TEXnica Ars

Rivista italiana di TEX, LATEX e tipografia digitale

33 • 2022

TEXnica ArS

GUIT – Gruppo Utilizzatori Italiani di TEX

ArsTEXnica è la pubblicazione ufficiale del GUIT

ArsTEXnica è la prima rivista italiana dedicata a TEX, a LATEX e alla tipografia digitale. Lo scopo che la rivista si prefigge è quello di diventare uno dei principali canali italiani di diffusione di informazioni e conoscenze sul programma ideato da Donald Knuth.

Le uscite hanno cadenza semestrale e sono pubblicate nei mesi di Aprile e Ottobre. In particolare, la seconda uscita dell'anno contiene gli Atti del Convegno Annuale (GUIT*meeting*).

Chiunque volesse collaborare con la rivista a qualsiasi titolo (recensore, revisore di bozze, grafico, etc.) può contattare la redazione all'indirizzo:

arstexnica@guitex.org.

Pubblicare su ArsTEXnica

La rivista è aperta al contributo di tutti coloro che vogliano partecipare con un proprio articolo. Questo dovrà essere inviato alla redazione di ArsTEXnica, per essere sottoposto alla valutazione di recensori. È necessario che gli autori utilizzino la classe di documento ufficiale della rivista; l'autore troverà raccomandazioni e istruzioni più dettagliate all'interno del file di esempio (`.tex`). Tutto il materiale è reperibile all'indirizzo web della rivista.

Gli articoli potranno trattare di qualsiasi argomento inerente al mondo di TEX e LATEX e non dovranno necessariamente essere indirizzati ad un pubblico esperto. In particolare tutorials, rassegne e analisi comparate di pacchetti di uso comune, studi di applicazioni reali, saranno bene accetti, così come articoli riguardanti l'interazione con altre tecnologie correlate.

Di volta in volta verrà fissato, e reso pubblico sulla pagina web, un termine di scadenza per la presentazione degli articoli da pubblicare nel numero in preparazione della rivista. Tuttavia gli articoli potranno essere inviati in qualsiasi momento e troveranno collocazione, eventualmente, nei numeri seguenti.

Nota sul Copyright

Associarsi a GUIT

Fornire il tuo contributo a quest'iniziativa come membro, e non solo come semplice utente, è un presupposto fondamentale per aiutare la diffusione di TEX e LATEX anche nel nostro paese. L'adesione al Gruppo prevede una quota di iscrizione annuale diversificata: 30,00 € soci ordinari, 20,00 (12,00) € studenti (junior), 75,00 € Enti e Istituzioni.

Indirizzi

Gruppo Utilizzatori Italiani di TEX
c/o Università degli Studi di Napoli Federico II
Dipartimento di Ingegneria Industriale
Via Claudio 21
80125 Napoli – Italia
`http://www.guitex.org`
guit@guitex.org

Redazione ArsTEXnica:
`http://www.guitex.org/arstexnica/`
arstexnica@guitex.org

ISSN 1828-2350 (Stampa)
ISSN 1828-2369 (Online)

15 Aprile 2022

ArsTEXnica

Rivista italiana di TEX e LATEX

Numero 33, Aprile 2022

Gruppo Utilizzatori Italiani di TEX

Editoriale

Francesco Biccari

Cari lettori, prima di presentarvi i sei articoli che troverete in questo numero di ArsTEXnica, vorrei spendere qualche parola sul futuro del GUIT. La situazione organizzativa non è delle migliori. Il GUIT si regge sul tempo libero che una manciata di persone dedicano a questa associazione. E questo tempo è in lenta ma costante diminuzione. È un processo fisiologico, ma diventa inesorabile quando si inizia a prendere coscienza del fatto che l'arrivo di nuove leve è una flebile speranza. I segnali sono abbastanza palesi, come l'organizzazione del prossimo GUIT meeting. Nonostante la fine dell'emergenza COVID, verrà tenuto molto probabilmente online e trasmesso in streaming, come nelle due precedenti edizioni. L'appuntamento è fissato per l'ultimo o penultimo sabato di ottobre 2022, ma non abbiamo ancora una data precisa. Per presentare un contributo al meeting dovete inviare il relativo articolo all'indirizzo `arstexnica@guit.it` entro il 31 agosto 2022. Invito chiunque legga questo editoriale e abbia a cuore il GUIT a farsi avanti per aiutare nell'organizzazione della nostra amata associazione. Il consiglio direttivo saprà come sfruttare al meglio qualsiasi competenza verrà messa a disposizione.

Veniamo ora alla GUIT *challenge* che era stata pubblicata nel numero 31: i partecipanti avrebbero dovuto cimentarsi nel preparare un pacchetto per elaborare e mettere in grafico dei dati sperimentali forniti in un file esterno. Per ora però non è arrivata alcuna soluzione. La scadenza per la partecipazione è posticipata quindi al 31 agosto 2022, così il vincitore potrà presentare la sua soluzione al prossimo GUIT meeting. Al vincitore andrà il volume *TEX by topic* avvolto nella famosa carta da regalo del GUIT.

Ma vediamo cosa troverete in questo numero di ArsTEXnica. Nel primo articolo Gianluca Pignalberi, in risposta ai recenti interventi comparsi anche su ArsTEXnica, intervista Enrico Gregorio e Frank Mittelbach sul futuro di LATEX e sul ruolo che avrà il progetto LATEX3.

Nel secondo articolo Gianluca Pignalberi lascia il ruolo dell'intervistatore e veste i panni dell'*influencer* appassionato di TEX. Ci mostra infatti degli esempi di codice LATEX e dei suggerimenti su come realizzare delle immagini per essere usate come storie o post su Instagram.

Nel terzo articolo Claudio Beccari ci presenta la sua nuova fatica: il pacchetto wrapfig2. Probabilmente conoscete il pacchetto wrapfig di Donald Arsenau che permette di inserire immagini o tabelle avvolte dal testo principale. Questo è molto utile quando queste sono particolarmente strette. Purtroppo però è anche noto che wrapfig mostra spesso dei problemi, principalmente dovuti al calcolo delle righe di testo da far rientrare per far posto all'immagine/tabella e alla sua didascalia. Per risolvere queste idiosincrasie Claudio Beccari ha sviluppato wrapfig2, inserendo inoltre ulteriori funzionalità rispetto al pacchetto originale.

Nel quarto articolo rimaniamo sempre sul tema immagini. Herbert Voß ci presenta il suo pacchetto hvfloat: un pacchetto molto potente che permette di gestire il posizionamento delle immagini e della loro caption in una moltitudine di varianti. Per esempio, permette di affiancare la didascalia all'immagine, anziché posizionarla sopra o sotto; oppure permette di inserire immagini che coprano entrambe le colonne di testo nel caso di testo a due colonne; oppure

ancora permette di inserire immagini a doppia pagina. Questo articolo è già stato pubblicato su TUGboat ... e viene qui riprodotto su permesso dell'autore e della rivista TUGboat, grazie a Claudio Beccari che si è occupato dell'adattamento per ArsTEXnica.

Gli ultimi due articoli sono di Frank Mittelbach e riguardano la resa dei caratteri non ASCII in TEX. Il primo ci parla infatti del *tofu*, che non è un alimento, ma il nome con cui vengono indicati i famosi rettangolini che compaiono al posto dei caratteri quando tali caratteri non sono presenti nel font che si sta usando. Frank Mittelbach ripercorre brevemente la storia delle codifiche di input e delle codifiche dei font e la situazione attuale di Unicode nei sistemi TEX. Nell'ultimo articolo, naturale continuazione del precedente, Frank Mittelbach ci presenta il suo unicodefonttable, un pacchetto per stampare tabelle di caratteri, specificando il range di caratteri Unicode o i blocchi Unicode da mostrare.

Il 25 febbraio 2022 si è spento all'età di 75 anni Robin Fairbairns, un nome illustre del mondo TEX. Autore di pacchetti come endnotes, footmisc, setspace, presidente per anni del TUG del Regno Unito, e gestore del CTAN del Regno Unito.

Vi saluto e vi auguro una buona lettura.

Francesco Biccari
Dipartimento di Fisica e Astronomia
Università degli Studi di Firenze
Firenze, Italia
biccari@gmail.com

Enrico Gregorio parla di LaTeX3

Gianluca Pignalberi, Enrico Gregorio and Frank Mittelbach

Sommario Enrico Gregorio (con uno *spin-off* inatteso), intervistato in merito, ci parla del presente e del futuro di LaTeX3, analizzandone col codice alla mano vantaggi e potenzialità.

Abstract Enrico Gregorio (with the help of an unexpected spin-off), interviewed about it, tells us about the present and the future of LaTeX3, analyzing advantages and potential with meaningful code snippets.

Enrico Gregorio non ha bisogno di presentazioni. Potrebbe tranquillamente essere il condiscendente utente Unix ritratto da Scott Adams in Dilbert (figura 1) nell'aspetto e, soprattutto, nella conoscenza, se non fosse che non indossa *mai* bretelle. Al 31 gennaio 2022, il numero delle sue risposte sulla sola piattaforma StackExchange su TeX e derivati supera 22 000, con una reputazione di oltre 960 000. Il numero di pacchetti da lui scritti (in base a CTAN) è 17 (senza contare quelli in cui è intervenuto senza prendersene il merito come il "mio" ormai obsoleto combelow).

Al GUIT *meeting 2020* qualche cassandra ha fatto delle osservazioni, a mio parere ingenerose, sul futuro di LaTeX ed ho voluto capirne di più con l'aiuto di chi LaTeX lo vive e quotidianamente lo mantiene e migliora in tantissimi modi.

GIANLUCA PIGNALBERI: Enrico, potresti riassumere la storia di TeX in dieci date rappresentative?

ENRICO GREGORIO: Dieci date che giudico significative sono:

1978 prima versione di TeX

1982 seconda versione di TeX, con molti cambiamenti

1984 prima versione di LaTeX

1985 terza versione di TeX, con alcuni cambiamenti

1989 AMS-TeX

1991 NFSS

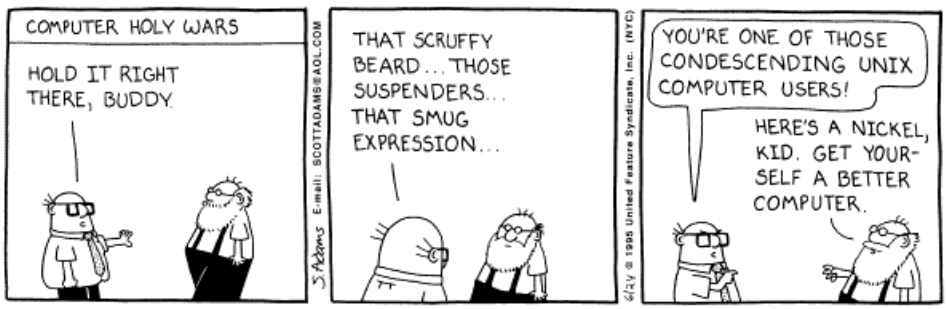


Figura 1. Famosissima striscia di Dilbert sull'archetipico utente Unix.

1992 $\mathcal{AMS}$-LaTeX

1993 teTeX

1994 LaTeX 2_ε

1996 TeX Live

G.P.: Da molti anni "un fantasma si aggira per il mondo". No, non è il topo di DeLillo che cita Marx e il comunismo in Europa: è LaTeX3. È significativo il fatto che tu non lo abbia incluso nel precedente elenco? Ce ne parli?

E.G.: Lo sviluppo di expl3 è cominciato subito dopo il rilascio di LaTeX 2_ε. Le prime idee risalgono al 1995, con la distinzione tra funzioni e variabili. Il problema, per parecchi anni, è stato la difficoltà di ottenere un kernel funzionante date le restrizioni di memoria. Perciò è rimasto silente fino a che Joseph Wright notò le potenzialità del linguaggio e lo impiegò per scrivere la seconda versione di siunitx nel 2010.
Il che causò il suo ingresso nel LaTeX team. Poco dopo venne cooptato anche Bruno Le Floch, e lo sviluppo fu piuttosto tumultuoso. Nel 2012 ci entrai anch'io dopo l'uscita di kantlipsum, un fondamentale pacchetto di cui tutti sentivano il bisogno. Scherzi a parte, cominciai ad adoperarlo per varie risposte su TeX.SE ma anche per xpatch, cominciando anche a sviluppare regexpatch: un buon numero di funzionalità di l3regex sono il risultato della corrispondenza per risolvere alcuni problemi.
Molte parti di expl3 sono state ammodernate, alcune abbandonate, altre modificate. Dalla versione 2020-10-01, expl3 è entrato nel kernel di LaTeX.
Fino a un paio d'anni fa, l'idea era di sviluppare LaTeX3, ma non accadrà mai: un nuovo kernel avrebbe mandato in soffitta migliaia di pacchetti e nessuno l'avrebbe usato. Quindi la strategia è di inserire a poco a poco codice basato su expl3, per esempio i nuovi "hooks" (si veda lthooks per maggiori informazioni).
Un esempio? Ecco: la definizione standard di `\par` è

```
\cs_new_protected:Npn \para_end: {
  \mode_if_horizontal:TF {
    \mode_if_inner:F {
         \tex_unskip:D
         \hook_use:n{para/end}
         \@kernel@after@para@end
         \mode_if_horizontal:TF {
           \if_int_compare:w 0 < \tex_lastnodetype:D
             \tex_kern:D \c_zero_dim
           \fi:
           \tex_par:D
           \hook_use:n{para/after}
           \@kernel@after@para@after
         }
         { \msg_error:nnnn { hooks }{ para-mode }{end}{horizontal} }
    }
  }
  \tex_par:D
}
```

che non intendo spiegare, ma che mostra l'uso di expl3 nel kernel. Ci sono altre varianti, ma il kernel dice a un certo punto

```
\cs_set_eq:NN \par \para_end:
```

Anche la parte fondamentale di xparse è stata inserita nel kernel, dando accesso senza bisogno di altro alle funzionalità più potenti per definire comandi a livello utente.

G.P.: L'esempio che ci hai mostrato evidenzia chiaramente quello che, a tutta prima, sembra un deterrente all'uso di expl3: la verbosità autoesplicativa dei comandi. Non dico di essere affezionato alla stringatezza dell'assembly, ma al mio sguardo incompetente sembra di vedere un dizionario inverso: molte parole per indicare un unico comando. Ci spieghi da dove nasce l'esigenza di avere comandi cosí prolissi, qual è l'utilità e, se ci sono, quali gli svantaggi?

E.G.: Facciamo l'esempio di `\stepcounter`:

```
\def\stepcounter#1{%
  \addtocounter{#1}\@ne
  \begingroup
    \let\@elt\@stpelt
    \csname cl@#1\endcsname
  \endgroup}
```

Si deve sapere che la lista dei contatori associati al contatore `foo` è la macro `\cl@foo` la cui espansione è del tipo

```
\@elt{gnat}\@elt{gnu}
```

e che si ha

```
\def\@stpelt#1{\global\csname c@#1\endcsname \m@ne\stepcounter{#1}}%
```

Vediamone un'ipotetica implementazione in expl3:

```
\NewDocumentCommand{\stepcounter}{m}
 {
  \user_counter_step:n { #1 }
 }
\cs_new_protected:Nn \kernel_counter_step:n
 {
  \user_counter_incr:n { #1 }
  \seq_map_inline:cn { g__kernel_counter_#1_linked_seq }
   {
    \user_counter_set:n { ##1 } { -1 }
    \user_counter_step:n { ##1 }
   }
 }
```

con opportune definizioni delle funzioni richiamate. Ancora: per aggiungere un contatore alla lista di quelli associati si adopera internamente `\@addtoreset`

```
\def\@addtoreset#1#2{\expandafter\@cons\csname cl@#2\endcsname {{#1}}}
```

Come sarebbe in expl3? Più o meno

```
\cs_new_protected:Nn \user_counter_link:nn
 {
  \int_if_exist:cTF { g__user_counter_#1_int }
   {
    \seq_gput_right:cn { g__kernel_counter_#1_linked_seq } { #2 }
```

```
    }
    {
     \msg_error:nnn { user } { counter-not-exist } { #2 }
    }
 }
```

La variabile di tipo seq che contiene i contatori associati sarebbe allocata quando si esegue `\newcounter`

```
\def\newcounter#1{%
  \expandafter\@ifdefinable \csname c@#1\endcsname
    {\@definecounter{#1}}%
  \@ifnextchar[{\@newctr{#1}}{}}
```

che diventerebbe

```
\NewDocumentCommand{\newcounter}{mo}
 {
  \user_counter_new:n { #1 }
  \IfValueT{#2}{ \user_counter_link:nn { #1 } { #2 } }
 }
\cs_new_protected:Nn \user_counter_new:n
 {
  \int_if_exist:cTF { g__user_counter_#1_int }
   {
    \msg_error:nnn { user } { counter-exist } { #1 }
   }
   {
    \int_new:c { g__user_counter_#1_int }
    \seq_new:c { g__kernel_counter_#1_linked_seq }
    \cs_new:cn { the#1 } { \arabic{#1} }
    % più qualcos'altro ...
   }
 }
```

Più lungo, ma si sa in ogni riga ciò che si sta eseguendo e su che cosa.

