solution 9-sets for the cells in a row (column, block) are added together and the result is then searched for a loner, which is a digit that occurs only once in the resulting 9-set summation.

The macro `\numofnuminset` is also used to search for pairs in a row (column, block) following a similar summation of the potential binary solution 9-sets.

`\settonum` is used to determine if a cell's 9-set has been reduced to a single digit, which is then a solution. It can alo be used in printing out the current status at any point during the solution process, showing for each cell either the solution or the list of potential solutions for that cell.

For further details look at the documented code — there's only about 1400 lines of it — for the SOLVESUDOKU package.

Towards the end of this article the solution to the puzzle presented in Figure 3 found by `\sudokusolve` is provided as Figure 6 on page 240. The figure is produced by the following code:

```
\begin{figure}
\centering
\cluefont{\normalsize}\cellsize{1.5\baselineskip}
\sudokusolve{cal4s4.sud}
\caption{The puzzle from \fref{fig:puz2} together with its solution
         as found and presented by \cs{sudokusolve}}
         \label{fig:ans2}
\end{figure}
```

where the `\cs` macro is defined as

```
\DeclareRobustCommand\cs[1]{\texttt{\char`\\#1}}
```

which is most useful if you ever need to typeset the name of a macro.

Generating

The CREATESUDOKU package lets you automatically create Sudoku puzzles of the kind that the SOLVESUDOKU package, which it uses, can solve. It also uses Donald Arseneau's RANDOM.TEX for generating random numbers [1].

The package requires a completely solved puzzle to start with, which can be either from a file that you provide, or it uses a default solved puzzle.

The starting grid is modified in a random manner. Within one of the three columns of blocks exchanging any two of the three cell columns alters the puzzle

6	9	4	8	3	5	1	7	2
3	1	2	6	7	4	5	8	9
8	7	5	2	9	1	3	6	4
5	3	8	4	6	2	7	9	1
7	2	6	5	1	9	8	4	3
9	4	1	7	8	3	2	5	6
1	6	3	9	5	7	4	2	8
4	5	9	3	2	8	6	1	7
2	8	7	1	4	6	9	3	5

Figure 4: Solution to the simpler example puzzle in Figure 1

but leaves it still as a valid result. For example columns 1 and 3 (in the first column of blocks) may be exchanged and columns 8 and 9 (in the third column of blocks) be exchanged and the result is still a solved grid. Similarly, within a row of blocks, exchanging any two of the three rows of cells changes the puzzle but leaves it as a valid result.

The default starting grid is shown in Figure 5. Check that it is a valid Sudoku solution and then try exchanging pairs of rows and columns, as described above, to check that the result, although different, is still a valid solution.

Row pairs and column pairs are exchanged in a random fashion a random number of times. At this point the grid is a complete solution. There is a macro that will randomly eliminate 17 numbers from the grid; a puzzle is ambiguous, that is it has more than one solution, if two numbers are completely absent from the grid. You can then get the package to delete particular numbers, rows, columns, blocks, or diagonals from this grid.

When given its head the package writes the puzzle to an external file `\prevfile` and then uses `\sudokusolve` to try and solve the puzzle. If it can not find a solution then you would have to go back and try again, eliminating fewer and/or different clues. If `\sudokusolve` can solve the puzzle the package randomly eliminates a clue, writes the revised puzzle to another external file (`\currfile`) and uses `\sudokusolve` to try and solve the new puzzle. If it can then the

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	3	4	5	6	7	8	9	1
5	6	7	8	9	1	2	3	4
8	9	1	2	3	4	5	6	7
3	4	5	6	7	8	9	1	2
6	7	8	9	1	2	3	4	5
9	1	2	3	4	5	6	7	8

Figure 5: Default initial puzzle for generation

`\currfile` puzzle is written into the `\prevfile`, another clue is randomly eliminated, and the process continues on. At the point where `\sudokusolve` fails, the last succesfully solved puzzle is that in the `\prevfile`. This is the puzzle that is presented as the newly created puzzle.

An interactive program

For checking how well the `\sudokusolver` was working I wrote a small interactive LaTeX program that asked for a Sudoku puzzle file to solve, tried to solve it, and kept on asking for more files until I effectively said 'no more'.

Here is a version of it, which I have called 'solvem'. To use it simply call:
`pdflatex solvem`
and a session will look like this, where, for exposition purposes, the user's commands/responses are typeset in *this font* and `solvem`'s are in `this font`, and the user wants solutions to the puzzles in the puzzle files `cal4s4.sud` and `st123.sud`:

pdflatex solvem
`New file?  y/n`
`\getans=`*y*
`Enter the file name`

```
\sudfile=cal4s4.sud
...progress report on the solution...
New file?  y/n
\getans=y
Enter the file name
\sudfile=st123.sud
...progress report on the solution...
New file?  y/n
\getans=n
Output written on solvem.pdf
```

Here is the program:

```
% solvem.tex   Solve Sudoku
%              author Peter Wilson
\documentclass{article}
\usepackage{solvesudoku}
\newcommand*{\solvefile}[1]{%
\begingroup
  \sudokusolve{#1}%
  \par
  \vspace{\baselineskip}%
  Number of clues = \the\numcluesctr\ and difficulty = \the\difficultyctr.
\endgroup}
\def\yesans{y}
\begin{document}
\loop
  \typein[\getans]{New file? y/n}
  \ifx\yesans\getans
    \typein[\sudfile]{Enter the file name}r
    \IfFileExists{\sudfile}{%
    \clearpage
    \begin{center}\huge \sudfile\end{center}
    \solvefile{\sudfile}%
    }{\typeout{I can't find file \sudfile}}
\repeat
\end{document}
```

The command `\typein[`⟨*csname*⟩`]{`⟨*text*⟩`}` is a LaTeX macro that outputs ⟨*text*⟩ to the terminal and `.log` file and waits for some response text which it assigns to the command ⟨*csname*⟩ and you can then do something with `\csname`. It is an extended version of `\typeout{`⟨*text*⟩`}` which just outputs ⟨*text*⟩ to the terminal and the `.log` file.

Basically `solvem.tex` goes round a loop asking for a puzzle file and calls `\sudokusolve` to try and solve it. There is code to catch if a file specified by the

	3		7			2	9	
2	5	8			1	7		
					5			
		9				8		
			4	2	3			
		2				3		
			8					
		5	6			9	3	7
	9	6			4		8	

THE ANSWER

6	3	1	7	4	8	2	9	5
2	5	8	9	6	1	7	4	3
9	7	4	2	3	5	6	1	8
3	4	9	5	7	6	8	2	1
8	1	7	4	2	3	5	6	9
5	6	2	1	8	9	3	7	4
1	2	3	8	9	7	4	5	6
4	8	5	6	1	2	9	3	7
7	9	6	3	5	4	1	8	2

Figure 6: The puzzle from Figure 3 on page 228 together with its solution as found and presented by `\sudokusolve` (design of the output is slightly modified for the sake of the article)

user does not exist, in which case it asks for another one. After each solution it gives some information about how hard it was to solve (or not as the case may be) the puzzle. The puzzles and their solutions are written to the output file from `solvem.tex`, either `solvem.dvi` or `solvem.pdf`, depending on how `solvem` is called.

References

[1] Donald Arseneau. Generating random numbers in TEX, 1995. Available on CTAN.ORG in `macros/generic/misc/random.tex`.

[2] The Editors. Distractions: Sudoku. In *The PracTEX Journal*, Volume 1, Number 4, 2005. ISSN 1556-6994. Available at `http://tug.org/pracjourn/2005-4/distract/`.