Già che ci siamo, trovo che, rispetto a

```
\@nocounterr{#2}
```

il corrispondente expl3

```
\msg_error:nnn { user } { counter-not-exist } { #2 }
```

sia molto più chiaro.

All'inizio la verbosità può sconcertare, ma alla lunga la chiarezza paga. E l'uso del prefisso, qui `user` per indicare qualcosa che ha a che fare con l'interfaccia utente (è solo ipotetico) permette di distinguere al volo di che tipo sia la funzione da eseguire e dove andarla a cercare.

Perché `\@stpelt`? Eh, bisognerebbe chiederlo a Leslie Lamport. Credo che la parte `elt` venga dal Lisp, mentre `stp` sta per "step". Nome che rimane impresso nella memoria, vero?

G.P.: Indubbiamente questa chiarezza paga. C'è stato qualche linguaggio che abbia ispirato questa sintassi o è una scelta autonoma degli sviluppatori di "LaTeX3" (lo chiamo cosí per brevita)?

E.G.: Credo sia un'invenzione di Frank Mittelbach.

G.P.: A questo punto è opportuno sentire Frank Mittelbach. Torno tra pochissimo da te. Ciao Frank. È trascorso un po' di tempo dalla tua intervista in cui c'era anche Dave Walden. Felice di risentirti. Dunque, possiamo ritenere te l'artefice della sintassi dei comandi (a questo punto forse è meglio chiamarle istruzioni) di expl3?

FRANK MITTELBACH: Ti rispondo sinteticamente perché al momento sto lavorando all'uscita della terza edizione di *The LaTeX Companion*. E poi c'è il progetto sul *tagged* PDF che preme… Comunque, sì, sono largamente io, insieme ai componenti del team dell'epoca. Era circa il 1992. Impossibile ricordare chi ha portato come contributo cosa.

G.P.: Hai preso ispirazione da un linguaggio preesistente o è stata una scelta dettata da altre motivazioni? E perché?

F.M.: Ma, sai, si prende sempre ispirazione da ciò con cui hai lavorato, ma in questo caso direi certamente di no: l'ecosistema di TeX è così differente che il tradizionale concetto di linguaggio non è molto adatto. Se penso ai criteri di progettazione, li trovo molto "anti":

1. tieni lontano (e nascondi) il più possibile alcune delle stranezze di un linguaggio di espansione di macro in generale, e in particolare alcune delle stranezze di TeX;
2. sii molto diverso dall'usuale TeX;
3. sii molto strutturato (non sempre riuscito) ma permetti all'utente di intuire come invocare un comando senza fargli avere troppe sorprese.

Riguardo al punto 1., è stato fatto in modo che

- i trucchi di espansione fossero ridotti al minimo e relegati nelle porzioni di codice più interne;
- per la maggior parte del tempo fosse ragionevole pensare ai comandi come a funzioni e procedure e non a macro, con input e output chiari ed effetti diretti e collaterali ben definiti;
- l'espansione fosse per lo più gestita alterando la segnatura dell'argomento; ciò ha un meccanismo sempre uguale che può essere applicato a ogni funzione di base;
- i concetti fossero unificati quando possibile, così, per esempio, ... i comandi condizionali sono sempre forniti con argomenti TF (ma i predicati di basso livello esistono ancora quando servono per ragioni di velocità).

Riguardo al punto 2.

- abbiamo sempre visto esserci un miscuglio di codice (e sarà ancora più preminente con lo strato di programmazione L3 del formato LaTeX) e dovrebbe essere semplice distinguere cos'è expl3 e cosa no;
- aiuta a rimanere nello spirito: non usi dei truccacci di programmazione TeX se solo ti mantieni al codice di livello interface3;
- quando gli utenti si trovavano davanti a parecchi _ e : per ogni dove

Riguardo al punto 3.

G.P.: Grazie, Frank, per la tua disponibilità, nonostante il pochissimo tempo a tua disposizione, e buon lavoro per il tuo libro. Torniamo a Enrico. Hai chiarito perfettamente il salto di qualità espressiva del nuovo linguaggio ma, a quanto pare, l'utente finale non ne beneficerà direttamente. Quali prevedi saranno i benefici indiretti per questa classe di utenti?

E.G.: Non ci sono benefici "visibili" per l'utente finale, quello che si limita a scrivere documenti senza bisogno di definire più che qualche semplice comando. Se non consideriamo la maggiore robustezza delle implementazioni.

Per l'utente più smaliziato sono a disposizione molti nuovi metodi, fra cui gli "hook". Già ora è possibile adoperarli (occorre LaTeX del 2020-10-01 o successivo), un meccanismo che generalizza di parecchio i comandi di etoolbox per eseguire codice a ogni chiamata di un ambiente, ma non solo. Pacchetti come 'atbeginshi' sono diventati obsoleti, perché il nucleo include le loro funzionalità: l'ovvio vantaggio, a parte non dipendere da un pacchetto esterno, è che questi "hook" sono definiti in modo uniforme.

È assai probabile che altri pacchetti di uso frequente, penso a enumitem, per esempio, saranno poco alla volta integrati nel nucleo.

Il grosso problema è mantenere la compatibilità con i vecchi documenti, in cui ambienti e comandi sono spesso modificati agendo a basso livello. Ma quando il team decide qualche modifica importante, gli autori dei pacchetti che potrebbero essere resi instabili vengono contattati suggerendo che cosa cambiare. Se si seguono gli aggiornamenti su CTAN, si può vedere come questo accada abbastanza di frequente.

Piccoli passi. Sarebbe bello poter ridefinire `\parbox` con le nuove funzioni per snellire e semplificare l'implementazione, ma troppi documenti e codice esistente si appoggiano a comandi di livello più basso che sparirebbero.

In ogni caso, quando si decide per una modifica, il codice 'antico' non sparisce e si può adoperare latexrelease per tornare indietro. Un vecchio documento potrebbe non funzionare, ma basta aggiungere una chiamata a latexrelease per far tornare il nucleo a una versione precedente.

G.P.: Torniamo alle cassandre da cui è partita l'intervista, le quali prevedono un futuro breve per TeX e derivati e ne evidenziano una inutilità didattica. Credo che il tuo impegno su più fronti di sviluppo del "nostro" programma di composizione la dica lunga sul tuo pensiero in merito, ma vorrei comunque il tuo "disegno" sul futuro (o sui futuri) di questo programma che reputo complicato e affascinante. E quanto lungo prevedi questo futuro?

E.G.: Quale futuro ha TeX? Nell'ambito scientifico non dovrebbe aver problemi a sopravvivere. Molte case editrici (Springer in testa) forniscono classi per i loro libri e riviste; arXiv chiede che gli articoli siano sottoposti come TeX: sono centinaia ogni mese. Purtroppo, molti ambiti scientifici sono irraggiungibili ma non perché LaTeX non sarebbe un valido strumento: il problema è la corsa alla pubblicazione nel minor tempo possibile di montagne di articoli che nessuno leggerà mai, anche perché contengono poco più che risultati di un esperimento o di uno studio di portata limitata: qualche tabella generata da un foglio di lavoro, poco testo di commento sempre essenzialmente uguale, una bibliografia sterminata generata automaticamente. Il che non è un giudizio di merito, sia chiaro, solo un'amara constatazione.

Ma c'è un campo nel quale ci possono essere sviluppi molto interessanti: l'accessibilità. Da qualche mese la PDF Association, che si occupa di tutto quanto riguarda il formato PDF ha un gruppo di lavoro che riguarda proprio LaTeX ed è mirato a fornire gli strumenti per l'accessibilità. Il LaTeX team ne fa ovviamente parte. Lo scopo, forse ambizioso, è di rendere un documento PDF prodotto con LaTeX accessibile (nel senso tecnico, pensiamo a nonvedenti o altre categorie di persone che non possono maneggiare lo strumento nel modo usuale). Lo strato per l'accessibilità sarà creato nel modo più trasparente possibile, quindi con minimo intervento dell'autore del testo.

> Se combiniamo con le capacità di programmazione proprie di LaTeX, molto più agevoli avendo a disposizione expl3, il progetto può dare nuova linfa al nostro sistema. E non è campato per aria: i primi risultati sono già disponibili anche se l'usabilità è ancora primitiva. Le ricadute saranno anche per gli utenti "normali": per esempio, hyperref sarà integrato nel formato, senza la necessità di modificare centinaia di comandi quando lo si vuole adoperare.
>
> Il team di LuaTeX si muove in altre direzioni, non sempre così comprensibili. Tuttavia non vedo del tutto impossibile che il nuovo LuaMetaTeX diventi il motore "definitivo" (anche se c'è parecchio lavoro da fare).
>
> In ogni caso, già oggi è possibile fare con XeLaTeX e LuaLaTeX cose che vent'anni fa sembravano al di là di Marte. E da quello che si vede nei siti che ne trattano, la base di utenti si espande. Le cassandre possono strillare quanto vogliono.

Ringrazio Enrico (e Frank) per aver esposto con voce autorevole una decisa precisazione alle affermazioni supportate da sensazioni e opinioni personali, ma non da dati di fatto, che tanto mi avevano colpito negativamente. Spero con questo di aver reso un servizio a quanti, come me, hanno avuto la stessa reazione.

Gianluca Pignalberi
(intervistatore)
g.pignalberi@gmail.com

Enrico Gregorio
(intervistato)
Enrico.Gregorio@univr.it

Frank Mittelbach
(intervistato)
frank.mittelbach@latex-project.org

Post per Instagram con LaTeX

Gianluca Pignalberi

Sommario Viene proposta una bozza di formato per comporre post per Instagram, formato ovviamente basato su LaTeX. Viene mostrato passo per passo l'intero processo.

Abstract We propose a format draft to typeset Instagram posts, obviously LaTeX-based. We show the whole process, step by step.

1. Introduzione

Da qualche mese ho un account Instagram (@texnico001) e, da buon ignorante in materia, ho trascorso i primi giorni a capire 1) cosa pubblicare, e 2) come farlo.

Per il cosa, dopo aver fatto un breve sondaggio tra alcuni amici già presenti su detto social network, ho scelto di seguire in parte i loro consigli e in parte i miei interessi. Per il come, invece, la premessa è un po' più lunga. Ho trascorso i primi giorni a guardare alcuni "contenuti", *post* e *stories* (storie), e mi sono reso conto che spesso contenuti di utenti diversi sembrano mostrare uniformità d'aspetto (sospetto l'utilizzo di una qualche app molto diffusa) e, soprattutto, mostrano la profondità tipica di quelli che Daniel Kahneman (2012) chiama *pensieri veloci*: nulla. Altri contenuti, specialmente quelli di divulgatori coscienziosi, sono, al contrario, molto profondi, ma talvolta fuori fuoco rispetto al mezzo: testo troppo lungo per le storie e per essere composto a epigrafe (centrato), scritto in corpo troppo piccolo e troppo poco interlineato.

Dopo qualche giorno di riflessione, ho deciso di pubblicare i miei "contenuti" sotto forma di post, che possiamo considerare uno *slideshow* in cui è l'utente a decidere quando cambiare slide. Per creare tali post non avrei potuto che usare LaTeX, in particolare LuaLaTeX. L'aspetto avrebbe dovuto essere rigoroso ma moderatamente colorato, il contenuto ben leggibile e stringato ed essere compreso, quando possibile, entro le dieci slide permesse dalla piattaforma per ogni post. In caso di esubero, sarebbero necessarî più post.

2. La "progettazione"

Terminata la fase di riflessione, è iniziata quella "progettuale", cioè di creazione di un disegno semplice delle slide. I punti in cui si articola il disegno della pagina sono i seguenti:

1. rapporto d'aspetto e dimensione di pagina;
2. gabbia;
3. font, dimensione e posizionamento del testo;
4. sfondo e colori.

Per quanto riguarda il punto 1, mi sono attenuto a uno dei possibili formati proposti da Instagram, quello verticale con rapporto d'aspetto 4:5. Per evitare di ottenere pagine eccessivamente grandi e, di conseguenza, piene di testo, ho scelto come dimensioni 10×12.5 cm. La gabbia (punto 2), quando diversa dalla pagina, è semplicemente centrata nel foglio e con margini scostati dai bordi della pagina di 0.5 cm per lato.

Per fare in modo che non ci fosse troppo testo da leggere, anche il punto 3 (con le eccezioni di alcuni vecchî racconto brevi) ha avuto come linea guida la leggibilità e la stringatezza, come se fosse lo strillo di un giornale o di una locandina. Quindi poche righe di testo ben visibile e centrato (imbandierato a sinistra in alcuni casi) per slide. La maggior parte dei post è composto con un font di 36 pt e interlinea 42 pt. Eccezioni ricorrenti sono i racconti (font 16 pt e interlinea 20 pt) e i post sulla geometria spicciola (font 24 pt e interlinea 1 cm per avere come linea di base i quadretti del "foglio"). Il font usato per tutti è Linux Libertine.

Infine, il punto 4 ha come criterî l'adozione di un colore di sfondo uniforme per argomento[1] e l'uso di massimo quattro colori per testo e sfondo; le foto sono ovviamente escluse e anche un'illustrazione/logo ha disatteso la regola.

Questo aspetto, a detta di qualcuno dei miei sparuti *follower*, rende i post immediatamente riconoscibili. Quasi un marchio distintivo o, se vogliamo, un'identità visiva.

3. La realizzazione

Vediamone la realizzazione pratica. Per evitare le complicazioni di un pacchetto *ad hoc*, per esempio beamer, ho usato inizialmente la "semplicissima" classe article con l'aggiunta di pochi, selezionati pacchetti. Dopodiché, non essendoci nelle slide una struttura, sono passato alla classe minimal.

Avendo scelto di compilare con LuaLaTeX, il preambolo tipico dei miei post è il seguente:

```
\documentclass[a4paper]{minimal}

\usepackage[english,italian]{babel}
\usepackage{fontspec}
\setmainfont[Contextuals=Alternate]{Linux Libertine O}
\RequirePackage[dvips=false,pdftex=false,vtex=false,
                xetex=true,driver=none]{geometry}
\geometry{papersize={10cm,12.50cm},body={9cm,10.5cm}}
\usepackage{xcolor}
```

con le poche variazioni necessarie a variare la gabbia in base al post.

A seconda delle esigenze, a questi pochi pacchetti aggiungo graphicx per includere le immagini (o per effetti particolari sul testo), siunitx, Ti*k*Z, textpos (per posizionare elementi in posizioni specifiche delle slide), soul (finora usato solo per ottenere del testo barrato), unicode-math (per poter caricare libertinus come font per la matematica).

La maggior parte dei miei post ha una struttura semplicissima (e mista TeX/LaTeX):

1. Pratica ispirata da Magnus (1994) e da un'opera mai realizzata dello stesso autore, *Il conte notte*, in cui i disegni sono realizzati su cartoncini Pantone.

```
\begin{document}
\pagestyle{empty}
\fontsize{36}{42}\selectfont
\pagecolor{green!50!white}
\color{gray}

% Ogni slide inizia così
\phantom{}\vfill
\centering

\vfill
\newpage

\end{document}
```

La figura 1 mostra uno dei tanti post realizzati con quest'impianto di base (ma con colori diversi da quelli qui riportati).

Come detto, poi, ci sono delle eccezioni per cui il testo è un po' più piccolo, o imbandierato a sinistra e senza margini, ma finora l'aspetto principale è stato questo. Ho allo studio alcune modifiche che potrebbero rendere piú fruibile ogni post, ma senza stravolgere questo impianto di base. Per esempio, è previsto un file di configurazione che stabilisca tutte le impostazioni in base alla presenza di un'opzione. Forse sarebbe opportuno adottare un comando simile a `\maketitle` per comporre la prima pagina del post senza interventi manuali. Infine, specie per le recensioni di libri in pillole, è opportuno che i libri menzionati vadano nella prima slide, in modo da rendere più fruibile il post.

4. Passo finale

Una volta generato il PDF finale, bisogna convertirlo in una serie di immagini. Ognuno sceglierà il programma con cui si trova più a suo agio. Io trovo molto comodo `pdftopng`, filtro che prende in input un file `.pdf` e restituisce una serie di tanti file `.png` quante sono le pagine del PDF e numerati progressivamente. Questo rende immediata il corretto caricamento dei file nell'ordine previsto.

All'atto del caricamento delle slide su Instagram, dovrà essere selezionato manualmente l'aspetto 4:5 del post, pena l'adozione di default di un formato quadrato che taglierebbe le estremità verticali di ogni post.

5. Conclusioni

Nell'intento di creare una serie di post su Instagram elegantemente composti e facilmente riconoscibili ho creato un documento semplice e funzionale. Per ora il codice è impostato a mano. Nell'immediato futuro è prevista l'adozione di una serie di automatismi di configurazione e impostazione del titolo, simili a `\maketitle`.

Figura 1. Uno dei miei post su Instagram.

Riferimenti bibliografici

Kahneman, Daniel. 2012. *Pensieri lenti e veloci.* Milano: Mondadori.

Magnus. 1994. *Il principe nel suo giardino.* Bologna: Granata Press.

Gianluca Pignalberi
g.pignalberi@gmail.com

The wrapfig2 package

Claudio Beccari

Abstract Package wrapfig is extremely useful, but is delicate. It has many limitations because it has to compute the space in terms of number of lines necessary to indent the surrounding text so that it can wrap the insertion. Its documentation explains the limitations the users should be attentive to.

This short paper describes a new package, wrapfig2, that is backwards compatible with the original wrapfig, but aims to let the users have a better control on the number of lines. At the same time it defines a new environment, wraptext, that exploits the same ideas in order to insert a framed text block on a shaded background wrapped by the surrounding text.

Sommario Il pacchetto wrapfig è molto utile, ma è delicato. Presenta diverse limitazioni dovute alla difficoltà di calcolare correttamente di quante righe deve essere il rientro che consenta l'inserimento del materiale da contornare con il testo.

Questo breve articolo descrive un nuovo pacchetto, wrapfig2, retrocompatibile con il pacchetto originale wrapfig, che consente all'utente di avere un controllo migliore sul numero di righe del rientro del testo circostante. Contemporaneamente definisce un nuovo ambiente, wraptext per inserire un blocco di testo incorniciato e su uno sfondo sfumato, in un testo che lo circonda.

1. Introduction

Package wrapfig by Donald Arseneau has been available for several years; the last upgrade to our best knowledge dates back to 2003; it is very useful to insert small objects (figures, tables, and similar ones) in an indented part of the normal text; it is possible to configure the insertion so as to specify the overhang into the margin, the total width of the insert, even the total height of the necessary indention; it is possible to specify the margin where to protrude the insertion and even to float the inserted material. The available environments are wrapfigure, and wraptable; they depend on wrapfloat⟨*float kind*⟩[1], where the ⟨*float kind*⟩ is already available.

Such insertions may be fixed or floated; placement codes may be specified by the user.

The general syntax of this environment is the following (the example refers to a figure, but the syntax is similar for tables and other objects):

```
\begin{wrapfigure}[⟨total number of indented lines⟩]
    {⟨position⟩}[⟨overhang⟩]{⟨insert width⟩}
    ⟨material to be inserted⟩
\end{wrapfigure}
```

It should be sufficient; actually if the insert is pretty long the environment may have difficulties in computing the correct number of indented lines; the reasons are explained in the wrapfig package documentation; its author warns the user about the package idiosyncrasies.

1. Users are familiar with the figure and table float kinds, but by means of the float package it is possible to define other kinds.

Most of these idiosyncrasies can be avoided by choosing the right place in the source file where to specify the environment: it should not follow too close a page break, a sectioning command, a list, and so on. But according to our experience one of the most annoying features is the difficulty of counting the correct number of indented lines; we found that it is simpler to count the (generally small) number of lines to add to or subtract from the line number computed by the original software.

There are a couple of reasons for following this corrective approach, rather than the original approach.

1. The first time any wrapped insertion is specified, the users have no idea of how many indented lines are necessary; therefore they do not specify the optional argument ⟨*total number of indented lines*⟩; they let the software compute the necessary number of such lines. More often than not such number is not the most suitable one, and often the white space below the insertion is too high; or it may happen that the computed number of lines is too small so that the inserted material overlaps the neighbouring text. In the first case it is necessary to diminish the computed number of lines, while in the second case such number should be increased.
2. It is simpler to count the small number of extra indented lines, or the small number of overlapped full lines.

2. How to compute the correction line number

The new language LaTeX 3 offers many possibilities. Since the fall of 2020, most, if not all, the xparse functionalities are included into the LaTeX kernel, and, except in special (deprecated) cases, it is not necessary to load any package: the xparse documentation is still available, but the package is not explicitly required. This wrapfig2 uses one of the *deprecated* cases, therefore it provides to load such package.

On the opposite, the computing LaTeX 3 functionalities are not yet included into the kernel; therefore the xfp package is loaded by wrapfig2; among such functionalities the `\fpeval` function can operate also on length values and can perform several kinds of rounding operations.

We found more efficient to redefine known environments, or define new ones, with similar syntaxes; we therefore redefined wrapfigure and wraptable and defined the new wraptext environment.

The core of the computation of the corrective line number is the following.

1. By using the environment arguments, store the object to be wrapped into a box and determine its height.
2. Use the `\fpeval` to compute the ratio of the above height to the current base line height, and let `\fpeval` round such ratio to the nearest integer.
3. Add a small fixed correction number and eventually add the user specified positive or negative correction value.
4. Pass such computed number of lines as the first optional argument to wrapfloat with the specific ⟨*float kind*⟩ string.

Following the above policy, both original wrapping environments are redefined as described in the following subsections.

2.1. Necessary packages and preliminary settings

The redefinitions we devised require some packages, therefore the following ones are required and the new package provides to load them if they are not already loaded.

xfp provides the powerful LaTeX 3 commands `\fpeval` and `\inteval`; the former is used to make calculations and round the results to a specified number of fractional decimal digits. The second is similar but operates on integer operands.

curve2e is going to be used with the wraptext environment so as to draw the emphasising frame and a coloured background around the text. See the specific subsection below.

etoolbox is always handy when code has to be written so as to render it more easily readable and maintained.

3. Wrapped figures

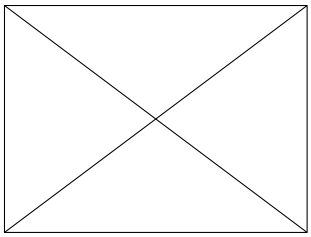


Figure 1. A rectangle with diagonals

The main redefinition of the wrapfigure, wrspatable and wraptext environments is commented in a further section that describes the relevant parts of the code; here we are more interested to show what is possible to do.

Let us first describe the syntax of these revised or new environments. The syntax shown in the box below refers to version 6 of this package. Some options specify how to select fallback settings in order to get the functionalities of versions 5 or 4). Please read the wrapfig2 package documentation.

```
\begin{wrapfigure}[⟨total indented lines number⟩]{⟨position⟩}[⟨overhang⟩]{⟨width⟩}⟨star⟩
⟨inserted figure⟩
\end{wrapfigure}

\begin{wraptable}[⟨total indented lines number⟩]{⟨position⟩}[⟨overhang⟩]{⟨width⟩}⟨star⟩
⟨inserted table⟩
\end{wraptable}

\begin{wraptext}<⟨indented lines number correction⟩>{⟨position⟩}[⟨overhang⟩]{⟨width⟩}
⟨optional style settings⟩
⟨inserted text block with other settings and corner radius of curvature⟩
\end{wraptext}
```

The package code for inserting a figure is the following; the one for inserting a table is very similar.

```
\NewDocumentEnvironment{wrapfigure}{o m o G{\z@}}%
    {\wrapfloat{figure}[#1]{#2}[#3]{#4}}%
    {\endwrapfloat}
```

As it is can be seen from the code, the environment contents is first boxed into the box register `\NWFbox` with the help of the lrbox environment. Then, depending on the presence of the optional asterisk the total number of lines of the object to be included is measured; if the asterisk is present the corrective argument is algebraically added to the estimated height; if the asterisk is absent then if the total number of lines is specified, such a number is used as in the original environment; if the user did not specify any value, the number of indented lines is evaluated by the original environment. A specific figure might be inserted by one of the following statements

```
\begin{wrapfigure}{l}{50mm} figure code \end{wrapfigure}
\begin{wrapfigure}[10]{l}{50mm} figure code \end{wrapfigure}
\begin{wrapfigure}[3]{l}{50mm}* figure code \end{wrapfigure}
```

The first statement is possibly the first one the users write in their document; the second is the one that exploits the original environment functionalities, while the third uses just the correction value to add after controlling what the first statement failed to compute correctly.

4. Wrapped tables

first	second
third	fourth

With (small) tables the situation is very similar; the difference relies on the fact that small tables generally are not typeset with a prescribed width; generally the user is not so good in estimating a suitable width for a table that relies on the widths of all the tabular cells. Therefore the last mandatory argument of the wraptable opening statement may be *omitted* thanks to a feature of the xparse package that may use brace delimited optional arguments; if this argument is omitted, together with the braces, the software is capable of determining the object width and may provide accordingly.

The code for the redefinition of the new wraptable environment is not so different from that of wrapfigure.

```
\NewDocumentEnvironment{wraptable}{o m o G{\z@}}%
    {\wrapfloat{table}[#1]{#2}[#3]{#4}}%
    {\endwrapfloat}
```

It goes by itself that the user has to insert the whole table code using, for example, the tabular environment. Therefore the user has a wider range of possibilities to change the size of the table, as well to correct the estimated height of the actual table. Since the underlying code is the one of the wrapfloat environment, it is possible that the various packages that deal with tables and/or with their captions still work properly with the new definition. In any case the result is still acceptable as with figures.

Caution

In both redefinitions of the standard wrapping environments, if the optional asterisk is specified while the first optional argument is omitted, no error is raised and no warning is output; simply the presence of the asterisk is ignored and the wrapped figure or table is typeset with the original environments; if the authors use the asterisk, they should pay attention to enter into the first optional argument the *corrective* number of lines, not the absolute one.

5. Wrapped text

Recently a question was posted on stackexchange in order to get help for creating some sort of small framed medallions on a coloured background to be wrapped by text as well as a figure.

Text, text, text, text, text, text, text, text, text, text, text.

Of course it would not be too complex to create the medallion with the functionalities of the tcolorbox and saving the graphic result into a box to be inserted as a wrapped figure by means of the wrapfigure environment. But in this last version 6 of the wrapfig2 package we preferred to draw the frame and is background by means of the picture environment extended with the curve2e package functionalities.

The participants to that thread on stackexchange suggested the WrapText environment based on the use of the original wrapfigure. Unfortunately the vertical space computed by wrapfigure more often than not was too small, and the medallion bottom would overlap the following normal text. A good friend of ours asked us if it was possible to maintain the same structure based on wrapfigure, but with some means to correct the actual number of lines of the indented normal text.

```
\NewDocumentEnvironment{wraptext}{O{0} m O{Opt} G{0.5\columnwidth}}%
{%                                                          Open environment
  \insertwidth=#4\WFscalewidth
  \def\textplacement{#2}%
  \def\textcorrection{#1}%
  \def\textoverhang{#3}%
  \bgroup\edef\x{\egroup\noexpand\wrapfloat{text}%
    [\textcorrection]{\textplacement}[\textoverhang]{\insertwidth}*}\x%
  \def\caption{\unskip%                      Redefine the \caption command
    \refstepcounter\@captype
    \let\@tempf\@caption
    \unless\ifcsname @float@c@\@captype\endcsname
      \expandafter\expandafter\let
        \expandafter\@tempf\csname @float@c@\@captype\endcsname
    \fi
    \@dblarg{\@caption\@captype}%    Use the internal LaTeX kernel commands
    }%
}{%                                                        Close environment
\endwrapfloat\ignorespaces}%
```

The user command syntax is the following:

```
\begin{wraptext}[⟨indented lines number correction⟩]{⟨location⟩}[⟨overhang⟩]{⟨width⟩}
    ⟨optional colour settings⟩
  \includeframedtext[⟨insertion measure⟩]{⟨text to frame⟩}[⟨settings⟩][⟨radius⟩]
\end{wraptext}
```

The solution of this problem was the motivation for designing the environments described in this article. As we mentioned above, the syntax of this environment is slightly different from the other two, but the philosophy is the same; as a matter of fact all environments use the

same wrapfloat environment behind the scenes. If the wrapped framed text needs a caption (something we discourage, since we believe that the wrapping text is such as to describe why that text has been wrapped) it is necessary to use the `\caption` command with its arguments; with the `texstackexchange` solution and its default parameters, the caption would turn out to be in the form "Figure 3.2", but the label "Figure" does not seem to be the most appropriate; with version 6 the default value for the caption label is "Text", but by means of the babel or polyglossia language handlers it is possible to adapt it to any language.

The above code is not simpler than the ones used for the redefinitions of wrapfigure and wraptable, not only because there is no need to be backwards compatible, but also because some parameters, such as the text box width, are preset. The environment allows the user to specify the text box width; nevertheless a width shorter than the default might pose some justification problems, while a longer one would be too intrusive; depending on the text to be wrapped and on the page layout, we recommend to maintain the text box width in the range from 40% to 60% of the current column width; remember that while typesetting in one column mode the `\columnwidth` value equals that of `\textwidth`.

6. Performance

We don't know if this new extended version wrapfig2, performs better than the original version by Donald Arseneau. We partially modified his code, in particular the commands that define the total height of the wrapped object, or the total indention of the wrapping lines. We adopted the robust definitions of the LaTeX 3 language.

But certainly the idiosyncrasies that Arseneau described in his original documentation are still there; since the original patches to render the wrapping procedure compatible with other classes and/or packages have not been modified, it is possible they have the same pros and cons.

What we noticed is that the two revised environments and the new one are performing pretty well even without resorting to the correction mechanism that was the main reason for upgrading the package; probably this is due to the different way of boxing the material to be wrapped, or to the well conceived approach used by Arseneau.

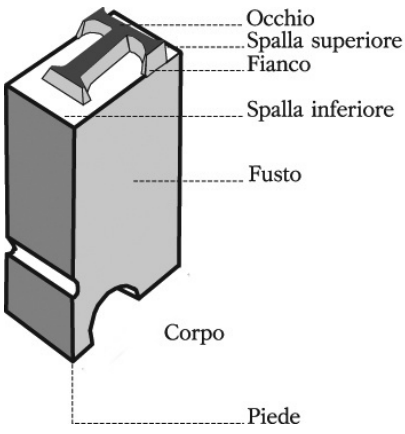


Figure 2. Italian names of a metal type details

The small examples we showed in the previous sections do not actually require any correction except for the wrapped figure; probably with taller objects the idiosyncrasies are more easily triggered. In any case the small wrapped figure 2 did not require any adjustment.

Notice, though, that the ⟨*width*⟩ of the object to be inserted refers to the true or estimated horizontal dimension of that object; if the width estimation is difficult, as with tabular material, it is better to specify a larger value and to specify `\centering` in order to insert the thinner object within the wider space that wrapfig2 reserves to the object. If the larger estimated width is too large, on a second compilation run it is possible to specify a more suitable value; when the specified width value is

too small a black bar appears to the right of the inserted object. Even better: it is possible to omit the ⟨*width*⟩ specification; as we said before, the ⟨*width*⟩ parameter is a *braced optional argument* such that the software computes the necessary width quite well with figures and tables; this is what we did with figure 2.

In any case the inserted object must not have white margins around; for the wrapped text the built-in frame does not have any margins; for tables we already explained the difficulty of estimating their width, but they don't have any built-in white margins; the situation may be critical with images; it is important to carefully crop them so as to eliminate all four margins; resorting to the standalone package functionalities may be precious.

Acknowledgements

A hearty thank you goes to Donald Arseneau who first created the wrapfig package; many users are grateful to him for this package. I warmly acknowledge his original package paternity.

Heinrich Flech submitted me the described problems and suggested to implement more flexible correction solutions; I thank him very much.

I gratefully thank Lorenzo Pantieri who spotted some glitches in the first versions of this new package.

I also heartily acknowledge the suggestions concerning various testing sections, and some smart solutions to some problems received from Juan Luis Varona Malumbres.

Claudio Beccari
claudio.beccari@gmail.com

Controlling captions, fullpage and doublepage floats: hvfloat

Herbert Voß

Abstract The package hvfloat defines macros which place objects and captions of floats in different positions with different rotating angles for the object and caption. The object can fill a full column, a full page or full doublepage, with or without taking margins into account.

Sommario Il pacchetto hvfloat definisce macro che posizionano oggetti e didascalie di float in posizioni diverse con angoli di rotazione diversi per l'oggetto e la didascalia. L'oggetto può riempire un'intera colonna, una pagina intera o un'intera pagina doppia, con o senza considerare i margini.

1. Introduction

The well-known floating environments like figure and table are easy to handle if there is only one object and one caption which fits into the current page text layout. If you want a caption rotated and beside the object (an image, tabular, …) then you need some LaTeX knowledge or a package which does the rotation and the checking of the current page number if you want to place the rotated caption for a twocolumn document into the outer margin.

All this can be simplified by using the package hvfloat which has a variety of possible options for the floating object and caption. The package is loaded in the usual way:

```
\usepackage[options]{hvfloat}
```

The package has options `hyperref`, `nostfloats`, and `fbox`. The latter is only used for locating spacing problems in the document: objects and captions are framed, so unwanted whitespace can easily be seen. With `nostfloats` one can prevent the loading of the package stfloats, which allows bottom floats in a twocolumn document. This option is needed only in rare cases where a package conflict between stfloats and another package exists. With `hyperref` the package of that name is loaded.