[3] Michel Goossens, Frank Mittelbach, et al. *The LaTeX Graphics Companion*. Addison-Wesley, 2nd edition, 2008. ISBN 0-321-50892-0.

[4] Donald Knuth. *The TEXbook*. Addison-Wesley, 1986. ISBN 0-201-13448-0.

[5] Sudoku Online: Home of the Sudokulist. `http://www.sudoku.org.uk/`

[6] Sudoku Solver . . . by logic. `http://www.sudokusolver.co.uk/`

[7] Peter Wilson. Glisterings. In *TUGboat*, Volume 26, Number 3, pp. 253–255, 2005. ISSN 0896-3207. Available at `http://tug.org/TUGboat/Articles/tb26-3/tb84glister.pdf`.

[8] Peter Wilson. The sudoku bundle for displaying, solving and generating Sudoku puzzles, 2006. Available on CTAN.ORG in `macros/latex/contrib/sudokubundle/`.

[9] Peter Wilson. Glisterings: stringing along, loops. In *TUGboat*, Volume 28, Number 1, pp. 12–14, 2007. ISSN 0896-3207. Available at `http://tug.org/TUGboat/Articles/tb28-1/tb88glister.pdf`.

Summary: The sudoku bundle

The SUDOKU BUNDLE provides a coordinated set of packages for displaying, solving, and generating Sudoku puzzles. This article describes some of the internal aspects of the packages.

Keywords: Sudoku, sudokubundle package.

Peter Wilson, herries.press@earthlink.net
CSTUG c/o FEL ČVUT, Technická 2
Prague, CZ-166 27, Czech Republic

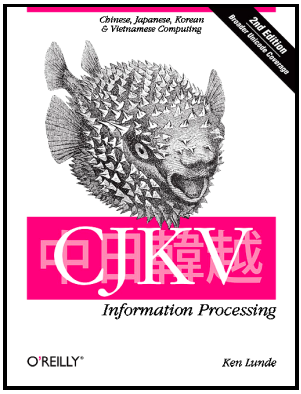
Nové a staronové knihy

Pavel Stříž, Michal Mádr

CJKV Information Processing

Chinese, Japanese, Korean & Vietnamese Computing

Ken Lunde



Vydání: druhé.
Počet stran: 864 a přílohy A–M.
Vazba: brožovaná.
Rok vydání: 2008.
Nakladatel: O'Reilly Media, Inc.
ISBN-10: 0-596-51447-6.
ISBN-13: 978-0-596-51447-1.
Informace od nakladatele:
`http://oreilly.com/catalog/9780596514471/`

Od svého prvního vydání před deseti lety se tato kniha stala bohatým zdrojem informací o zpracování textů v čínštině, japonštině, korejštině a vietnamštině (ČJKV). Kniha seznamuje tvůrce webových stránek a aplikací s technikami a nástroji pro šíření informací jihoasijskému publiku. V aktualizovaném druhém vydání si kniha navíc všímá nedávného významného vlivu kódování Unicode, XML a formátu fontů OpenType na zpracování východoasijských textů.

Řada příloh, zdrojových kódů, binárek a ukázek je umístěna na webové stránce prvního vydání `http://examples.oreilly.com/9781565922242/`. Ken Lunde byl tak laskav a napsal nám, že kvůli vytížení v práci nestihl ukázky k druhému vydání ještě aktualizovat.

Kniha vám výrazně pomůže:

- poznat písma a znaky jazyků ČJKV a metody jejich přepisu,
- nahlédnout do trendů a vývoje sad znaků a kódování, zvláště kódování Unicode,
- prozkoumat svět typografie, obzvláště záležitosti spojené s rozložením textu v jazycích ČJKV na stránce,
- poznat techniky zpracování informací (např. algoritmy konverze kódování) a aplikovat je pomocí různých programovacích jazyků,
- zpracovat texty v jazycích ČJVK na rozličných platformách pomocí různých textových editorů a textových procesorů,
- získat informace o ČJKV slovnících, slovníkovém softwaru, softwaru a službách strojového překladu,

- zvládnout zacházení s ČJVK obsahem i formou při publikování na papíře nebo na webu.

Pokud někdo ze čtenářů Zpravodaje zná, používá nebo studuje některý z jazyků ČJKV, pak tato kniha bude pro něj jako příchod druhých Vánoc. Pokud jste příznivci konverzí přes iconv, tak vás zahřeje existence nástroje JConv z roku 1993. Teď jen vyřešit otázku, který program byl na světě první, dodáváme s úsměvy.

Za zajímavost uvádíme, že kniha se na několika místech odkazuje na TEX, ovšem komplexní profil možností sazby ČJKV v TEXu zde nečekejte.

Autorem obou vydání knihy je Ken Lunde, seniorský IT pracovník vývoje písem ČJKV ve firmě Adobe Systems.

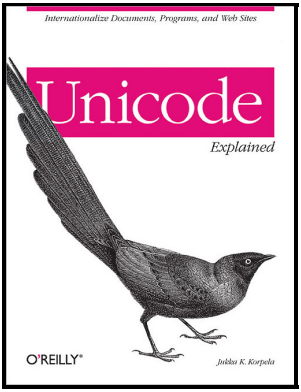
Obsah [Contents]

Přílohy [Appendixes]

Unicode Explained

Internationalize Documents, Programs, and Web Sites

Jukka K. Korpela



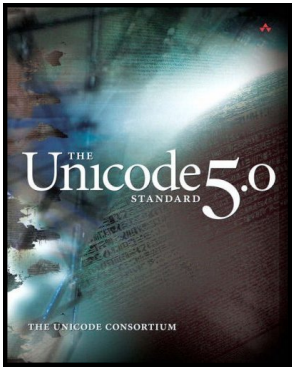
Vydání: první.
Počet stran: 688.
Vazba: brožovaná.
Rok vydání: 2006.
Nakladatel: O'Reilly Media, Inc.
ISBN-10: 0-596-10121-X.
ISBN-13: 978-0-596-10121-3.
Informace od nakladatele:
`http://oreilly.com/catalog/9780596101213/`

Jedná se o komplexní náhled na problematiku Unicode od finského autora.

Kniha je rozdělena do tří částí. V první části se autor zabývá prací se znaky [Working with Characters], následuje podrobný náhled na Unicode [A Systematic Look at Unicode] a závěrečná část se zabývá pokročilými oblastmi Unicode [Advanced Unicode Topics].

The Unicode Standard, Version 5.0

The Unicode Consortium



Vydání: páté.
Počet stran: 1472.
Vazba: vázaná.
Rok vydání: 2006.
Nakladatel: Addison-Wesley Professional.
ISBN-10: 0321480910.
ISBN-13: 978-0321480910.
Informace od nakladatele:
`http://www.awprofessional.ca/`

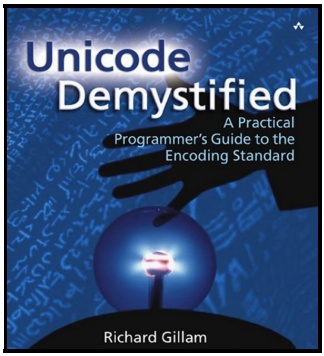
Jedná se o poslední knižní vydání standardů Unicode ve verzi 5.0. Tato kniha je oficiální referenční příručkou. Autory knihy jsou lidé ze společenství Unicode, což je nezisková organizace působící na celém světě.

Zájemce najde dostatek informací k úplně poslední verzi 5.2 na webových stránkách `http://www.unicode.org/versions/`. Nákresy glyfů a symbolů lze nalézt na `http://www.unicode.org/charts/`.

Unicode Demystified

A Practical Programmer's Guide to the Encoding Standard

Richard Gillam



Vydání: první.
Počet stran: 896.
Vazba: brožovaná.
Rok vydání: 2002.
Nakladatel: Addison-Wesley Professional.
ISBN-10: 0201700522.
ISBN-13: 978-0201700527.
Informace od nakladatele:
`http://www.awprofessional.ca/`

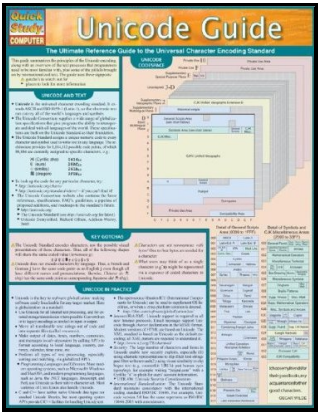
Tato kniha nahlíží na standardy Unicode z pohledu programátora, experta firmy IBM. Autor v knize představuje praktické techniky zpracování a lokalizaci textů, dělení slov, vyhledávání, třídění, možnosti vstupu, konverze a převody textů.

Postupně v knize odhalí integraci standardů s jinými technologiemi, jako jsou programovací a skriptovací jazyky Java, JavaScript, XML a HTML.

Unicode Guide

The Ultimate Reference Guide to the Universal Character Encoding Standard

Joe Becker, Richard Gillam, Mark Davis



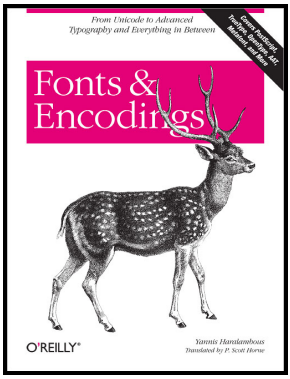
Vydání: první.
Počet stran: 6.
Vazba: brožura.
Rok vydání: 2006.
Nakladatel: Barcharts, Inc.
ISBN-10: 1423201809.
ISBN-13: 978-1423201809.
Informace od nakladatele:
`http://www.barcharts.com/`

Na závěr představujeme protiklad předchozích obřích knih, konkrétně sumář v podobě šesti zafóliovaných stran z produkce firmy BarCharts. Jedná se o zhuštěnou formu podkladů se vztahem ke standardům Unicode.

Fonts & Encodings od Yannise Haralambouse

Ulrik Vieth

Základní informace



Překlad knihy: P. Scott Horne.
Vydání: první.
Počet stran: 1040.
Vazba: vázaná.
Rok vydání: 2007.
Nakladatel: O'Reilly Media Inc.
ISBN-10: 0-596-10242-9.
ISBN-13: 978-0-596-10242-5.
Informace od nakladatele:
`http://www.oreilly.com/catalog/9780596102425`

O autorovi

Yannis Haralambous je v mezinárodní TEXové komunitě velmi dobře znám nejen jako spoluzakladatel projektu Omega, ale i pro svá četná přispění jako vývojář písem pro rozličné jazyky. Zdálo se tedy zcela příhodné, když se Yannis rozhodl napsat souhrnnou knihu na téma fontů a jejich kódování.

Původní vydání, nazvané *Fontes & Codages*, se objevilo v roce 2004, avšak pouze ve francouzském jazyce. Anglický překlad, *Fonts & Encodings, From Advanced Typography to Unicode and Everything in Between,* připravený P. Scottem Hornem, je k dispozici teprve krátce, po vydání nakladatelstvím O'Reilly.

Obsah

Rozsahem o trochu více než tisíc stran se kniha vyrovná velikosti *The LaTeX Companion* ve druhém vydání. Zdá se být velmi působivou, nejen svým pouhým rozsahem, ale i vzhledem k širokému rozpětí témat, jež pokrývá, jakož i hloubkou,

This review is a translation of the article "Book review: *Fonts & Encodings* by Yannis Haralambous", which first appeared in *TUGboat,* Volume 29, Number 2: pp. 331–332, 2008. The book review is available from `http://tug.org/TUGboat/Articles/tb29-2/tb92vieth.pdf`. Reprinted with permission. Translation by Marcel Svitalský.

s jakou je pokrývá, a mírou podrobnosti. V několika případech autor spotřebovává tucty stran, aby dokumentoval některé tajemné detaily písmových formátů, jež posud postrádaly souhrnnou nebo přístupnou dokumentaci z jiných zdrojů.

Kniha sestává z hlavní části o čtrnácti kapitolách v rozsahu kolem šesti set stran, následované přídavkem sedmi kapitol dosahujících dalších čtyři sta stran.

Autor objasňuje již v úvodu, že různé skupiny čtenářů mohou mít užitek z různých částí knihy, aniž by ji všichni museli číst celou: některé kapitoly se věnují především problematice kódování, zabývajíce se znaky na vstupu, jiné jsou převážně o písmech a věnují se glyfům na výstupu, zatímco ještě další kapitoly se nalézají někde uprostřed, kde se musejí zabývat jak písmy, tak kódováním zároveň. Některé kapitoly jsou přístupné koncovým uživatelům zajímajícím se o instalaci a užívání písem, kdežto jiné budou zajímat pouze návrháře písem nebo vývojáře softwaru vztahujícího se k písmům.

První část knihy počíná tématy týkajícími se kódování. Kapitola první poskytuje přehled historie kódování před příchodem Unicode, počínaje sedmibitovým ASCII a rozličnými osmibitovými kódováními ISO až k šestnáctibitovým kódováním východoasijských jazyků. Kapitoly druhá až čtvrtá pokrývají standard Unicode, přičemž nejprve začínají přehledem symbolů a skriptů zahrnutých do standardu a dále pokračují ke stále komplexnějším implementačním detailům. Kapitola pátá dovršuje tuto část předvedením některých užitečných nástrojů k užití Unicode vstupů na rozličných systémových platformách.

Druhá část se věnuje tématům správy písem na různých systémových platformách a pohybuje se kdesi v šedé zóně mezi písmy a kódováním. Kapitoly šestá až osmá pokrývají každá podobná témata, jednou pro platformu Macintosh, poté pro Windows a konečně pro Unix/X11. Zatímco však popis platformy Macintosh je dosti podrobný v rozboru rozdílů v zacházení s písmy mezi MacOS 9 a Mac OSX, popis platformy Unix/X11 pokrývá pouze některé nejzákladnější a starobylé nástroje pro X11. Zde by bylo lze si přát poněkud rozsáhlejší pokrytí správy písem v moderních linuxových desktopových prostředích jako KDE či Gnome.

Následující dvě kapitoly diskutují na platformně nezávislé užití písem v systémech TEX/Omega a na webu. Kapitola devátá začíná přehledem vysokoúrovňového výběru písem v LATEXu/NFSS2, následovaným podrobným popisem nízkoúrovňové instalace písem pro *dvips*. Zbytek kapitoly pak diskutuje řady příkladů vytváření virtuálních písem s užitím nástroje *fontinst* k implementaci různých efektů potřebných v rozličných skriptech. Kapitola desátá uzavírá tuto část přehledem písem na webu při užití buď (X)HTML/CSS nebo alternativně SVG.

Závěrečná část knihy pokrývá témata vztažená k písmům. Kapitola jedenáctá se věnuje rozličným klasifikacím latinkových písem a zároveň poskytuje pěkně ilustrovaný přehled historie nejdůležitějších vzorů typů písem. Kapitoly dvanáctá a třináctá poté probírají tvorbu, úpravy a optimalizaci PostScriptových, TrueTypeových a OpenTypeových písem s užitím nástrojů jako jsou *FontLab* a *FontForge*. Konečně kapitola čtrnáctá uvádí koncept pokročilých typografických vlastností

poskytovaných v OpenType či AAT písmech a diskutuje způsoby obohacení písem využitím těchto možností.