If you would like to reset the default for the float position parameters to `htp` (here, top and page) (the default is `tbp`, top, bottom, and page), then you can load the helper package hvfloat-fps. It knows the optional arguments `table`, `figure`, and `all`. If you have a document with a large number of floats and relatively short text you can load the package with

```
\usepackage[all=!htb]{hvfloat-fps}
```

The exclamation allows LaTeX to ignore the internal parameter settings for the floats, e.g. the number of floats on one page Mittelbach et al. 2004.

Usually several LaTeX runs will be needed until hvfloat knows whether figures are on even or odd page and to get all the references correct. The usual warning '`Label(s) may have changed`' will be shown if another compilation is needed.

2. Dependencies

The following packages are loaded by default:
afterpage, caption, expl3, graphicx, ifoddpage, multido, picture, stfloats, subcaption, trimclip, xkeyval.

3. The macros and optional arguments

The three main macros are `\hvFloat`, `\hvFloatSet`, and `\hvFloatSetDefaults`. The syntax for calling them is somewhat complex. Optional arguments are gray shaded:

```
\hvFloat * [options] + {float type}{floating object}
     [short caption] {long caption}{label}
\hvFloatSet{key=value list}
\hvFloatSetDefaults
```

The star version of `\hvFloat` is explained in section 4 on page 30 and the optional `+` is explained in section 7.2 on page 41.

The `\hvFloatSet` macro allows the global setting of parameters via the given keyword=value list, while `\hvFloatSetDefaults` sets all parameters to their default values, as shown in Table 2 on page 28.

If `\hvFloat` is given an empty second argument for *float type*, it switches by default to a nonfloat object and activates the option `onlyText` (see Table 2). The *short caption* is a second optional argument; if given, it specifies, as usual, the caption entry for the `\listof. . . .` All other arguments are mandatory but may be empty.

Some other macros are defined, mostly for use in the hvfloat implementation, but they can also be used for a user's own purposes. Only `\tabcaption` should be placed at the top of an object.

```
\figcaption [short caption] {long text}
\tabcaption [short caption] {long text}
\tabcaptionbelow [short caption] {long text}
```

They are used for the `nonFloat` keyword, where these macros write captions in the same way but outside of any float environment. The default caption cannot be used here. It is no problem to use the `\tabcaption` command to place a caption anywhere, for instance here in an inline mode:

Table 1. A caption with neither sense nor object.

In this case a label should be put inside of the argument and not after the command `\tabcaption`, so that a reference to the nonexistent object Table 1 will still work. Source for this:

```
It is no problem to use the \verb|\tabcaption| command ... here in an inline mode:
\tabcaption[The caption without sense ...]{A caption with neither sense nor
  object.\label{dummy}}
In this case a label should be put inside the argument ... so that a reference to
the nonexistent Table~\ref{dummy} will still work.
```

With the macro `\hvDefFloatStyle` one can define a style to be used instead of the individual setting. Internally the style is saved in a macro named `\hv@`*name*.

`\hvDefFloatStyle{`*name*`}{`*setting*`}`

The possible keywords are listed in the rotated and full page Table 2 on the next page. To make this table, we first save it in the predefined box `hvOBox` as a `tabularx` with the tabular width of the current textheight. A `tabularx` cannot be used as an argument to `\hvFloat`. This is the reason we use the intermediate box:

```
\begin{lrbox}{\hvOBox}\small
  \begin{tabularx}{\textheight}{@{} l>{\small\ttfamily}cX @{}}\toprule
   \emph Keyword & \emph Default ...
[...]
  \end{tabularx}
\end{lrbox}
```

Then, to typeset the table, we use the keyword `rotAngle`, which rotates object and caption together:

```
\hvFloat*[floatPos=p,rotAngle=90,capPos=top,capWidth=w,useOBox=true]{table}{}
  {The optional keywords for the \texttt{\textbackslash hvFloat} macro.}
  {tab:options}
```

3.1. Caption positioning

By default the caption is set below the object and the macro `\hvFloat` behaves like the usual figure or table environment. With the keyword `capPos` and the value `before`, the caption can be placed beside the object. For small objects (smaller than a column/page), `before` is equivalent to `left`. Thus, here is the code for our first example:

```
\hvFloat[capPos=left]{figure}{\includegraphics{frose}}{A short caption beside a
    figure [...] without a label.}{}
```

If the caption is shorter than the possible width it is horizontally centered. The vertical position is by default also centered. This can be changed by the optional argument `capVPos`. The formatting can be modified by the optional arguments of the (already-loaded) package caption. They can be specified to `\hvFloat` via the optional argument `capFormat` (see Figure 2 on page 29). The caption is also rotated by setting `capAngle=90`, which is a counter-clockwise rotation:

```
\hvFloat[capPos=right, capAngle=90,capWidth=h, capFormat={font=sf}]{figure}
  {\includegraphics{frose}}{A caption in sans [...], to the right [...], as
    wide as [...], and rotated by 90\textdegree [...]}{fig:1}
```

Table 2. The optional keywords for the `\hvFloat` macro.

Keyword	*Default*	*Description*
`floatPos`	`tbp`	This is the same default placement setting as in standard LaTeX; maybe not always the best setting.
`rotAngle`	`0`	The value for the angle if both the object and the caption should be rotated together.
`capWidth`	`n`	The width of the caption. Can be `n` for a natural width given by the current linewidth, `w` for the width of the object, `h` for the height of the object, or a scale factor for `\columnwidth`.
`capAngle`	`0`	The integer value for the angle if the caption should be rotated. Positive is counter-clockwise.
`capPos`	`bottom`	The position of the caption relative to the object. Possible values: `before`: *always* before (left) from the object. `top`: *always* on top of the object. `left`: *always* before (left) from the object, but on the same page in twocolumn mode. `after`: *always* after (right) from the object. `bottom`: *always* on the bottom of the object. `right`: *always* after (right) from the object, but on the same page in twocolumn mode. `inner`: in twoside mode always typeset at the inner margin. `outer`: in twoside mode always typeset at the outer margin. `evenPage`: in twoside mode with fullpage objects always on an even page. `oddPage`: in twoside mode with fullpage objects always on an odd page.
`capVPos`	`center`	Only used when `capPos=left | right`; in these cases, the caption can be vertically placed at the `bottom`, `center` or `top`.
`objectPos`	`center`	Horizontal placement of the object relative to the document. Possible values are (**l**)eft, (**c**)enter, (**r**)ight.
`objectAngle`	`0`	Integer value for the angle if the object should be rotated. Positive is counter-clockwise.
`floatCapSep`	`5pt`	Additional space between the object and a left- or right-placed caption.
`useOBox`	`false`	Instead of passing the object to `\hvFloat`, with `useOBox=true` the contents of the predefined box `\hvOBox` is used.
`onlyText`	`false`	The caption is printed as normal text with no entry in any list of …
`nonFloat`	`false`	The object isn't put in a floating environment, but printed as standard text with an additional caption. The float counter is increased as usual and can be referenced.
`wide`	`false`	The float can use `\textwidth` + `\marginparwidth` as horizontal width.
`objectFrame`	`false`	Put a frame with no separation around the float object.
`style`	*none*	Use a defined style.
`capFormat`	*none*	Define formatting options for `\caption`; see documentation of package caption.
`subcapFormat`	*none*	Define formatting options for `\subcaption`.
`fullpage`	`false`	Use a complete column in twocolumn mode.
`FullPage`	`false`	Use the full text area for the object.
`FULLPAGE`	`false`	Use the full paper width/height for the object.
`doublePage`	`false`	Use the text area on a doublepage with additional text.
`doublePAGE`	`false`	Use the text area on a doublepage without additional text.
`doubleFULLPAGE`	`false`	Use the paperwidth on a doublepage without additional text.
`vFill`	`false`	Put a `\vfill` between every two objects in a multi- or subfloat.
`sameHeight`	`false`	use the same text height on both pages for a `doublePage` object.

Figure 1. A short caption beside a figure object (`capPos=left`) without a label.

Figure 2. A caption in sans serif (`capFormat={font=sf}`), to the right of the object (`capPos=right`), as wide as the object (`capWidth=h`), and rotated by 90° (`capAngle=90`).

The caption's vertical position is controlled by the keyword `capVPos` which accepts the values `top`, `center`, and `bottom`. The `capPos=inner` setting is explained later (§5.2, p. 33). Typographically, a side caption for images should usually be at the bottom and for a table at the top of the object (Figure 3).

```
\hvFloat[capPos=inner,capVPos=bottom,objectAngle=180]{figure}
  {\includegraphics{frose}}{This caption is at the inner margin [...],and
   vertically at the bottom [...], and the object is rotated [...]}{fig:11}
```

Figure 3. This caption is at the inner margin (`capPos=inner`, see p. 33), vertically at the bottom of the object (`capVPos=bottom`), and the object is rotated 180° (`objectAngle=180`).

3.2. The caption width

For a caption beside the object the horizontal justification is by default centered if the total width of object and caption are less than the current column/line width. The caption width itself can be controlled by the keyword `capWidth`, which can be set to `n` (natural width), `w` (width of the object), `h` (height of the object), or a value by which to scale `\columnwidth`. Figure 2 on the previous page shows the use of `capWidth=h`, which is used for rotated captions beside the object and Figure 4 shows a caption above the object with the same width.

```
\hvFloat[capWidth=w,capPos=top,capAngle=180,objectAngle=90]{figure}
  {\includegraphics{frose}}{A 180\textdegree-rotated caption above [...] with the
    same width.}{fig:1a}
```

Figure 4. A 180°-rotated caption above a 90°-rotated object, with the same width.

4. The star version `\hvFloat*`

In `twocolumn` mode the floating environment can occupy both columns using the star version `\hvFloat*`. This is analogous to the environments figure* and table*.

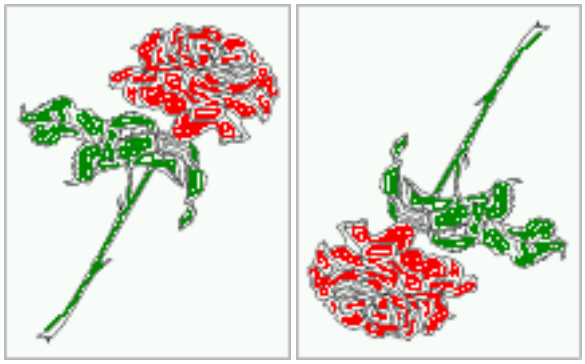


Figure 5. A longer caption to the right of the object (`capPos=right`), and vertically at the bottom of the object (`capVPos=bottom`). It spans both columns (`\hvFloat*`) and may be at the top or bottom of the page.

Table 3. Additional keywords for the `\includegraphics` macro.

name	width=	height=	keepaspectratio=
fullpage	\columnwidth	\textheight	false
FullPage	\textwidth	\textheight	false
FULLPAGE	\paperwidth	\paperheight	false
doublefullPage	2\paperwidth-2in -2\evensidemargin		true
doubleFULLPAGE	2\paperwidth	\paperheight	false
doubleFULLPAGEbindCorr	2\paperwidth−2\bindCorr	\paperheight	false

If possible, the floating environment will be placed at the top of the following page or at the bottom of the current page. The latter needs the package stfloats which is loaded by hvfloat by default. (stfloats cannot place a float at the bottom of the first page of an article or chapter when using the core LaTeX document classes; these classes also include code that prevents placement of a float at the top of the first page.) Placing the float across both columns within the text area is not possible. Here is the code for the following example (Figure 5 on the preceding page):

```
\hvFloat*[capVPos=bottom,capPos=right]{figure}
 {\includegraphics{frose} \includegraphics[angle=180,origin=c]{frose}}
 {A caption to the right [...], It spans both columns [...]}{fig:2}
```

The same can be seen in Table 3, which also spans two columns (we'll discuss the content of that table later). Internally the number of possible floating objects on top of the page is controlled by the parameters `\topnumber` (onecolumn mode) and `\dbltopnumber` (twocolumn mode). They are preset for this documentclass (ArsTeXnica) to 2 and 2 and differ for other document classes. For doublepage objects the values will temporarily be changed to 1.

5. Full column or fullpage objects

As mentioned in Table 2 there are three keywords for fullpage objects:

- `fullpage` for a complete column or page in a onecolumn mode,
- `FullPage` for a complete text area of a page or both columns in a twocolumn mode, and
- `FULLPAGE` for the complete paper area without leaving any margin.

This refers to the reserved space which `\hvFloat` will use when typesetting the object and caption. The object itself can be smaller than a full column or page. Package hvfloat defines five additional optional arguments for the package graphicx which can be used together with `\includegraphics` to make the code a bit shorter. They are listed in Table 3. The so-called bind correction is additional free space at the inner margins of a twoside document.

In general, the interface is the same whether using the complete text area or the complete paper area for the floating object; the only difference is `fullpage` vs. `FULLPAGE`. By default, such a page will have no page number, no header, and no footer, and the pagestyle is `empty`.

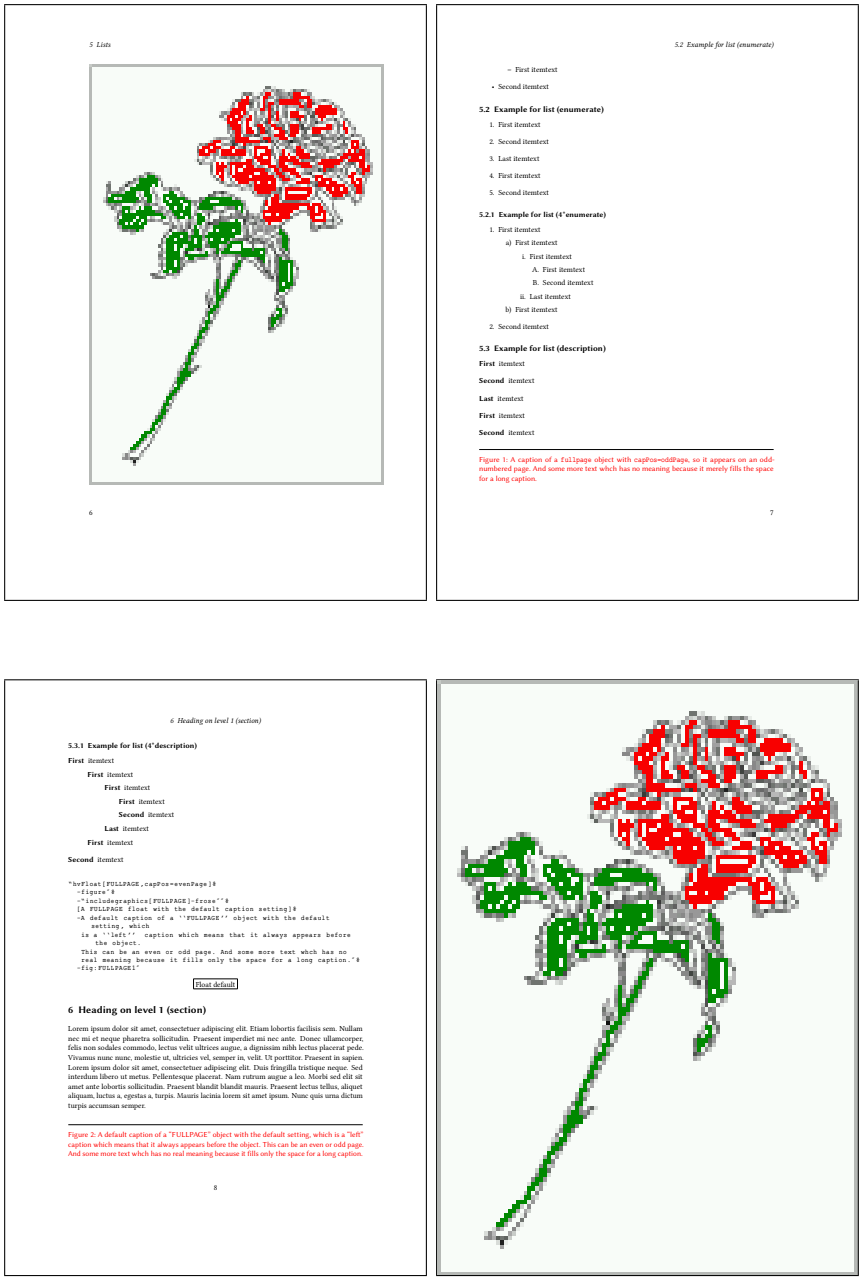


Figure 6. Twoside documents, onecolumn mode.
Top: a `fullpage` float and `capPos=oddPage` (example document `odd2s1c.tex`, pp. 6–7);
bottom: `capPos=evenPage` and a `FULLPAGE` float (example document `paper-even2s1c.tex`, pp. 8–9).

The following pages are printed in the \twocolumn mode to show that all full column, page, and double page examples also work in this mode.

Setting the keyword keepaspectratio to false only makes sense for images which have nearly the same ratio as the current column or page height/width. Using a full column or page for an object implies to put the caption on the preceding or following column/page. For a twocolumn document this should *always* be the opposite column on the *same* page and for twoside documents the opposite page. Only for doublepage objects (left–right pages) the caption must be on the preceding or following column/page, by default at the bottom of that page or column.