Přídavek knihy převážně sestává z podrobných popisů písmových formátů. Začíná bitmapovými písmy a písmovými formáty vztaženými k TEXu, pokračuje k písmům PostScriptovým, TrueTypeovým, OpenTypeovým a písmům AAT, prakticky všechny významné formáty písem jsou do detailu probrány v dodatcích A až E. Konečně dodatek F probírá principy návrhu písem v METAFONTu a odvozených systémech, jakými jsou METAPOST, MetaFog a MetaType 1.

Komentář

Uvážíme-li rozsah knihy, je pochopitelné, že mezi zapsáním rukopisu a vydáním anglického překladu uplynulo několik let. Některé kapitoly jsou vzhledem k tomu bohužel v nebezpečí dosti rychlého zastarání. Pro většinu obsahu je třeba předpokládat, že anglické vydání z roku 2007 reprezentuje stav věcí z roku 2003.

Pro mnoho kapitol sloužících jako příručka takové zpoždění nepředstavuje problém, jelikož popisy kódování či formátů písmových souborů jsou nezměněny a stále platné. Na druhé straně je politováníhodné, že zejména kapitoly o TEXu zcela pominuly nebo přehlédly některé velmi důležité změny dosažené v posledních několika letech.

Jedním příkladem je, že při popisu podrobností instalace písem autor pokrývá pouze TEX/Omega s *dvips*, zatímco pdfTEX není v této souvislosti vůbec zmíněn, dokonce i přesto, že mnohé z popisu bylo by aplikovatelné na oba systémy v rámci systémů TEX Live. (Ve skutečnosti pdfTEX není zmíněn v knize vůbec nikde, možná proto, že v době jejího vzniku dosud nepodporoval Omegu.)

Dalším příkladem je hlavní autor pdfTEXu Hàn Thế Thành, jenž je zmíněn pouze jednou v souvislosti s vietnamskými písmy, zatímco jeho významné úspěchy týkající se implementace mikrotypografických vlastností pdfTEXu byly zcela pominuty. Je to ještě překvapivější s ohledem na to, že autor věnuje několik stran diskusi příkladu, kde efekt *okrajového kerningu*, jenž by byl k dispozici v pdfTEXu, je simulován dosti nemotorným způsobem s užitím virtuálních fontů vytvořených nástrojem *fontinst*.

Při probírání příkladů rozšíření písem Computer Modern autor doporučuje písma *CM-Super*, zatímco (nyní) mnohem populárnější písma Latin Modern jsou pouze mimochodem zmíněna jako příklad užití MetaType 1.

A konečně když přišla řeč na způsoby užití pokročilých typografických rysů písem OpenType nebo AAT v TEXu, autor nabízí pouze pár narážek na své vlastní výzkumy v rámci projektu Omega 2 (jenž se nedostal za prototypickou fázi), zatímco (nyní) k užití připravený nový projekt XƎTEX zmíněn není.

Abychom byli spravedliví, je třeba přiznat, že úspěch a důležitost těchto nejnovějších projektů ve světě TEXu nebylo lze předvídat v době psaní knihy

v roce 2003. I přesto však bylo možné přidat některé dodatky a/nebo opravy v době překladu do angličtiny v roce 2007.

Je dosti nepříjemné, že tato příležitost pro úpravy byla promeškána, neboť ty by knihu učinily mnohem užitečnější a cennější pro TEXové uživatele, již se zajímají o užití nejposlednějších úspěchů ve vývoji technologie písem.

I přes tyto nedostatky je kniha cenným zdrojem pro TEXové uživatele a softwarové vývojáře, zajímající se do hloubky o technologii písem a jejich kódování. Žádná jiná kniha nepokrývá ani přibližně tak široký rozsah témat v takové hloubce detailnosti v jediném svazku. Kdo by se k tomu chtěl byť i jen přiblížit, musel by sesbírat tucty materiálů z nejrůznějších zdrojů a stále by mu zůstalo mnoho prázdných oblastí k doplnění.

Podtrženo a sečteno, kniha je zcela jistě doporučeníhodná. Přece však pro budoucí vydání bylo by velmi žádoucí přidat některé dodatky nebo opravy.

Jako závěrečnou poznámku by autor recenze rád zmínil jistou drobnou kuriositu: jak je obvyklé v moderních učebnicích, i tato kniha obsahuje mimo obvyklého obecného rejstříku i rejstřík osob. Na rozdíl od jiných knih zde však autor byl dosti svobodomyslný, pokud šlo o to, které typy osob do rejstříku zařadit.

V důsledku toho autoři písmového software a nástrojů nejen že mohou sami sebe nalézti ve skvělé společnosti některých z nejslavnějších tvůrců písem historie, ale také v sousedství spíše pochybných politických postav (jako Lenin, Hitler, Mao) či fiktivních a literárních hrdinů (kupř. Sherlock Holmes, James Bond, James T. Kirk), v textu pouze mimochodem zmíněných v některých odlehčených komentářích.

Jestliže tedy autor recenze (jenž byl jistou dobu přispěvatelem do projektu *fontinst*) by rád vyjádřil své díky za příležitost být zahrnut do tak unikátního výběru osob, pak by zároveň rád vyjádřil i své vážné pochybnosti, zda je pro čtenáře vskutku pomocí, jsou-li fiktivní charaktery zařazeny do rejstříku osob stejným způsobem jako lidé „techničtí“.

Recenzent: Ulrik Vieth, `ulrik.vieth@arcor.de`
Vaihinger Straße 69, DE-70567 Stuttgart, Germany

Překlad: Marcel Svitalský
`marcel.svitalsky@centrum.cz`

Typografové a programátoři – vzájemné inspirace: 18. BachoTEX (30. 4. – 4. 5. 2010)

Karel Horák

Na letošní konferenci polského GUST jsme se sešli z Čech a Moravy hned čtyři: kromě autora této zprávičky ještě jeho spolujezdec Jano Kula, z Ostravy Jan Šustek a tradiční konferenční turista Karel Píška.

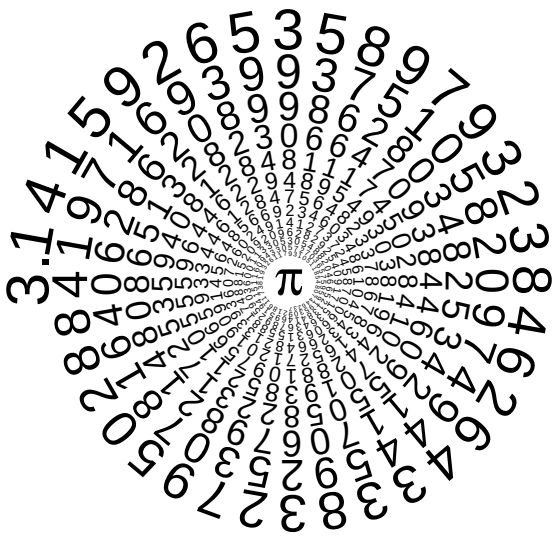
Do programu z nás lenochů nikdo nepřispěl, jen Honza Šustek na závěr neplánovaně vystoupil s vizualizací řádkového zlomu a blýskl se jednou TEXovou perlou. TEXové perly si mohou zájemci najít na stránkách GUST, `http://www.gust.org.pl/`. Najdou se tu nejen různé zajímavé a poučné triky, ale i některé málo zdokumentované vlastnosti TEXu.

Foto: Karel Horák (levé) a Jan Šustek (pravé)

Hned první páteční přednáška Adama Kolaného mě zaujala svým tématem: kombinací TEXového výstupu a možností programu Inkscape, volně šířeného programu pro vektorovou grafiku, s nímž mám sám nejednu velmi příjemnou zkušenost. Vlastně jsem do Bachotku odjížděl s nadějí, že mi tu někdo pomůže vyrobit wokenní binárku prográmku pdf2svg, který na rozdíl od filtru samotného Inkscape při převodu PDF vkládá do dokumentu i nesystémová písma. Naštěstí jsem předtím ještě znovu zagoogloval, a ejhle, fungující binárka už byla na světě, tedy na webu!

Po mnoha letech jsem tu zas potkal Keese van der Laana, který se po delší odmlce vrátil do světa TEXových setkání (jistě si naň vzpomenou účastníci historického EuroTEXu v Praze roku 1992). Předváděl zejména ukázky ze své knihovny procedurek napsaných v PostScriptu. Jednu z jeho vtipných ukázek

jsem pro vás zrekonstruoval (přitom jsem využil prvních šest řádků z výstupu prográmku `pi.tex` napsaného Denisem Roegelem, který jsem před časem někde objevil a s jehož pomocí jsem si jako kuriozitu a na památku nebohého pana Ludolfa nechal TEXem spočítat 5 000 platných číslic čísla π):



Nutno ovšem poznamenat, že něco podobného uměl už před mnoha lety Bogusław Jackowski v METAFONTu, a na konferenci to pomocí LuaTEXu předvedl jeho syn Paweł. Kromě toho jsem nemohl nenamítnout, že není nutné zvládat takové konstrukce v PostScriptu (konec konců dá se v něm i sázet), když tu máme METAPOST.

Totéž se dá říci i o proceduře na řešení Apolloniovy úlohy. Je to sice obdivuhodné využití programovacích možností PostScriptu, ale osobně dávám i nadále přednost mnohem čitelnějšímu METAPOSTu. Keesovy argumenty o přednostech PostScriptu na kreslení obloučků bych asi snadno vyvrátil (má sice pravdu, že METAPOST nemá `arc`, ale ve výsledku je makro na obloučky úhlů přehlednější). Nicméně na některé náročnější kousky jako třeba fraktály zatím METAPOST vzhledem ke známým omezením nestačí. Uvidíme však, co další úsilí Taca Hoekwatera ve vývoji METAPOSTu přinese.

Úhelnou přednáškou konference byl rozsáhlý příspěvek Bogusława Jackowského o interpolaci křivek METAFONTem, resp. METAPOSTem.

Ještě několikrát přišla řeč na matematiku, přičemž mě nejvíce překvapilo, že v Polsku se sice při zlomu na konci řádku stejně jako u nás opakují relační znaménka i na začátku nového řádku, ale neopakuje se takto minus! Údajně proto, že jak známo minus minus dává plus. Když uvážím, s čím zápasí dnešní

žactvo, tak nevím, jestli ve školních učebnicích lámání matematických vztahů vůbec nezakázat!

Ze zajímavých přednášek bych ještě vzpomněl „workshop“ Adama Twardocha týkající se tvorby OpenType písem. Téma rozhodně lákavé, oproti jiným Adamovým vystoupením, jež byla vždy excelentní, toto působilo poněkud nepřipraveně.

Impozantně naopak zapůsobila informace o systému utilit a maker na sazbu rozsáhlých katalogů zboží nazvaném autory PARCAT.

Na polských variantách etiket pomerančových džemů různých renomovaných výrobců předvedly dvě lepé TEXistky lesk a bídu typografie podobných produktů. Vybaveny skutečnými produkty a komentáři významného polského typografa Andrzeje Tomaszewského, tradičního to účastníka snad každého BachoTEXu.

Na klíčovou přednášku Andrzeje Tomaszewského o historii vzniku a vývoji symbolu letošní konference (¶) jsme si museli počkat až do samého závěru konference. Ještě předtím jsme však měli možnost zhlédnout 35minutový film *Falkbeerův protigambit* věnovaný významnému polskému typografu Stefanu Szczypkovi doplněný o krátkou animovanou pohádku *Jak vzdělání přišlo z lesa*. Kdo nezaváhal, mohl si pak oba filmy odnést na DVD.

Závěrem lze jen konstatovat, že téma konference bylo (alespoň pro mne) několikanásobně naplněno!

Karel Horák, `horakk@math.cas.cz`

Foto: Jan Šustek

TypeTalks 2010

Pavel Stříž

Tato krátká zpráva informuje o návštěvě denního pásma seminářů TypeTalks 2010 o písmech a jejich tvorbě, které se uskutečnilo v pondělí 21. června 2010 v Brně v Domě pánů z Kunštátu. Symposium TypeTalks (`http://typetalks.com/Symposium2010/`) bylo nekomerčním a nezávislým projektem a neoficiálním předvojem slavnostního otevření 24. mezinárodního bienále grafického designu Brno 2010 a velké akce Symposium bienále.

Kvalitních písem není nikdy dost

Akci nejlépe vystihují slova organizátorů: „I přes svou zásadní důležitost pro naši kulturu a rozvoj jsou písmo a textové informace obecně v grafickém designu a visuální komunikaci často zásadně podceňovány. Patrné je to především ve středoevropském postkomunistickém prostředí. TypeTalks je nezávislá aktivita, jež se snaží o zlepšení povědomí o moci písma, písmařství a typografie mezi studujícími i praktikujícími grafickými designery.“

Otcem i matkou akce byl David Březina (na fotce; `http://davi.cz/`) za spolupráce jeho studentů, kolegů a spolupracovníků. Při příjezdu jsme obdrželi řadu dárečků, jako kdybychom přijeli na velkou mezinárodní konferenci.

Fotky na této straně: Rob Keller a Florian Hardwig.

Přednášky a přednášející

V ochozech byla slyšet čeština, slovenština, angličtina, němčina i polština. Vystoupilo šest plus jeden přednášející. Přednášky a přednášející si představíme.

Florian Hardwig (D):
Lokalizujte! Rukopisná nářečí v typografickém písmu
Co dělá písmeno italským nebo německým? Florian Hardwig zkoumá regionální tvarosloví (latinkových) písmen a to, jak odpovídají příslušným rukopisným a písmomalířským tradicím. Jak současní tvůrci písem a grafičtí designeři využívají výrazně lokálních tvarů pro svoje písmové skripty a logotypy?