A label, which is defined by \hvFloat always points to the image, not to the caption. This makes no difference for the default floats, where the image and caption are on the same page. For fullpage or doublepage objects, however, the macro internally defines additional labels; one pointing to the caption (defined by label ⟨*label*⟩-cap) and, if it is a doublepage object, another pointing to the second (right) part of the object (defined by label ⟨*label*⟩-2).

All labels, the given one ⟨*label*⟩ and the internal ones ⟨*label*⟩-cap and ⟨*label*⟩-2, will point to the same object counter, but possibly to different page numbers. An example is shown in section 6, where Figure 13, defined with label fig:dP, has its caption on page 39 and its image on pages 38 and 39. The following table shows the behavior:

	fig:dP	fig:dP-cap	fig:dP-2
\ref{..}	13	13	13
\pageref{..}	38	39	39

5.1. *Twoside and onecolumn mode*

In a twoside document with onecolumn mode, a fullpage object and the corresponding caption should be on facing pages (left–right). This can be specified with the keyword capPos and the values evenPage or oddPage. To save space we show only the output of two example documents (Figure 6 on the facing page). The upper pair of pages uses the following settings:

```
\hvFloat[fullpage,capPos=evenPage]
  {figure}
  {\includegraphics[fullpage]{frose}}
  {A caption of a \texttt{fullpage}
   object with \texttt{capPos=oddPage}
   ... for a long caption.}
  {fig:fullpage1}
```

The lower two pages in Figure 6 are similar, except capPos=evenPage and the object is set as FULLPAGE instead of fullpage.

The captions here (and throughout) are typeset in red to make them more visible in the examples, which are often reduced in size. The complete code for all examples is on CTAN (mirror.ctan.org/macros/latex/contrib/hvfloat/doc/examples).

5.2. *Twoside and twocolumn mode*

In contrast, in a twoside document in twocolumn mode, by default a caption appears *before* the fullpage or fullcolumn object, independent of an even or odd column or page. Figure 8 on page 35 shows the output of this example code:

```
\hvFloat[fullpage,capPos=inner]
  {figure}
  {\includegraphics[fullpage]{frose}}
  [A short caption for the LoF.]
  {A caption on the inner side of a
   twoside and twocolumn document
   (\texttt{capPos=inner}). This can
   be an even or odd page. And ...
   long caption.}
  {fig:full0}
```

The caption is in the inner column, which is the second one for an even (left) page and the first for an odd (right) page. For a twoside document it also makes sense to have the caption on the even (left) page in the second

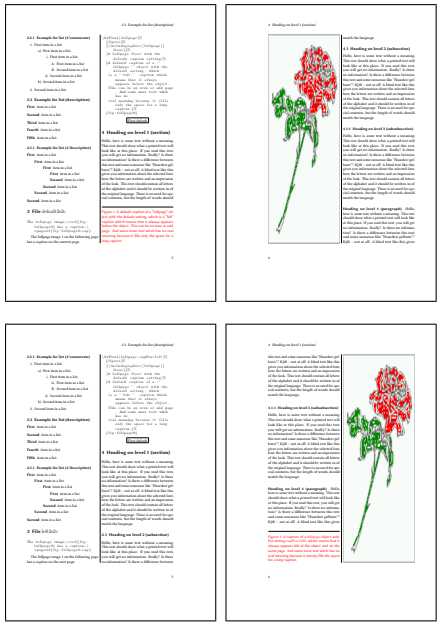


Figure 7. Twoside and twocolumn documents. Top: `capPos=before` (default); bottom: `capPos=left`. Pages 3–4 of example documents `default2s2c.tex` and `left2s2c.tex`, respectively.

margin and the object on the odd page (right) in the first margin. This can be achieved with the setting `capPos=inner`.

You can expect problems if you use the full column setting on a page which has full-width (double column) floats at the top. In such a case it is left to the user to modify the text structure to prevent such situations. You'll find many examples on CTAN (https://mirror.ctan.org/macros/latex/contrib/hvfloat/doc/examples) or in the documentation directory of your TEX distribution.

In twoside and twocolumn modes the keyword setting `capPos=left` is different from `capPos=before`. For this setting it makes no difference on what page and column the caption appears, it will always be *before* the object. For `capPos=left` the caption will *always* be left of the object *and* on the same page! Figure 7 shows this behavior.

6. Doublepage objects

A doublepage object makes sense only for twoside documents. Then the doublepage object can be placed on facing left–right pages and the caption perhaps on the right page or, in a case where the complete paper width is used, below the right part of the image, or, if need be, on the bottom of the preceding or following page. For example: suppose a doublepage object uses the complete paper area (`2\paperwidth` × `\paperheight`) on the (left–right) pages 80–81; then the caption can be printed at the bottom of page 79 or page 82 (see Figure 11 on page 37). It is also possible to print the caption *over* the right part of the object (image) on the bottom or rotated at the right (see Figure 12 on page 37).

With using the keyword `doublePage`, additional document text may appear below the doublepage object, that is, the object does not occupy the entire textheight. The other two possibilities `doublePAGE` (using the doublepage text area) and `doubleFULLPAGE` (use the doublepage paperwidth) have no additional document text on the two pages, but are still floating environments. We'll now describe all these in detail.

6.1. *Keyword* `doublePage`

This is the same as putting two different floats, one each at the top of the left and right pages. The package hvfloat clips an image which would be wider than the paperwidth. Otherwise it makes no sense to use a doublepage float.

For `doublePage` the object starts at the left top of the text area and ends on the right page, depending on its width. The inner margins of the two-sided document are ignored, but a binding correction (`bindCorr`) can be set and

will be taken into account. The caption will *always* be on the right page either beside, rotated or not, or below the object. For example, in Figure 13 on page 38 the caption is on the right (`capPos=right`) and rotated by 90° (`capAngle=90`). The left part of the image is on page 38, the right part on page 39 and the caption is on page 39. Incidentally, the internally-created labels described earlier were used to print this information. The label for the figure is `fig:dP`, and so the source for the previous sentence is:

```
The left part of the image is on
page~\pageref{fig:dP}, the right part on
page~\pageref{fig:dP-2} and the caption
is on page~\pageref{fig:dP-cap}.
```

A `doublePage` object allows for document text in addition to the two parts of the object. As for the caption, with `capWidth=n` and `capPos=right` the caption will be set to the right of the object with a natural width (from object to margin). This makes sense if the object is narrower than `\paperwidth` + `\textwidth`. Figure 13 on page 38 shows this, as well as (at a greatly reduced size) Figure 9 on the next page. The source for Figure 13 is as follows.

```
\hvFloat[doublePage,capWidth=n,
    capPos=right,capVPos=bottom]
  {figure}
  {\includegraphics[width=2\textwidth]
     {images/seiser}}
  [A short caption for the LoF]
  {A caption for a \texttt{doublePage}
   object, which will be placed on
   the right side of the right-hand
   part of the image. The image
   begins on the left edge of the
```

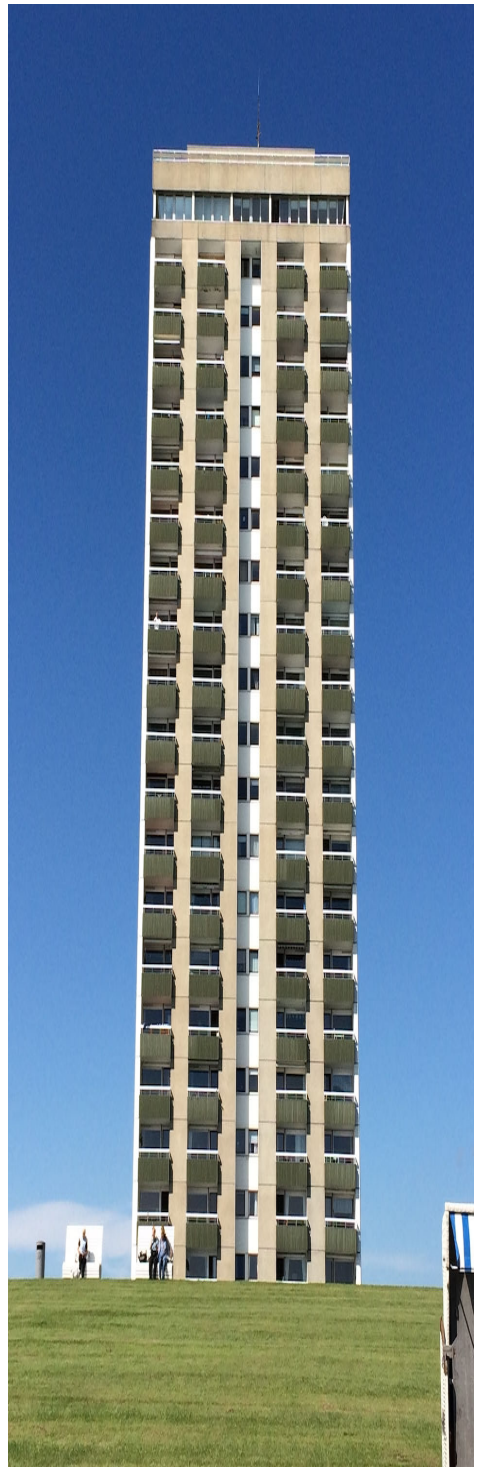


Figure 8. A caption on the inner side of a twoside and twocolumn document (`capPos=inner`). This can be an even or odd page. And some more text with no real meaning because it merely fills the space for a long caption.

```
paper [...] The photo was taken
[...]}{fig:dP}
```

In some cases it makes sense to have some whitespace, a binding correction, between the two split parts of the object. With the keyword `bindCorr` you can define a length value for the whitespace to be added both to the right of the left part and to the left of the right part (so the total whitespace added is $2 \times$ `bindCorr`).

The source for Figure 9 is the same as Figure 13, except for the addition of the setting `bindCorr=1cm` (and the label name).

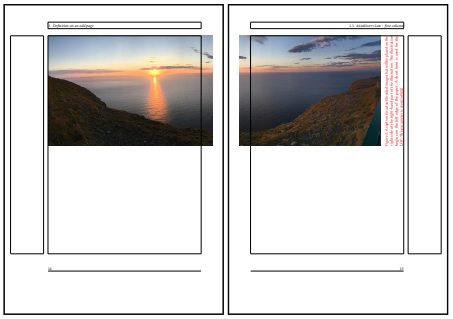


Figure 9. A `doublePage` object (the same image as Figure 13) with a binding correction of 1 cm. Pages 14–15 of example document `doublepage2s2c.tex`.

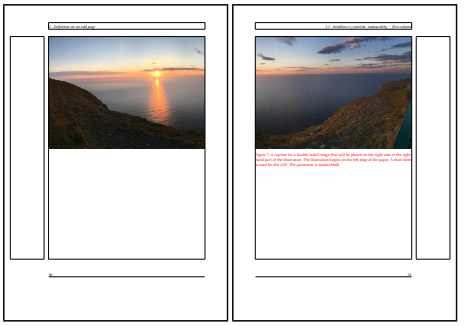


Figure 10. A `doublePAGE` image with `bindCorr` set to the inner margin. Pages 29–30 of example document `doublepage2s2c.tex`.

`bindCorr=inner`

Here is the source for Figure 10:

```
\hvFloat[doublePAGE,capWidth=n,
    bindCorr=inner]{figure}
  {\includegraphics[width=2\textwidth]
     {images/sonne-meer}}
  [A doublepage image with a
   caption ...]
  {A caption for a double-sided
   image ... The parameter is
   \texttt{doublePAGE}}
  {fig:doublePAGE3}
```

6.2. *Keyword* `doublePAGE`

A `doublePAGE` object appears alone on two facing pages, except for an optional caption. No additional document text will be printed on these two pages; this is the only difference between `doublePage` and `doublePAGE`. Figure 10 shows an example. The caption is below the object in the first column of the right (odd) page.

Figure 10 also shows an example of using the optional keyword `bindCorr` to specify whitespace between the parts of the split object. In this case, we use the inner margin for the binding correction to get the two images exactly fitting the textwidth. The value for the inner margin is computed internally:

6.3. *Keyword* `doubleFULLPAGE`

A floating object specified with the keyword `doubleFULLPAGE` always starts in the upper left corner of the left (even) page. The defined text area has no meaning, it will be completely ignored for these two floating pages. The caption can be printed before, after, below, or superimposed on the object.

Table 3 on page 31 lists the corresponding two optional keywords for the command `\includegraphics`, namely `doubleFULLPAGE` and `doubleFULLPAGEbindCorr`, with a preset of `keepaspectratio` to false. These keywords may make code more readable but have otherwise no special meaning

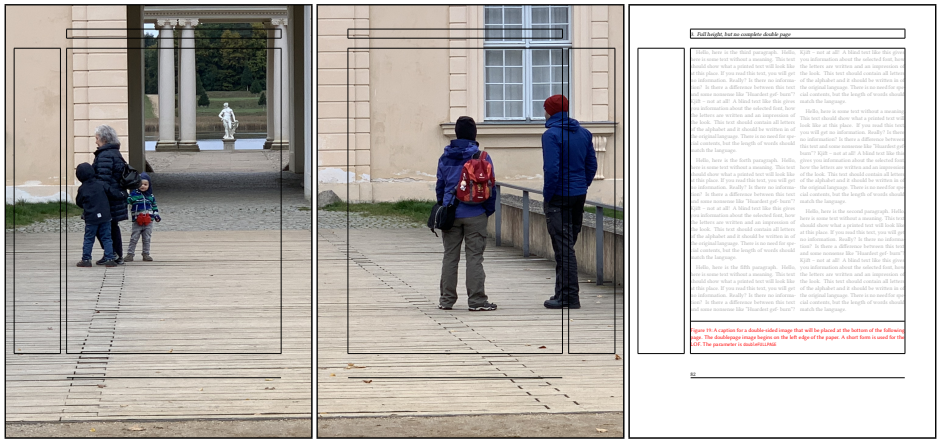


Figure 11. A doubleFULLPAGE object with capPos=after, so the caption is on the following page. Pages 80–82 of example document doublefullpage2s2c.tex.

Figure 12. A doubleFULLPAGE object with capPos=right, so the caption appears on the right page. Pages 72–73 of example document doublefullpage2s2c.tex.

for any objects other than images, e.g. a tabular or something else.

The object can have any width and height but it should be at least as wide as the given \paperwidth and not less than 50% of the \paperheight. For smaller objects, use one of the other two possibilities, doublePage or doublePAGE.

The caption can be superimposed on the object or, as an alternative, printed on the bottom of the page preceding or following the doublepage (left–right) object. For a two-column document there exists the keyword twoColCaption which can be used to span both columns. This will only work for twocolumn documents which define the column mode using \twocolumn, such as the TUGboat document class. The multicol package is *not* supported.

Figure 12 shows two pages with an image spread across the double page which is small enough to get a rotated caption on the right of the page which, for our demonstration, is printed in red as usual. The page layout is also printed as frames, which makes it easier to understand and choose values for the full page mode. These frames are shown by loading the package showframe.

The code for Figure 12 is:

```
\hvFloat[doubleFULLPAGE,capPos=right]
  {figure}
  {\includegraphics[height=\paperheight]
     {images/rheinsberg}}
  [A doublepage image ...]
  {A caption for a double-sided image ...
   The parameter is
   \texttt{doubleFULLPAGE}}
  {fig:doubleFULLPAGEOn}
```

If the image has nearly the same ratio as the current `\paperwidth` / `\paperheight`, then a caption can reasonably appear at the bottom of the following page. This is specified with `capPos=after`; Figure 11 on the previous page shows the result. Similarly, the setting `capPos=before` would put the caption on the preceding page.

Here is the code for Figure 11, specifying the option `doubleFULLPAGE` option to both `\hvFloat` and `\includegraphics`:

```
\hvFloat[doubleFULLPAGE,capPos=after,
     twoColumnCaption]{figure}
  {\includegraphics[doubleFULLPAGE]
      {rheinsberg}}
  {A caption for a double-sided
   image ... The parameter is
   \texttt{doubleFULLPAGE}}
  {fig:doubleFULLPAGE02ndnn}
```

7. Subfloats and multifloats

A floating environment can have any content except another floating environment. The only requirement for the content is that it must be smaller than one page spread. The content itself can be any combination of text, equations, tabulars, and/or images. We call it a *subfloat* if the content has *one* main caption and several subcaptions for any object. We call it a *multifloat* if the content has no main caption of its own, but the objects have their own captions.

Table 4 on the facing page gives the two keywords, `subFloat` and `multiFloat`, which introduce such special content. They can be placed as a default floating environment, full column, full page, or full doublepage.

The syntax for the macro which defines such sub- or multifloats is somewhat complex. Only the keyword defines whether the float is a multifloat or subfloat; the syntax of the macro shows no difference. With the optional argument `vFill` the objects in a column (two column) or a page (one column) are stretched over the given height `\textheight`. The default is no stretching so that extra whitespace appears at the bottom of the column/page.

Figure 13. A caption for a `doublePage` object, which will be placed on the right side of the right-hand part of the image. The image begins on the left edge of the paper. A short form can be used for the LoF. The photo was taken in the Italian Alps at the Alpe di Siusi (Seiser Alm).

Table 4. Keywords `subFloat` and `multiFloat` for multiple objects in a float.

Name	*Description*
`subFloat`	For multiple objects with one main caption and several subcaptions.
`multiFloat`	For multiple objects, each with its own caption.

7.1. Subfloats

A subfloat page can have only one type of object which will have one main caption and individual subcaptions. (For completeness: If you define no subcaption then it does not matter what kind of object we have.) The syntax for subfloats and multifloats is similar, but some arguments are ignored for a subfloat, so can be left empty.

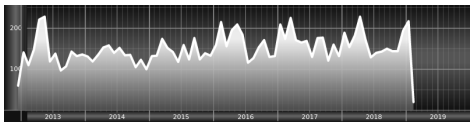
```
\hvFloat [subFloat,...]
  +{main float type}{} [short caption]
      {long caption}{label}
  +{}{sub floating object} [short caption]
      {long caption}{label}
  ⋮ ...
  +{}{sub floating object} [short caption]
      {long caption}{label}
```

The first line defines only the floating type and the main caption, the object entry is ignored! All additional lines will have the same float type; this is why the float type entry is ignored.

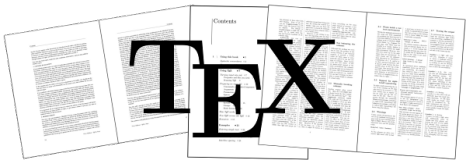
The + symbol defines an additional object which will be part of the same floating environment. It's up to the user to be sure that one page or one column can hold all defined objects.

The code for Figure 14 on the next page, which comprises the subfigures 14a to 14e, is as follows:

```
\hvFloat[subFloat,vFill,fullpage,
```

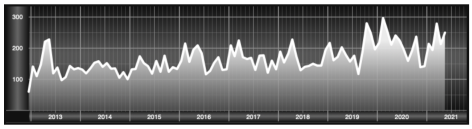


(a) Subcaption A of a fullpage subobject.



(b) Subcaption B of a fullpage subobject, a little longer for no particular reason.

(c) Subcaption C of a fullpage subobject.



(d) Subcaption D of a fullpage subobject.

(e) The last subcaption E of a fullpage subfloat object, which has subcaptions 14a–14e, and the main caption is beside (to the right of) this full column object.

```
    capPos=after]
  +{figure}{}[Short caption of the
    subfloat]{The main caption of a
    fullpage subfloat, which appears
    in the left or right column. This
    can be an even or odd page. The
    \texttt{vFill} option is set,
    so vertical space is distributed
    between the subobjects.}{sub:demo}
  +{}{\includegraphics[columnWidth]
   {CTAN}}[Short caption A]{Subcaption
    A of a fullpage subobject.}{sub:demo0}
  +{}{\includegraphics[columnWidth]
   {CTAN1}}{Subcaption B of a fullpage
    subobject, a little longer for no
    particular reason.}{sub:demo1}
  +{}{\includegraphics[columnWidth]
   {CTAN2}}{Subcaption C of a fullpage
    subobject.}{sub:demo2}
  +{}{\includegraphics[columnWidth]
   {CTAN3}}{Subcaption D of a
    fullpage subobject.}{sub:demo3}
  +{}{\includegraphics[trim=0 1.5cm
    0 5mm,clip,columnWidth]{TUGboat}}
   {The last subcaption E of a
    fullpage subfloat object, which
    has subcaptions \ref{sub:demo0}%
    --\ref{sub:demo5}, and the main
    caption is beside (to the right
    of) this full column object.}
    {sub:demo5}
```

The keyword `subFloat` defines the following images or tabulars as subfloats. The keyword `figure` in the second line of the code defines the main type of the floating environment; all subobjects must be of the same type. This is the reason why all following arguments are empty: `+{}{...`.

The package subcaption is loaded by default and is usually activated with `\captionsetup[sub][singlelinecheck]`.

Figure 14. The main caption of a fullpage subfloat, which appears in the left or right column. This can be an even or odd page. The `vFill` option is set, so vertical space is distributed between the subobjects.

The main label of the subfloat is sub:demo, which points to the object column on page 40. In this case the internal label sub:demo-cap points to the same page 40, because object and caption are in different columns but on the same page. Both refer to the same object: \ref{sub:demo} → 14 and \ref{sub:demo-cap} → 14.

7.2. Multifloats

With a multiFloat object, no main caption is given. Every object gets its own caption, which is the reason that figures, tabulars, etc., can be mixed. All individual captions are listed before or after the full column/page, at the bottom of the column/page (see example on the following page).

```
\hvFloat [multiFloat,...]
  +{float type}{floating object}
       [short caption] {long caption}
       {label}
  +{float type}{floating object}
       [short caption] {long caption}
       {label}
  ⋮ ...
  +{float type}{floating object}
       [short caption] {long caption}
       {label}
```

The + symbol defines an additional object which will be part of the same floating environment. For a multifloat object all parameters are valid. It's up to the user to be sure that one page or one column can hold all defined objects.

The captions of Figures 15–18 and of Tables 5 and 6 are on page 41, and all objects also appear on the same page. All of these figures and tables are part of the same multifloat. Here is the code of the multifloat example:

```
\captionsetup{singlelinecheck=false}
\hvFloat[multiFloat,vFill,
        fullpage,capPos=before]
+{figure}
{\includegraphics[columnWidth]{dove}}
 [Short caption A]
 {Caption A of a fullpage multifloat
  object, which follows in the left or
  right column. This can be an even or
  odd page. And some more text with no
  real meaning because it merely fills
  the space for a long caption.}
 {img:demo}
+{table}{\begin{tabular}
  {@{}lrcp{2cm}@{}}\hline
  Left & Right & Centered & Parbox\\
    \hline
  L    & R     & C        & P\\
  left & right & center   & Text
                 with line breaks\\
  L    & R     & C        & P\\
  left & right & center   & Text
                 with line breaks\\
  \multicolumn{4}{c}{Centered
    multicolumn over all columns}\\
      \hline
        \end{tabular}}
 [Short example caption B1]
 {Caption B of a fullpage object, a
   tabular in this case.}{tab:demo}
+{figure}
 {\includegraphics[columnWidth]{CTAN1}}
 {Caption C of a fullpage object.}
 {img:demo1}
+{figure}
```

Figure 15. Caption A of a fullpage multifloat object, which follows in the left or right column. This can be an even or odd page. And some more text with no real meaning because it merely fills the space for a long caption.

Table 5. Caption B of a fullpage object, a tabular in this case.

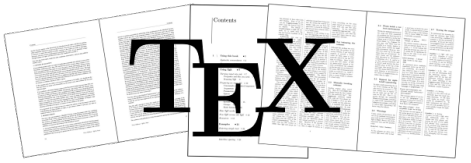
Figure 16. Caption C of a fullpage object.

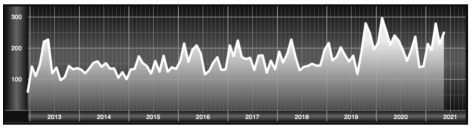
Figure 17. Caption D of a fullpage object.

Figure 18. Caption E of a fullpage object.

Table 6. Caption B2 of a fullpage object, another tabular repeating Table 5.

Left	Right	Centered	Parbox
L	R	C	P
left	right	center	Text with possible line breaks
L	R	C	P
left	right	center	Text with possible line breaks
Centered multicolumn over all columns			





Left	Right	Centered	Parbox
L	R	C	P
left	right	center	Text with line breaks
L	R	C	P
left	right	center	Text with line breaks
Centered multicolumn over all columns			

```
 {\includegraphics[columnWidth]{CTAN2}}
 {Caption D of a fullpage object.}
 {img:demo2}
+{figure}
 {\includegraphics[columnWidth]{CTAN3}}
 {Caption E of a fullpage object.}
 {img:demo3}
+{table}{\begin{tabular}
  {@{}lrcp{2cm}@{}}\hline
  Left & Right & Centered & Parbox\\
    \hline
  L    & R     & C        & P\\
  left & right & center   & Text
                  with line breaks\\
  L    & R     & C        & P\\
  left & right & center   & Text
                  with line breaks\\
  \multicolumn{4}{c}{Centered
    multicolumn over all columns}\\\hline
       \end{tabular}}
 [Short example caption B2]
 {Caption B2 of a fullpage object,
  another tabular repeating
  Table~\ref{tab:demo}.}
 {tab:demo2}
```

8. Splitting tables across two pages

By default a table can only be split in the vertical direction, as a so-called longtable. Large tables can be rotated on a page (see Table 2 on page 28), but splitting it automatically in the horizontal direction is currently (TeXLive 2022) not supported by core LaTeX.

However, saving a table without page breaks into a box is no problem and such a box can be handled like an image, which is also like a box. The only problem is that the table must be split horizontally between two columns, as a split column may likely be unreadable.

The package hvfloat provides the box \hvOBox for public use. We can save a table into this box:

```
\savebox\hvOBox{%
  \begin{tabular}{l @{} *{18}r}
    ... the table ...
```

```
\end{tabular}}
```

and then use it in the same way as a doublepage image, with the table split in two pieces. If the split occurs at an unfavorable point in the table, e.g. in the middle of a column, then insert some horizontal space between the two columns with `@{\hspace{...}}`. For example (the output is shown in Table 7):

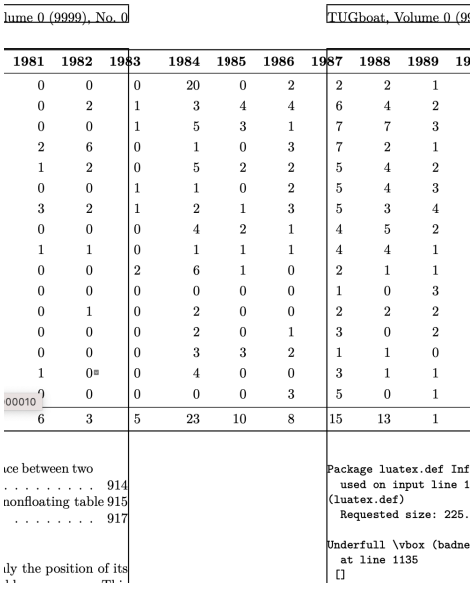


Figure 19. The table column *1985* appears between the two pages and would not be readable.

```
\begin{tabular}
  {lll@{\hspace{1cm}}ll}\hline
1 & 2 & 3 & 4 & 5 \\
1 & 2 & 3 & 4 & 5 \\
1 & 2 & 3 & 4 & 5 \\
1 & 2 & 3 & 4 & 5 \\\hline
\end{tabular}
```

Figure 19 shows how the table looks in the middle of the doublepage (the text shown at the bottom of the page is just filler). The column with *1985* will be cut and not readable. There are two solutions to split the table at a better position: insert some space *before* this column, or use the `bindCorr` keyword to insert a binding correction space. For Table 8 on the following page both possibilities are used. Inserting more space:

Table 7. Adding space between two columns

1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5

```
\begin{tabular}
  {l @{} *{13}r @{\quad}*8r}
```

and using 8 mm for the binding correction (shown below) which was found by trial and error.

The code for the split table on a double page is:

```
\hvFloat[doublePage,capWidth=n,
   capPos=right,capVPos=bottom,
   useOBox,% use the defined box
   bindCorr=8mm]
  {table}
  {}% no need for an object
  [A doublepage tabular.]
  {A caption for a doublePage tabular that
   will be placed on the right side of the
   right-hand part of the tabular. The
   table begins on the left edge of the
   text area of the left page. The
   additional space between the columns
   1984 and 1985 is \texttt{\textbackslash
   quad}, which is the same as 1\,em. The
   binding correction is set to 8\,mm,
   which gives additional whitespace of
   16\,mm.}{tab:dP}
```

and the output is Table 8 on pages 44 and 45. It depends on the way the document is printed whether more or less space between the two pages makes sense.

	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982
Line No 1	1	3	1	1	1	0	1	1	0	0	0
Line No 2	1	1	3	1	0	0	0	0	0	0	2
Line No 3	2	1	2	1	0	0	0	0	0	0	0
Line No 4	1	0	5	1	2	0	0	0	0	2	6
Line No 6	2	1	1	0	0	0	0	0	0	1	2
Line No 5	0	0	4	2	1	2	2	1	0	0	0
Line No 8	0	1	1	0	0	0	1	1	0	3	2
Line No 9	0	0	0	0	0	1	2	1	0	0	0
Line No10	0	1	3	0	1	0	1	0	0	1	1
Line No11	0	2	2	1	1	0	1	0	0	0	0
Line No12	2	0	2	4	1	0	4	0	0	0	0
xyz	2	3	0	0	0	0	0	0	0	0	1
Line No13	0	1	0	0	1	0	3	0	0	0	0
Line No14	0	1	0	0	0	0	0	0	0	0	0
Line No15	0	0	0	0	0	0	0	0	0	1	0
Line No16	0	0	0	0	0	1	0	0	0	0	0
Some numbers	2	6	13	8	4	3	5	4	0	6	3

9. Todo list

The macro `\hvFloat` only checks the position of its definition if it is defined on an odd or even page. This is done with the help of the macro `\checkoddpage` from the package ifoddpage. Together with the internal LaTeX macro `\if@firstcolumn` it knows exactly the position of its definition in the source of the document: left or right page, first or second column. But it doesn't know if the current page is completely empty, which is the case if `\hvFloat` is the first command on a new page. If this is also an even page, then a doublepage object can be placed immediately. But the current code always uses the *next* even–odd page combination. In a future release there should be a test like `\if@newpage`.

More checks for the correct use of the parameters would be useful. For example: if one uses the keyword `doubleFULLPAGE` with an object which is narrower than `\textwidth`, then the output will be rubbish.

The optional argument `wide` as shown in Figure 21 on page 46 works only in oneside mode if you also use twocolumn mode (see Figure 20 on the facing page). For twoside mode we have different margins for a possible wide float in the first or the second column; this is not recognized by hvfloat. However, if you need wide floats in a twoside and twocolumn mode you can move the macro `\hvFloat` to places in the source where the output is always in the outer column, which uses the marginpar width. Using the argument `nonFloat`, as shown in Figure 20 on the next

1983	1984	1985	1986	1987	1988	1989	1990	1991	1992
0	20	0	2	2	2	1	2	1	0
1	3	4	4	6	4	2	2	1	0
1	5	3	1	7	7	3	2	1	0
0	1	0	3	7	2	1	2	1	0
0	5	2	2	5	4	2	2	1	0
1	1	0	2	5	4	3	2	1	0
1	2	1	3	5	3	4	2	1	0
0	4	2	1	4	5	2	2	1	0
0	1	1	1	4	4	1	2	1	0
2	6	1	0	2	1	1	2	6	0
0	0	0	0	1	0	3	2	6	0
0	2	0	0	2	2	2	2	6	0
0	2	0	1	3	0	2	2	6	0
0	3	3	2	1	1	0	2	6	0
0	4	0	0	3	1	1	2	6	0
0	0	0	3	5	0	1	2	6	1
5	23	10	8	15	13	1	32	51	1

Table 8. A caption for a doublePage tabular that will be placed on the right side of the right-hand part of the tabular. The table begins on the left edge of the text area of the left page. The additional space between the columns 1984 and 1985 is `\quad`, which is the same as 1 em. The binding correction is set to 8 mm, which gives additional whitespace of 16 mm.

page, the float appears exactly at the place of the definition.

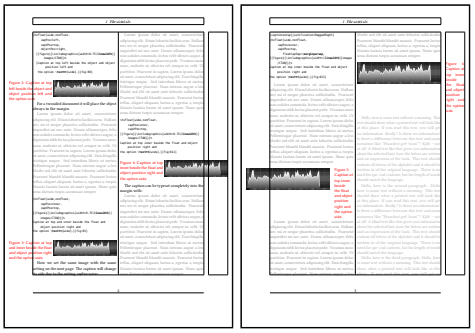


Figure 20. Pages 2–3 of example document `wide1s2c.tex`, oneside with twocolumn and the `wide` option.

In some cases the option `useOBox` for a predefined savebox `\hvOBox` does not work. One can use instead `{\usebox\hvOBox}` as the argument for the object, which has the same effect. However, the box `\hvOBox` must have valid contents, and be set before it is used.

10. Conclusion

The package hvfloat should work with all kinds of documents, oneside in one- or twocolumn mode, twoside in one- or twocolumn mode. It is much easier to place doublepage objects in a onecolumn document than a twocolumn document. Internally, LaTeX puts two single pages together to one page with two columns. Only the optional header and footer are printed across these "two" pages.