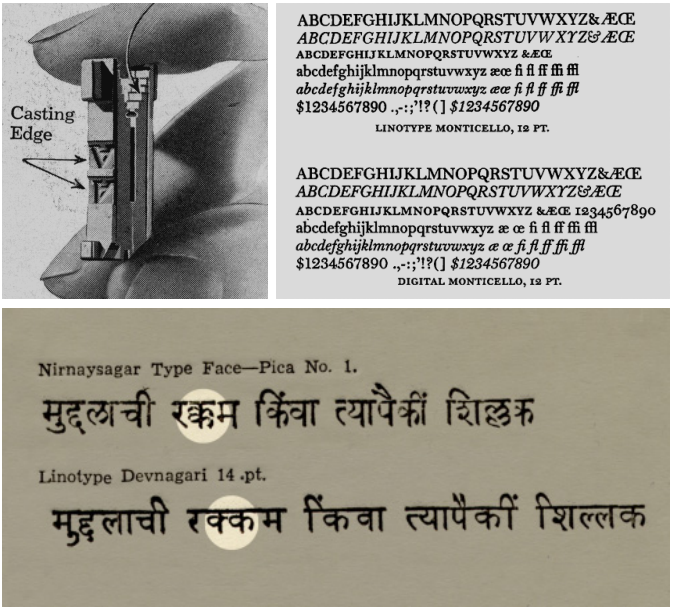
> Florian Hardwig je grafický designer sídlící v Berlíně, kde spolu s Maltem Kaunem provozuje studio. Od roku 2007 vyučuje typografii na Hochschule für Bildende Künste Braunschweig. Floriana můžete často najít na stránkách Typophile, kde je jedním z moderátorů Type ID Board (identifikace písem). Jeho specializací jsou psací vzory. Ve svém projektu Manuscribe zkoumá rozličné rukopisné modely, jež se používají k výuce psaní na základních školách. `http://florian.hardwig.com/`

Snímky z přednášky Floriana Hardwiga.

Tato přednáška ve mně okamžitě evokovala nedávné zkoumání formátu OTF a jeho vlastností. Není těžké si představit, že si připravíme alternativy jednotlivých glyfů a pak si je na úrovni editoru přepínáme. V případě TEXu by to mohlo být na úrovni příkazu `\selectlanguage`, používáme-li `babel` či vlastní přepínač.

Rob Keller (US/D): Písmové technologie blázní!
Písmena a typografická sazba prošly během posledních pár tisíc let několika zásadními posuny. Tyto rozličné změny v mnoha ohledech výrazně ovlivnily vzhled písmen, navrhování tiskového písma i typografie. V nejlepších případech tyto kroky napravily nedostatky předchozích technologií, rozšiřujíce typografické a lingvistické možnosti. Jindy zadupaly do země tradice, poškodily celé písmové systémy a snížily typografické standardy. Tak či onak, písmové technologie se stále vyvíjejí…

Rob Keller (nar. 1981) je písmař pocházející z kukuřičných a sojových polí státu Illinois (US). O typografii se začal vážně zajímat během studií grafického designu a sochařství na Univerzitě v Illinois. Tato vášeň ho zavedla na Univerzitu v Readingu, kde vystudoval tvorbu písma. Po tomto intenzivním roce se přesunul do Německa, aby pracoval pro společnost Linotype, kde také potkal svoji budoucí ženu, Sonju. V roce 2009 společně založili v Berlíně písmolijnu Mota Italic. Zde se v současnosti zabývají vývojem rozsáhlých písmových rodin se zaměřením na širokou jazykovou podporu a rozličné písmové systémy. `http://motaitalic.com/`



Snímky z přednášky Roba Kellera.

Michael Hochleitner (AT):
Současný pohled na vztah písmomalířství a typografického písma
Jaký je rozdíl mezi písmomalířstvím, logotypy a (typografickým) písmem? Jaké jsou jejich výhody a jaký je jejich vzájemný vztah, co se dnešního použití týče?

Michael Hochleitner se narodil a žije ve Vídni. Ve Vídni také pět let studoval grafický a komunikační design. Studia zakončil v roce 2003 diplomem. Po roce civilní služby pro svou zemi studoval mediální studia ve Vídni a zároveň řídil taxi, aby si vydělal peníze. Mezi lety 2004 a 2007 pracoval jako nezávislý grafický designer. V roce 2007 dokončil s vyznamenáním studia tvorby písma (MA Typeface Design) v britském Readingu, kde také začal navrhovat

svoje písmo Ingeborg. V roce 2008 založil spolu s Annou Fahrmaierovou a Thomasem Gabrielem písmolijnu Typejockeys. `www.typejockeys.com`

Tomáš Brousil (CZ): Tabac

Specifické požadavky novinové a časopisecké typografie daly vzniknout novému extrémně rozsáhlému písmovému systému. Tomáš Brousil představil svoji doposud nevydanou písmovou rodinu Tabac, která ve své serifové verzi obsahuje 96 řezů a neobvyklý koncept písmových odstínů.

Tomáš Brousil (nar. 1975, Nitra) se vyučil leteckým mechanikem. Od roku 2002 navštěvoval Vysokou školu uměleckoprůmyslovou v Praze, v ateliéru Tvorba písma a typografie nyní působí jako asistent. V roce 2003 založil písmolijnu Suitcase Type Foundry, která dnes nabízí přes 200 řezů původních písem. Je držitelem řady ocenění: TDC Certificate of Excelence in Type Design (2008), hlavní cena v kategorii Tvorba písma, Brno (2008), nominace na Design Award of the Federal Republic of Germany 2009, German Design Council, Frankfurt (2008), Hlavní cena v kategorii písmo a cena poroty ED-Awards (2009). `http://www.suitcasetype.com/`

Foto: Rob Keller a Florian Hardwig.
Snímky z přednášky Tomáše Brousila.

Dle slov přednášejícího trávil nad touto písmovou rodinou dva roky. Mě samozřejmě oslovily šachové figurky, především proto, že jsem se nedávno zabýval nastavením ochranné zóny glyfů, poněvadž u šachů a černého/šedého/vyšrafovaného pozadí to byla pro oči příjemná změna. O těchto experimentech přes balíček chessboard a nástroje FontForge a Inkscape snad jindy.

Dan Reynolds (US/D): Vášeň mladého multilingvního designera
Mnoho mladých písmařů se věnuje designu pro různé písmové systémy. Volí si tento druh práce především pro výzvy, jež skýtá; kreslí písma, protože musí. Po důkladném zkoumání se zdá, že prvky jiných písmových systémů nejsou až tak odlišné od našich. Učení se jiným sadám pravidel, zvyklostem a jiné historii však otvírá nové cesty k lepšímu pochopení podstaty písma. Nezáleží na tom, jestli se výsledků bojíme, pochybujeme o nich nebo je respektujeme, spousta nelatinkových písmových systémů prostě funguje jako zrcadlo designerovy vášně pro písmo. Je nepodstatné, zda se kreslí znaky z latinky, cyrilice nebo dévanágarí. Podstatné je, aby písmo fungovalo.

> Dan Reynolds sídlí v Berlíně, je typografickým inženýrem a specialistou v Linotype GmbH a vyučujícím na Hochschule Darmstadt. Během své první návštěvy v německé Mohuči se Dan rozhodl, že se stane písmařem. Poté, co dokončil studia grafického designu na RISD (Rhode Island School of Design) s titulem BFA, se nadobro přesunul do Evropy. Se čtyřmi dalšími studenty z HfG Offenbach založil v roce 2004 Offenbach Typostammtisch (pravidelné písmařské srazy). V roce 2008, rovněž dokončil s vyznamenáním studia tvorby písma na Univerzitě v Readingu. Jeho poslední písmo Malabar získalo ocenění TDC Certificate of Excellence in Type Design, stříbrnou medaili na ED-Awards 2009 a v roce 2010 jednu z pěti zlatých medailí Německa v celonárodní designové soutěži. `http://www.linotype.com/`

Tato přednáška mě zaujala, poněvadž písma, která jsou stylisticky obdobně pojednaná, kde nerozeznáte na první pohled různé abecedy a slabičná písma, obecně písmové systémy – latinka, cyrilice či dévanágarí – mezi sebou, jsou dosti výjimečná a stojí za naši pozornost. Během přednášky byla zmíněna řada reálných ukázek. Toto téma bádání se opakovalo i u jiných přednášejících.

Dan Rhatigan (US/UK):
How I learned to stop worrying and love bad type
It's a look at the need for objectivity in type design and typography, emphasizing the importance of looking not just at reliable sources but also at "bad" or amateur examples. These can have many useful lessons of their own that can inform the designer's work.

> Dan Rhatigan is a type designer/graphic designer living in London, England. He currently works on a joint project between the University of Reading and Monotype Imaging, researching and designing non-Latin typefaces, primarily those used in India. `http://www.ultrasparky.org/`

Snímek z přednášky Dana Rhatigana.

Veronika Burian (CZ/D): Typografické párování

Když designeři začínají nový projekt, je volba písma jedním z mnoha úkolů, které je čekají. Zásadní je rozpoznání efektu kombinování písem. Některé z důležitých otázek v tomto kontextu jsou tyto: jaký styl typografie přenese sdělení nejlépe, jaká písma spolu pro daný účel dobře fungují, jaká znaková sada a kolik tučností je potřeba a nakonec jaké funkce očekáváme, že písma budou plnit. Tato prezentace se zabývá právě takovými základními problémy a nabízí návod, jak přistupovat k citlivým vztahům lásky a nenávisti, které mezi písmy mnohdy existují. Pojednání je ilustrováno ukázkami „typografických párů", těch dobrých i těch špatných.

Foto: Rob Keller a Florian Hardwig.
Snímky jsou z přednášky Veroniky Burian.

Veronika Burian se narodila v Praze, svůj první titul z průmyslového designu získala v Mnichově. Objevivši skutečnou vášeň pro písmo, vystudovala

v roce 2003 s vyznamenáním tvorbu písma v britském Readingu (MA Typeface Design) a začala pracovat na plný úvazek jako písmařka pro jiholondýnský DaltonMaag, kde zůstala až do roku 2007. Nyní žije a pracuje v Praze a naplno se věnuje TypeTogether, písmolijně, kterou založila s Josém Scaglionem. Její písmo Maiola získalo TDC Certificate of Excellence in Type Design 2004. Karmina a Bree, dva společné projekty z TypeTogether, rovněž získala ocenění v soutěži ED-Awards v letech 2007 a 2009. `http://type-together.com/`

Viva TypeTalks 2011 nebo snad 2012?

Přednášky byly poutavé, diskuze zajímavé, ale pracovní povinnosti volaly. Proto jsem se nepřipojil na pozdně noční večeři nezmíněnou v programu sympózia a neformální diskuze. Snad zas někdy příště!

David Březina přemýšlí, zdali zorganizovat TypeTalks 2011 či až TypeTalks 2012. Tak či onak, máme se na co těšit.

Summary: TypeTalks 2010

This is a short report of a participation at the TypeTalks 2010 symposium in Brno, the Czech Republic. The theme of this conference was type. This is a broad area embracing the history of type, the design of type, type education, the use of type (typography) and much more. The key criteria for the acceptance of a talk is that it have educational value. It is not enough to simply show work, knowledge must be shared as well.

There were seven invited speakers: Florian Hardwig (D): Localize! The dialects of handwriting in type design; Rob Keller (US/D): Font technology is crazy!; Michael Hochleitner (AT) A contemporary view on the relationship of lettering and type; Tomáš Brousil (CZ): Tabac; Dan Reynolds (US/D) The passion of the young, multi-script type designer; Dan Rhatigan (US/UK): How I learned to stop worrying and love bad type; and Veronika Burian (CZ/D): Typographic matchmaking.

Snímek: Michael Hochleitner.

Pavel Stříž
striz@fame.utb.cz

Svět knihy Praha 2010

Miloš Brejcha

Na následujících dvou stranách přinášíme krátkou informaci o proběhlém knižním veletrhu Svět knihy 2010.

Šestnáctý mezinárodní knižní veletrh a literární festival Svět knihy Praha 2010 proběhl na výstavišti Praha-Holešovice ve dnech 13. – 16. 5. 2010, pod záštitou ministra kultury ČR, primátora hlavního města Prahy a starosty Městské části Praha 7. Čestným hostem letoška bylo Polsko, které veřejnosti nabídlo mnohá setkání se spisovateli, výstavy, pořady i dílny pro děti.

Mezi vystavovateli zde byly nakladatelství z Belgie, Číny, Finska, Francie, Itálie, Izraele, Japonska, Jižní Koreje, Maďarska, Německa, Polska, Portugalska, Rakouska, Rumunska, Ruska, Saúdské Arábie, Slovenska, Španělska, Švédska, Švýcarska, Tchajwanu, USA či Velké Británie. Veletrhu ze zúčastnilo 414 vystavovatelů ve 194 stáncích na ploše 2 930 m^2. Veletrh se dočkal rekordní návštěvnické účasti, shlédlo jej čtyřicet tisíc návštěvníků, což je o pět tisíc více než v roce 2009. Návštěvnickému rekordu přispělo i chladnější počasí.

Velkým tématem odborného dne (čtvrtek) byly elektronické knihy a jejich masivní rozšíření. V několika realistických i nerealistických vizích byl nastíněn jejich další vývoj. Asi nejzajímavější vystoupení k tomuto tématu měli zástupci Googlu, kteří nabídli vydavatelům zajímavé možnosti jak prezentace, tak i distribuce elektronických knih. Méně povedená byla pak představa jednoho z „propagátorů“ elektronických knih, jehož představa byla, že všechny knihy budou zdarma (podle něj odpadnou všechny práce spojené s vydáním knihy, bude třeba ji jen napsat a autor si ještě sežene sponzory). Naštěstí většina přítomných jeho názor nesdílela a tak i nadále bude potřeba lidí, kteří dokáží udělat dobrou sazbu a tedy i nás TEXistů. :-)

Další dny už byly věnovány běžným návštěvníkům a tedy se plně rozběhla autorská čtení, křesty knih a autogramiády.

Oproti minulým ročníkům bylo velkou vadou na kráse, že se pořád nepodařilo, i přes prohlášení pražských zastupitelů, dostavět shořelé levé křídlo Průmyslového paláce, snad se dočkáme v blízké budoucnosti.

Návštěva Světa knihy 2010 určitě byla zajímavá jak pro běžného čtenáře, tak i pro ty co se podílí na vydávání knih. Doufám, že i v příštím roce se výstava podaří a i nadále bude největší knižní akcí u nás.

Miloš Brejcha, brejcha@vydavatelskyservis.cz
Foto: František Minář


ZEMĚ SVĚTA
KRÁSY SLOVENSKA


OL – CZECH UNIVERSITY PUBLISHING

Zápis z valné hromady CSTUG Brno 21. 11. 2009

Program

Valná hromada začala ve 14:25 hod a řídil ji předseda sdružení J. Kuben.

1. Zahájení.
2. Zpráva o činnosti sdružení za rok 2009 — přednesl předseda.
3. Zpráva o hospodaření sdružení za rok 2009 — přednesl předseda.
4. Zprávy revizorů — přednesli oba revizoři T. Hála a P. Sekanina.
5. Informace o projektech podporovaných finančně CSTUGem — předseda informoval o skutečnosti, že CSTUG se zavázal v roce 2009 podpořit projekt TEX Gyre částkou 1 000 eur (kterou v únoru řešitelům poukázal) a poskytnout částku 1 000 eur na společnou podporu projektů MPlib a LuaTEX (kterou v červnu řešitelům poukázal).
   Na příští rok výbor počítá se dvěma částkami na podporu stejných projektů, vždy v rozmezí 500 až 1 000 eur s tím, že valná hromada pověří výbor, aby konkrétní výše stanovil dle okolností (požadavky řešitelů, výše příspěvku dalších organizací, ...).
6. Plán činnosti na rok 2010 — předseda seznámil přítomné s rámcovým návrhem činnosti:
   - Vydávání Zpravodajů (číslo 4/2009 půjde do tisku ještě v listopadu).
   - Zorganizování 3. ročníku TEXperience 2010 (akci případně spojit s mezinárodní konferencí uživatelů systému ConTEXt, která má proběhnout v okolí Prahy pravděpodobně v září).
   - Testování TEX Live a pořízení DVD.
   - Finanční podpora a případná pomoc s testováním projektů TEX Gyre a MPlib & LuaTEX.
   - Vyhlášení grantu pro podporu češtiny a slovenštiny ve fontech TEX Gyre (do 10 tisíc Kč).
   - Zprovoznění nového webové rozhraní CSTUGu.
   - Aktualizace členské databáze.