The package hvfloat makes intensive use of the macro `\afterpage` Carlisle and The LaTeX Team 2014. If one defines a doublepage object in the first column of a left (even) page, `\hvFLoat` needs *three* nested `\afterpage` commands, one for each column, to let an object or a caption start on the next left (even) page. Until LaTeX reaches this page for the object/caption, nearly two pages have to be filled with text or other objects which are defined after the macro `\hvFloat`. Especially in twocolumn mode you can expect problems, if you have too little text, images, tables or other simple objects to fill up these two pages until the doublepage object will be set. Such problems can only be solved by adding some text or moving the macro `\hvFloat` to another column of the document.

Just as with the standard floating environments figure and table, it is left to the user to ensure that the contents of the environment (object and caption) fit the page. If an object is wider than $2 \times$ `\paperwidth` or higher than `\paperheight` it cannot be placed on a doublepage and the output may be useless.

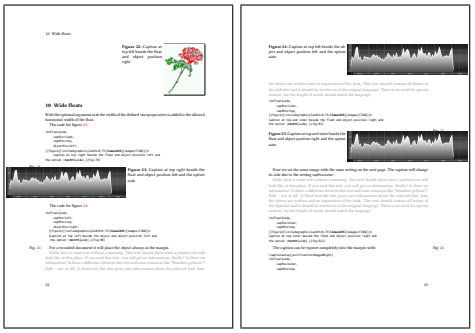


Figure 21. Pages 22–23 of the package documentation, showing examples using optional argument `wide` to use the margin space.

Before using a doublepage for an object, one should test if it might be sufficient to use the margin for additional space. `\hvFLoat` knows the optional argument `wide` which allows using the space of `\marginparwidth`. The caption can be placed in the usual way, above/below or left/right relative to the object. The use of the inner/outer position for twoside documents is also possible.

Figure 21 shows some examples of using the margin for a onecolumn document with the following use of `\hvFloat`.

```
\hvFloat[wide,capPos=inner,capVPos=top]
 {figure}
 {\includegraphics[width=0.75\linewidth]
   {images/CTAN}}
 {Caption at top inner beside the float ...
  and the option \texttt{wide}.}{fig:wide}
```

The list of figures and list of tables are not affected by package hvfloat and should work as usual. For example, here is the list of tables for this article:

Another feature is that simple and nonfloating objects can be placed by the environment hvFLoatEnv, which has only one optional argument, giving the horizontal width. For the caption one has to use the defined macro `\captionof{`*type*`}{...}` or the (usually internal) macro `\tabcaption{...}` mentioned on page 26:

List of Tables

```
\begin{hvFloatEnv}[0.5\columnwidth]
\centering
\captionof{table}{A short nonfloating table.}\label{tab:nonfloat}
\begin{tabular}{@{} l c r @{}}\hline
left & center & right \\
L    & C      &  R    \\\hline
\end{tabular}
\end{hvFloatEnv}
```

Table 9. A short nonfloating table.

left	center	right
L	C	R

But pay attention to references if floating and non-floating environments are mixed on one page; they can point to wrong numbers. Moving the floating environment to another place in the document is one workaround for such a problem. Alternatively, using only floating environments is preferred, if your document is mainly text, with only some figures and/or tables.

References

Carlisle, David, and The LaTeX Team. 2014. "The afterpage package." Execute command after the next page break, October 28, 2014. Accessed May 28, 2021. https://ctan.org/pkg/afterpage.

Mittelbach, F., M. Goossens, J. Braams, D. Carlisle, and C. Rowley. 2004. *The LaTeX Companion.* 2nd. Pearson Education. ISBN: 9780133387643.

Scharrer, Martin. 2016. "The ifoddpage package." Determine if the current page is odd or even, April 23, 2016. Accessed May 29, 2021. https://ctan.org/pkg/ifoddpage.

———. 2020. "The adjustbox package." Graphics package-alike macros for "general" boxes, August 19, 2020. Accessed May 29, 2021. https://ctan.org/pkg/adjustbox.

Sommerfeldt, Axel. 2020. "The caption package." Customising captions in floating environments, October 26, 2020. Accessed May 28, 2021. https://ctan.org/pkg/caption.

Tolušis, Sigitas. 2017. "The stfloats package." Commands to control the presentation of floats, March 27, 2017. Accessed May 28, 2021. https://ctan.org/pkg/stfloats.

Voß, Herbert. 2021. "The hvfloat package." Rotating and controlling float captions and objects, June 29, 2021. Accessed June 29, 2021. https://ctan.org/pkg/hvfloat.

Herbert Voß
Wasgenstrasse 21
14129 Berlin, Germany
Herbert.Voss@fu-berlin.de

Preventing tofu with pdfTeX and Unicode engines

Frank Mittelbach

Abstract Discussion of input encodings vs. font encodings, missing characters, Unicode, and TeX history.

Sommario Discussione a proposito di codifiche di immissione vs. codifiche dei font, caratteri mancanti, Unicode e la soria di TeX.

1. With tofu through the years

Tofu is not just an essential ingredient for many Asian dishes, it is also the nickname for the little squares produced by many browsers when they are asked to render a character for which they do not have a glyph available.

Especially in the early days of the World Wide Web, websites in foreign languages (from the perspective of your computer) got often littered with such squares, making text comprehension quite difficult if not impossible in some cases. So instead of getting

> ¿But aren't Kafka's Schloß & Æsop's Œuvres often naïve vis-à-vis the dæmonic *phœnix's official rôle* in fluffy soufflés?

you might have seen something like

> □But aren't Kafka's Schlo□ □ □sop's □uvres often na□ve vis-□-vis the d□monic *ph□nix's official r□le* in fluffy souffl□s?

Over the years the situation with browsers improved (partly because using inferior fonts was deemed acceptable as long as they could render the needed glyphs), but even nowadays you may find tofu-littered sites, or perhaps worse, those where your browser thinks it can show you the glyphs but renders the wrong ones.

While with browsers you may accept a certain imperfection in the rendering, tofu in printed material is quite unacceptable. Typesetting systems should always use the correct glyphs or at least tell you very explicitly if they are unable to do so for some reason, to allow you to apply some corrective actions. In the remainder of this article we will discuss how TeX and in particular LaTeX is doing in this respect and what a user can or must do to avoid such a capital blunder.

1.1. Early vegetarian dishes with TeX

In the early days of TeX the use of fonts was easy because you could use any font you wanted as long as it was called Computer Modern.

In other words there was essentially only one set of fonts available for use with TEX and the glyphs it contained and how to address them was described in TUGboat Knuth, 1986. Furthermore, all fonts only contained 128 glyphs, i.e., essentially the base Latin characters, a few accents to construct diacritical characters using the `\accent` primitive and a few other symbols such as †, $ and so forth to be accessed through command names.

Thus, once you learned the construction methods and memorized the control sequences for accessing the existing symbols you could be sure that the characters you used would faithfully appear in the printed result. Of course, part of the reason for this was the limited glyph set; already any Latin-based language other than English posed serious issues, namely that necessary glyphs were missing entirely, or only available as constructed characters (whenever accents where involved) — which prevented TEX from applying hyphenation.

So as TEX got more popular outside the English-speaking world there was considerable pressure on Don Knuth (largely by European users, the author among them) to extend TEX so that it could better handle languages with larger character sets. At the 1989 TEX conference in Stanford we finally managed to convince Don to reopen (in a limited way) TEX development and produce TEX 3. This version of TEX was then able to deal with more than one language within a document (e.g., use multiple hyphenation patterns) and support 8-bit input and output (that is, 256 characters in a font).

While this enabled the use of different input code pages for different character sets, as was standard in those days, and also solved the problem of hyphenating words containing accented characters (by using fonts with precomposed glyphs), it also posed new challenges.

Depending on the active code page when writing a document, a given keyboard character might be associated with a different number (between 0 and 255) and that number had to be mapped to the right slot in a font to produce the glyph that was originally intended. So the days of input number equals font glyph position were definitely over, and the TEX world had to come up with a more elaborate scheme to translate one into the other to avoid missing or wrong characters in the output.

1.2. The LaTeX 2ε solution

For LaTeX the solution came in the form of the New Font Selection Scheme Mittelbach and Schöpf, 1990, and in particular with the packages `inputenc` (for managing input in different code pages and mapping it to a standard internal representation) and `fontenc` (for translating this internal representation to the correct glyph positions in different fonts).

1.2.1. Introducing font encodings

LaTeX classified the font encodings and gave them names such as `OT1`, `T1`, `TS1`, `T2A`, `T2B`, etc. Each such font encoding defined which glyphs are in a font using that encoding and to which character code (again, 0–255) each glyph was assigned in the font. Thus, if you had two different fonts with the same encoding you could exchange one for the other and still be one hundred percent sure[1] that your document get typeset correctly, with no missing or incorrect glyphs in the output.

1. Well, more like 99% since sometimes fonts claimed to be in one encoding but didn't faithfully follow its specification, e.g., didn't provide all glyphs or sometimes even placed wrong glyphs at some slots.

In practice only a small number of font encodings ever got used and new fonts usually were made available in these "popular" encodings by providing the necessary font re-encodings through the virtual font mechanism, or through re-encodings done by device drivers (such as `dvips`) or directly in the engine (in the case of `pdftex`).

As an overall result, life for LaTeX users was again fairly easy after 1994 and remained this way well into this century, because by simply specifying which font encoding to use, documents would normally be typeset without defects, regardless of the font family that got used. Further, due to the fact that for users writing in Latin-based languages essentially every interesting font available was provided in the `T1` encoding, it was also clear which glyphs were available and those were available almost universally.

1.2.2. Pitfalls with missing input encodings

There was still the need to specify the input encoding — at least if one wanted to input accented characters directly from the keyboard instead of using TeX constructs like `\"a`. One problem in this respect was that, depending on the language you were writing in, it sometimes worked even without specifying the input encoding. This was possible because the `T1` font encoding was nearly identical to the quite common `latin1` input encoding.[2] Years later, omitting the input encoding declaration even when it worked initially finally backfired: once LaTeX moved on to make UTF-8 the default encoding, documents stored in legacy encodings failed if they didn't contain an input declaration.[3]

1.2.3. Pitfalls with the TS1 encoding

When 8-bit fonts became more common, the TeX community defined two font encodings during a conference at Cork in 1990. The first is `T1`, which holds common Latin text glyphs that play a role in hyphenation and therefore have to be present in the same font when seen by TeX. The second is `TS1`, which contains other symbols, such as oldstyle numerals or currency symbols; these can be fetched from a secondary font without harm to the hyphenation algorithm, because they do not appear as part of words to be hyphenated.

On the whole, the glyphs in the `T1` encoding were well-chosen and it is usually possible to arrange any commercial or free font to be presented in this encoding to TeX.[4] As a result, substituting `T1`-encoded fonts means that you can be fairly sure that there will be no tofu in your output afterwards.

Unfortunately, this is not at all true for the `TS1` encoding. Here the community made a big mistake by going overboard in adding several "supposedly" useful glyphs to the encoding that could be produced in theory (and for Computer Modern and similar TeX fonts were in fact produced), but that simply did not exist in any font that had its origin outside the TeX world.

2. For example, with French texts it worked throughout. However, with German only the "umlauts" worked, but the sharp s "ß" generated a different character.
3. The remedy for such old documents is to either add the missing declaration or re-encode the old source and store it in UTF-8.
4. There are a few exceptions where some seldom used glyphs are missing, e.g., `\textpertenthousand` or `\textcompwordmark`.

As a result, to use such glyphs from the TS1 encoding meant that you had to stay with a very limited number of font families. Alternatively, you had to be very careful not to use any of the problematic symbols to avoid tofu.

To ease this situation, the TS1 encoding was subdivided into five sub-encodings and a LaTeX interface was established to identify that a font family with a certain NFSS name belonged to one of the sub-encodings. This way LaTeX was enabled to make "reasonable" adjustments when a requested symbol was not available in the current font, either by substituting it from a different font or by giving you an error message that the symbol is not there — not perfect but better than tofu in the end. This was implemented in the `textcomp` package which provided the LaTeX commands to access the symbols from TS1.

In one of the recent LaTeX releases the code from `textcomp` was moved to the LaTeX format, so that these extra symbols are now available out of the box without the need to load an additional package. At the same time, the classification of fonts into TS1 sub-encodings was reworked. We now support nine sub-encodings and the LaTeX format contains close to 200 declarations that sort the commonly available font families into the right sub-encodings. Thus these days the situation is fairly well under control again — at least with pdfTeX.

2. Unicode

One of the goals of Unicode is to uniquely identify each and every character used in different languages and scripts around the world, thereby avoiding some of the possible translation problems that occurred because a text was written under the assumption of one (8-bit) encoding, but interpreted later under a different encoding.

While this was a huge step forward for correctly interpreting any source document (because it eliminated all of the the different input encodings — all is now Unicode), it unfortunately reintroduced a new helping of tofu through the back door.

The reason is simple: with Unicode as the means to reliably address a glyph to be typeset in a font, such a font has to contain glyphs for *all* characters available in Unicode, because TeX just takes the Unicode number and tells the current font "typeset this glyph". While this is in theory possible in the TrueType or OpenType font formats (using font collections), there is no single font (or collection) that offers anything close to this.[5] LaTeX has no way to identify if glyphs are missing, because the typesetting of paragraph text is a very low-level process in TeX and in contrast to the pdfTeX engine where LaTeX can reliably assume that a font in T1 encoding implements the whole encoding, in Unicode engines all fonts are in the TU encoding (the whole of Unicode), which no font provides.

In theory it would have been possible to devise sub-encodings of TU and assign each and every font to the appropriate sub-encoding, as was done with TS1, but in practice this would be a hopeless undertaking, because each and every font implements its own set of glyphs, so no useful classification is possible.

5. The font I know that comes closest is Code2000 Kass, 2000, which provides around 90K characters in its latest incarnation — but even that is only a fraction of the Unicode universe (over 140K characters). Google's Noto project Google Fonts, 2021, which stands for "**no to**fu", was established to develop fonts for typesetting text in any of the world's languages and scripts. It currently has almost 64K characters, which are split across nearly two hundred font families, e.g., if you want to typeset in Latin you can use Noto Sans, but for Japanese you need Noto Sans Japanese and so forth.

Thus when you typeset in XETEX or LuaTEX and you request using a certain font family with something like

```
\setmainfont{Alegreya}
```

you just have to hope your chosen family contains glyphs for all characters that you intend to use in your document; if not, you will end up with tofu.

To give you some figures: Latin Modern Roman (the default font in LATEX on Unicode engines) implements 794 characters, the ParaType font used for this article 717, the Optima font on the Mac just 264, the free Alegreya font 1251 and Noto Serif 2840. Regardless, there can be no guarantee that the characters contained in your document are covered.

2.1. Letting TEX tell you about your tofu

The TEX program offers one tracing parameter, called `\tracinglostchars`, that, if set to a positive value, reports missing glyphs (a.k.a. tofu) in the log file, e.g.,

```
Missing character: There is no È (U+00C8) in font cmr10!
```

Interestingly enough, this information is not even given by default, but only when you explicitly ask for it — obviously, Don Knuth did not foresee that TEX is used with fonts other than those carefully crafted for TEX and containing all the characters you may want.

Recently all TEX engines were enhanced to make tofu reporting a little better: you can now set this parameter to 2, after which it reports its finding also on the terminal (the new default value in LATEX), or you can set it to 3, after which it will throw an error rather than the easy-to-miss warning. With Unicode engines we strongly recommend to always set

```
\tracinglostchars=3
```

in the preamble of your document — it is much better to get errors when writing your documents instead of getting reports by others about tofu in your published work. As explained before, when typesetting with pdfTEX there is little danger of ending up with tofu, so there it is less important to change the parameter, though it obviously doesn't hurt.

3. Typesetting Unicode font tables

When I worked on the font chapter for the new edition of *The LATEX Companion*, third edition Mittelbach and Fischer, to appear in 2022, I wanted to produce glyph tables for various fonts to examine which characters they encode and how they looked. To my surprise I could not find any TEX tool to do this for me. There is, of course, the old `nfssfont` which I had adapted from work by Don Knuth, but that is of no help with Unicode fonts as it can only display tables of the first 256 characters, i.e., 8-bit fonts. So during my last stay at Bachotek (before the pandemic) I sketched out some code, the result of which is now available as the `unicodefonttable` package (see companion article).

References

Google Fonts (2021), *Noto: A typeface for the world*, https://fonts.google.com/noto.

Kass, James (2000), *Code2000*, Font resource implementing much of Unicode., https://en.wikipedia.org/wiki/Code2000.

Knuth, Donald E. (1979), "The Letter S," 2, 3, Also published as Knuth, 1980., pp. 114-112.

— (1980), *The Letter S*, tech. rep. STAN-CS-80-795, Also published as Knuth, 1979.

— (1986), *The TEXbook*, Computers and Typesetting, vol. A, pp. ix + 483, isbn: 0-201-13447-0.

Mittelbach, Frank and Ulrike Fischer (to appear in 2022), *The LATEX Companion*, 3rd ed., Pearson Education, Boston, MA, USA.