- Docílit stavu, aby Zpravodaje byly vždy nejpozději po roce vystaveny v elektronické podobě na webu sdružení.
- Zviditelňování CSTUGu.

7. Příští rok končí funkční období výboru. Je třeba promyslet a připravit systém voleb (a to včetně vhodných kandidátů) a následně volby zrealizovat.

8. Závěr.
   (a) Valná hromada jednohlasně (všichni přítomní v počtu 23 byli pro) schválila:
      - Zprávu o činnosti sdružení za rok 2009.
      - Zprávu o hospodaření sdružení za rok 2009.
      - Finanční podporu projektům TEX Gyre a MPlib & LuaTEX.
      - Vyhlášení grantu pro podporu češtiny a slovenštiny ve fontech TEX Gyre.
      - Neměnnou výši členských příspěvků.

   (b) Valná hromada pověřila výbor sdružení, aby konkrétní výši finanční podpory projektu TEX Gyre stanovil v rozmezí 500 až 1 000 eur a stejně tak konkrétní výši společné finanční podpory projektům MPlib & LuaTEX v rozmezí 500 až 1 000 eur.

   (c) Valná hromada vzala na vědomí zprávy revizorů.

   (d) Valná hromada doporučila, aby CSTUG podpořil mezinárodní konferenci uživatelů systému ConTEXt.

Zapsal: Rudolf Schwarz
Ověřili: Petr Sojka, Libor Škarvada

Permanent Scientific Board / Trvalá vědecká rada a recenzenti

Jaromír Antoch, Charles University, Prague; Petr Aubrecht, Czech Technical University; Radek Benda, Tomas Bata University in Zlín; Ivan Bíbr, Divido systems, s. r. o.; Michaela Beranová, Akademie Sting, o. p. s., Brno; Miloš Brejcha, Vydavatelský servis, občanské sdružení; Karel Brychta, Brno University of Technology; Miroslav Červenka; Miroslava Dolejšová, Tomas Bata University in Zlín; Jiří Dvorský, Technical University of Ostrava; Karel Horák, The Academy of Sciences of the Czech Republic; Janka Chlebíková; Petr Klímek, Tomas Bata University in Zlín; Miroslava Komínková, Tomas Bata University in Zlín; Jiří Kosek, University of Economics, Prague; Martin Kovářík, Tomas Bata University in Zlín; Michal Mádr, Czech Republic; Robert Mařík, Mendel University, Czech Republic; Michal Mráz; Petr Nevřiva; David Ondřich, Charles University; Vlastimil Ott, Editor-in-Chief, `www.e-ott.info`; Karel Píška, The Academy of Sciences of the Czech Republic; Michal Polášek; Michal Růžička, Masaryk University Brno; Jozef Říha, T-Systems Slovakia, s. r. o., Slovakia; Miroslav Saferna, consultant IDS-Scheer ČR, s. r. o.; Martin Stříž; Marcel Svitalský; Petr Šafařík, `Mandrivalinux.cz`; Libor Škarvada, Masaryk University Brno; Jan Šustek, University of Ostrava; Vojtěch Trefný, Ubuntu ČR; Radek Vicherek, Gedip, s. r. o., Zlín; Petr Vokáč, Nuclear Research Institute Řež plc; Vít Zýka, TYPOkvítek Praha, Czech Republic.

Supporters / Sympatizanti

TEX Users Group; Česká statistická společnost; Fakulta managementu a ekonomiky UTB ve Zlíně; Zdeněk Wagner, Ice Bear Soft; Liberix, o. p. s.; Stickfish, s. r. o.; Občanské sdružení Ubuntu pro ČR; Vydavatelský servis; CEED ing. Jena Švarcová, Ph.D.; DIVIDO systems, s. r. o.; KONVOJ, s. r. o.; nakladatelství Martin Stříž; Ing. Jiří Kosek.

Companies, Faculties and Colleges / Kolektivní členi

Fakulta prírodných vied Žilinskej univerzity; Technická univerzita v Košiciach; Fakulta dopravní ČVUT v Praze; Fakulta jaderná a fyzikálně inženýrská ČVUT v Praze; Univerzita Karlova v Praze, Ústav výpočetní techniky; Matematicko-fyzikální fakulta Univerzity Karlovy v Praze; Vysoká škola ekonomická, Fakulta informatiky a statistiky; Ústav makromolekulární chemie AV ČR; Centrum počítačových služeb Fakulty strojní ČVUT v Praze; Fakulta elektrotechnická ČVUTv Praze; Ústav informatiky a výpočetní techniky AV ČR; Ústav teorie informace a automatizace AV ČR; Ústav fyziky plazmatu AV ČR; Fyzikální ústav AV ČR; Astronomický ústav AV ČR; Západočeská univerzita v Plzni; SPŠ a VOŠ Písek; Fakulta stavební ÚMDG ČVUT v Praze; Fakulta informatiky Masarykovy Univerzity v Brně; Aura, s. r. o.; Provozně ekonomická fakulta MZLU v Brně; Fakulta strojního inženýrství VUT v Brně; Univerzita obrany v Brně; Konzervatoř, Brno; VŠB-TU Ostrava; Ústav geoniky AV ČR; VŠB-TU Ostrava, Fakulta elektrotechniky a informatiky; Slezská univerzita v Opavě; Fakulta aplikované informatiky UTB ve Zlíně; Fakulta elektrotechniky a informatiky STU; Ústav svetovej literatúry SAV; Matematický ústav SAV; Fakulta prírodných vied Univerzity Mateja Bela; Matematický ústav AV ČR.

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

Vydalo:	Československé sdružení uživatelů TEXu vlastním nákladem jako interní publikaci
Obálka:	Antonín Strejc
Ilustrace na obálce:	Ken Lunde a jeho spolupracovníci
Počet výtisků:	480
Uzávěrka:	1. 5. 2010
Odpovědný redaktor:	Zdeněk Wagner
Redakční rada:	Ján Buša, Jiří Demel, Tomáš Hála, Jaromír Kuben, Jiří Rybička, Rudolf Schwarz, Petr Sojka, Pavel Stříž, Marcel Takáč, Jiří Veselý
Výroba a distribuce:	KONVOJ, spol. s r. o., Berkova 22 Brno 612 00, tel. +420 541 245 548
Adresa:	CSTUG, c/o FEL ČVUT Technická 2, 166 27 Praha 6
Tel:	+420 224 353 611
Fax:	+420 233 332 938
E-mail:	cstug@cstug.cz

Zřízené poštovní aliasy sdružení CSTUG:

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

`board@cstug.cz`
korespondence členům výboru

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

`gacstug@cstug.cz`
grantová agentura CSTUGu

`secretary@cstug.cz`, `orders@cstug.cz`
korespondence administrativní síle sdružení, objednávky CD a DVD

`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`

CONTENTS