Mittelbach, Frank and Rainer Schöpf (1990), "The new font family selection — User interface to standard LATEX," *TUGboat*, 11, 1 (27 Apr. 1990), pp. 91-97, issn: 0896-3207, https://tug.org/TUGboat/tb11-1/tb27mitt.pdf.

Frank Mittelbach
Mainz, Germany
https://www.latex-project.org
https://ctan.org/pkg/unicodefonttable

The unicodefonttable package*

Frank Mittelbach

Abstract A package for typesetting font tables for larger fonts, e.g., TrueType or OpenType Unicode fonts. To produce a one-off table, a standalone version is available as well.

Sommario Un pacchetto per comporre tabelle di caratteri per font di grosse dimensioni, per esempio font TrueType o OpenType a codifica Unicode. È disponibile anche una versione autonoma per produrre tabelle *una tantum*.

1. Introduction

When I started to write a new chapter for the third edition of The LaTeXCompanion on modern fonts available for different LaTeX engines, I was a bit surprised that I couldn't find a way to easily typeset tables showing the glyphs available in TrueType or OpenType fonts. The nfssfont package available with LaTeX only supports fonts from the 8-bit world, but modern fonts that can be used with XeTeX or LuaTeX can contain thousands of glyphs and having a method to display what is available in them was important for me.

I therefore set out to write my own little package and what started as an afternoon exercise ended up being this package, offering plenty of bells and whistles for typesetting such font tables.

As there can be many glyphs in such fonts a tabular representation of them might run for several pages, so the package internally uses the longtable package to handle that. In most cases the glyphs inside the fonts are indexed by their Unicode numbers so it is natural to display them sorted by their position in the Unicode character set. Unicode is organised in named blocks such as "Basic Latin", "Latin-1 Supplement", etc., typically consisting of 265 characters each.[1] It is therefore helpful to use these block names as subtitles within the table, to more easily find the information one is looking for.

A common way to represent the number of a single Unicode character is U+ followed by four (or more) hexadecimal digits. For example, U+0041 represents the letter "A" and U+20AC the Euro currency symbol "€". We use this convention by showing a Unicode range of sixteen characters at the left of each table row, e.g., U+0040 - 004F, followed by the sixteen glyphs in the range. Thus that particular table row from the "Basic Latin" block would show something like

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0040 – 004F	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

If a Unicode character has no glyph representation in a given font then this is indicated

* This is version v1.0e of the package, dated 2021/10/29; the license is LPPL.

1. Some blocks are smaller, while those containing the Asian ideographs are much larger.

by a special symbol (by default a colored hyphen). By default some color is used, but we've grey scaled the output for ArsTEXnica. In order to easily locate any Unicode character the table shows by default sixteen hex digits as a column heading. For example, to find Euro currency symbol (U+20AC) one first finds the right row, which is the range U+20A0 - 20AF, and then the C column in that row, and the glyph is there (or an indication that the font is missing that glyph; the line shows that for some of the other slots).

It can be useful to compare two fonts with each other by filling the table with glyphs from a secondary font if the primary font is missing them. For example, the next display shows two rows of Latin Modern Math (black glyphs) and instead of showing a missing glyph symbol in most slots, we use the glyphs from New Computer Modern Math, which has a much larger glyph set (normally red glyphs with grey background but again, grey scaled for ArsTEXnica).

2. The user interface

The package offers one command to typeset a font table. The appearance of the table can be customised by specifying key/value pairs.

`\displayfonttable * [⟨key/value-list⟩] {⟨font-name⟩} [⟨font-features⟩]`

The ⟨*font-name*⟩ is the font to be displayed. This and the ⟨*font-features*⟩ argument are passed to fontspec, thus they should follow the conventions of that package for specifying a font. The ⟨*key/value-list*⟩ offers customisation possibilities discussed below. The `\displayfonttable*` is a variant of the command, intended for use with 8-bit legacy fonts. It presets some keys, but otherwise behaves identically. The preset values are:

nostatistics, display-block=none, hex-digits=head, range-end=FF

For details see the next section.

`\fonttablesetup {⟨key/value-list⟩}`

Instead of or in addition to specifying key/values to `\displayfonttable` it is possible to set them up as defaults. Inside `\displayfonttable` the defaults are applied first, so one can still overwrite their values for an individual table.

`\fonttableglyphcount`

While typesetting a font table the package keeps track of the number of glyphs it finds in the

font. After the table has finished, this value is available in `\fonttableglyphcount` and it is, for example, used when statistics are produced. At the start of the next table it is reset to zero.

2.1. Keys and their values

Several of the available keys are booleans accepting `true` or `false`. They usually exist in pairs so that one can specify the desired behaviour without needing to provide a value, e.g., specifying `header` is equivalent to specifying `header=true` or `noheader=false`, etc. In the lists below the default settings are indicated by an underline.

The first set of keys is concerned with the overall look and feel of the generated table.

header, noheader These keys determine whether a header to the table is produced.

title-format, title-format-cont These keys define what is provided as a header title or continuation title if the table consists of several pages. They expect code as their value. This code can contain `#1` and `#2` to denote the ⟨*font-name*⟩ and ⟨*font-features*⟩ arguments, respectively.

By default a title using the `\caption` command is produced; on continuation titles, the ⟨*font-features*⟩ are not shown. This is typeset as a longtable header row, so you need to use either `\multicolumn` or a `\caption` command — otherwise everything ends up in the first column.

These keys handle the inner parts of the table.

display-block The Unicode dataset is organised in named blocks that are typically 128 or 256 characters, though some are noticeably larger and a few are smaller. With the `display-block` key it is possible to specify if and how such blocks should be made visible. The following values are supported:

titles Above each display block that contains glyphs the Unicode title of the block is displayed.

rules Display blocks are indicated only by a `\midrule`.

none Display blocks are not indicated at all.

hex-digits To ease reading the table, rows of hex digits are added to it. Where or if this happens is controlled by this key. Allowed values for it are the following:

block A row of hex digits is placed at the beginning of each Unicode block containing glyphs in the displayed font.

foot A row is added to the foot of each table page.

head A row is added to the top of each table page.

head+foot A row is added to the top and the foot of each table page.

none All hex digit rows are suppressed.

hex-digits-font The font to use for the hex digits, by default `\ttfamily\scriptsize`.

color This key determines the color for parts of the table (hex digits and Unicode ranges).

It can be either `none` or a color specification as understood by the `\color` command. The default is `blue`.

The next set of keys allows altering the statistics that are produced.

`statistics`, `nostatistics` These keys determine whether some statistics are listed at the end of the table.

`statistics-font` The font used to typeset the statistics; the default is `\normalfont\small`.

`statistics-format` Code (text) to specify what should be typeset in the statistics. One can use `#1` for the ⟨*font-name*⟩ and `#2` for the glyph count. The material is typeset on a single line at the end of the table. If several lines are needed you need to use `\parbox` or a similar construct.

Another set of keys deals with customisation on the glyph level.

`glyph-width` All glyphs are typeset in a box with the same width, the default value is `6pt` which is suitable for most 10pt fonts and make the table fit comfortably into the text width of a typical document.

`missing-glyph` If a slot in a row doesn't have a glyph in the font you may still want display something to indicate this state. By giving the key a value any arbitrary glyph or material can be typeset. The default is to typeset a – (hyphen) in a special color.

Rows that contain no glyph whatsoever are not displayed at all. Instead a small vertical space is added to indicate the one or more rows are omitted.

`missing-glyph-font` The font used for displaying the missing glyphs (the default value is `\ttfamily\scriptsize`).

`missing-glyph-color` If not specified it uses the value specified with the `color` key. If you want a different color, e.g., `red`, you can use a color value or you can specify `none` to use no coloring.

You can make comparisons between two fonts, which is useful, for example when dealing with incomplete math fonts and you need to see how well the symbols from one font blend with the supplementary symbols from another font.

`compare-with` If given, the value is a ⟨*comparison-font-name*⟩ that is used to supply missing glyphs. This means that if the ⟨*font-name*⟩ to be displayed is missing a glyph in a slot, then the ⟨*comparison-font-name*⟩ is checked, and if that font has the glyph in question, it will be displayed instead of showing a missing glyph indicator.

`compare-color`, `compare-bgcolor` To distinguish real glyphs from missing but substituted glyphs, they can be colored specially (default `red`) and/or you can have their background colored (default is `black!10`, i.e., a light grey).

`statistics-compare-format` Code (text) to specify what should be typeset in the statistics when comparing two fonts. One can use `#1` for the ⟨*font-name*⟩ and `#2` for its glyph

count, `#3` is the name of the comparison font, `#4` its glyph count, `#5` for the number of glyphs missing in this font and `#6` the number of extra glyphs in it. This code is used instead of `statistics-format` when comparisons are made.

The material is typeset on a single line at the end of the table. If several lines are needed you need to use `\parbox` or a similar construct.

Finally there are two keys for restricting the display range.

`range-start`, `range-end` The full Unicode set of characters is huge and checking every slot to see if the current font contains a glyph in the slot takes a long time. If you know that font contains only a certain subset then you can speed up the table generation considerably by limiting the search (and consequently the output generation). The `range-start` specifies where to start with the search (default `0000`) and `range-end` gives the last slot that is tested (default `FFFF`).

Thus, by default we restrict the display to slots below `10000`, because text fonts seldom contain glyphs in the higher planes. But if you want to see everything of the font (as far as supported by this package) and are prepared to wait for the higher planes to be scanned, you can go up to a value of `FFFFF`.

These keys are also quite useful in combination with the previous `compare-with` key, to display only, for example, the Greek letters and see how glyphs from two fonts blend with each other.

2.2. A standalone interactive version

If you want to quickly display a single font, you can run `unicodefont.tex` through LuaTEX (or XETEX). Similar to `nfssfont.tex` (which is for 8-bit fonts with PDFTEX) it asks you a few questions and then generates the font table for you. There are fewer configuration options available, but this workflow saves you writing a document to get a one-off table.

Most font tables need several runs due to the use of longtable, which has to find the right width for the columns across several pages. The `unicodefont` file therefore remembers your selection from the previous run and asks you if you want to reapply it to speed up the process.

3. Notes on table data

If you look at some parts of a Unicode font table you see a number of slots that do not show a "missing glyph" sign, but nonetheless appear to be empty. For example:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0020 – 002F		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
U+0030 – 003F	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
U+0040 – 004F	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
U+0050 – 005F	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
U+0060 – 006F	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
U+0070 – 007F	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	-
U+00A0 – 00AF		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬		®	¯

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+00B0 – 00BF	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿

The reason is that Unicode contains a lot of special spaces or otherwise invisible characters, e.g., U+0020 is the normal space, U+00A0 is a non-breaking space, U+00AD is a soft-hyphen (what LaTeX users would indicate with \-), and so forth. Especially the row U+2000–200F in Table 6 looks strange as it appears to be totally empty, but in fact most of its slots contain spaces of different width.

General Punctuation

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+2000 – 200F															-	-
U+2010 – 201F	-	-	‒	–	—	―	‖	=	‘	’	,	-	“	”	„	-
U+2020 – 202F	†	‡	•	-	-	-	…	-	-	-	-	-	-	-	-	

Another somewhat surprising area is the "Mathematical Alphanumeric Symbols" block in math fonts, starting at U+1D400. There you see a number of missing characters, the first two being U+1D455 (math italic small h) and U+1D49D (math script B).

Mathematical Alphanumeric Symbols

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+1D400 – 1D40F	𝐀	𝐁	𝐂	𝐃	𝐄	𝐅	𝐆	𝐇	𝐈	𝐉	𝐊	𝐋	𝐌	𝐍	𝐎	𝐏
U+1D410 – 1D41F	𝐐	𝐑	𝐒	𝐓	𝐔	𝐕	𝐖	𝐗	𝐘	𝐙	𝐚	𝐛	𝐜	𝐝	𝐞	𝐟
U+1D420 – 1D42F	𝐠	𝐡	𝐢	𝐣	𝐤	𝐥	𝐦	𝐧	𝐨	𝐩	𝐪	𝐫	𝐬	𝐭	𝐮	𝐯
U+1D430 – 1D43F	𝐰	𝐱	𝐲	𝐳	𝐴	𝐵	𝐶	𝐷	𝐸	𝐹	𝐺	𝐻	𝐼	𝐽	𝐾	𝐿
U+1D440 – 1D44F	𝑀	𝑁	𝑂	𝑃	𝑄	𝑅	𝑆	𝑇	𝑈	𝑉	𝑊	𝑋	𝑌	𝑍	𝑎	𝑏
U+1D450 – 1D45F	𝑐	𝑑	𝑒	𝑓	𝑔	-	𝑖	𝑗	𝑘	𝑙	𝑚	𝑛	𝑜	𝑝	𝑞	𝑟
U+1D460 – 1D46F	𝑠	𝑡	𝑢	𝑣	𝑤	𝑥	𝑦	𝑧	𝑨	𝑩	𝑪	𝑫	𝑬	𝑭	𝑮	𝑯
U+1D470 – 1D47F	𝑰	𝑱	𝑲	𝑳	𝑴	𝑵	𝑶	𝑷	𝑸	𝑹	𝑺	𝑻	𝑼	𝑽	𝑾	𝑿
U+1D480 – 1D48F	𝒀	𝒁	𝒂	𝒃	𝒄	𝒅	𝒆	𝒇	𝒈	𝒉	𝒊	𝒋	𝒌	𝒍	𝒎	𝒏
U+1D490 – 1D49F	𝒐	𝒑	𝒒	𝒓	𝒔	𝒕	𝒖	𝒗	𝒘	𝒙	𝒚	𝒛	𝒜	-	𝒞	𝒟
U+1D4A0 – 1D4AF	-	-	𝒢	-	-	𝒥	𝒦	-	-	𝒩	𝒪	𝒫	𝒬	-	𝒮	𝒯

In this case the reason is *not* that the font fails to implement the characters, but that these characters have already been defined in earlier revisions of the Unicode standard in the lower Unicode plane. For example, the "h" is the Planck constant U+210E and U+212C is the script capital B, etc. The Unicode Consortium decided not to encode the *same* character twice, hence the apparent holes.

A. Examples

In this section we show the results of a few calls to `\displayfonttable`. The tables are a bit easier to navigate if they use color in some places, but for ArsTeXnica this is not practical, so we use black and grey.

A.1. Computer modern Sans — 7-bit font

Our first example is the original Computer Modern Sans, with character codes ≤ 127.
Command used:

```
\displayfonttable*[color=none, range-end=7F]{cmss10}
```

Table 1. cmss10

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0000 - 000F	Γ	Δ	Θ	Λ	Ξ	Π	Σ	Υ	Φ	Ψ	Ω	ff	fi	fl	ffi	ffl
U+0010 - 001F	ı	ȷ	`	´	ˇ	˘	¯	°	¸	ß	æ	œ	ø	Æ	Œ	Ø
U+0020 - 002F	˙	!	”	#	$	%	&	’	(	)	*	+	,	-	.	/
U+0030 - 003F	0	1	2	3	4	5	6	7	8	9	:	;	¡	=	¿	?
U+0040 - 004F	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
U+0050 - 005F	P	Q	R	S	T	U	V	W	X	Y	Z	[	“	]	^	˙
U+0060 - 006F	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
U+0070 - 007F	p	q	r	s	t	u	v	w	x	y	z	–	—	”	~	¨

A.2. TEX Gyre Heros — 8-bit font

This example shows the TEX Gyre Heros 8-bit font, in the T1 encoding, with character codes ≤ 255. Command used:

```
\displayfonttable*[color=none]{ec-qhvr}
```

Table 2. ec-qhvr

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0000 - 000F	`	´	ˆ	˜	¨	˝	°	ˇ	˘	¯	˙	¸	˛	,	‹	›
U+0010 - 001F	“	”	„	«	»	–	—		ₒ	ı	ȷ	ff	fi	fl	ffi	ffl
U+0020 - 002F	␣	!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
U+0030 - 003F	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
U+0040 - 004F	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
U+0050 - 005F	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
U+0060 - 006F	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
U+0070 - 007F	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	-
U+0080 - 008F	Ă	Ą	Ć	Č	Ď	Ě	Ę	Ğ	Ĺ	Ľ	Ł	Ń	Ň	Ŋ	Ő	Ŕ
U+0090 - 009F	Ř	Ś	Š	Ş	Ť	Ţ	Ű	Ů	Ÿ	Ź	Ž	Ż	IJ	İ	đ	§
U+00A0 - 00AF	ă	ą	ć	č	ď	ě	ę	ğ	Í	ľ	ł	ń	ň	ŋ	ő	ŕ
U+00B0 - 00BF	ř	ś	š	ş	ť	ţ	ű	ů	ÿ	ź	ž	ż	ij	¡	¿	£
U+00C0 - 00CF	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
U+00D0 - 00DF	Đ	Ñ	Ò	Ó	Ô	Õ	Ö	Œ	Ø	Ù	Ú	Û	Ü	Ý	Þ	SS
U+00E0 - 00EF	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
U+00F0 - 00FF	ð	ñ	ò	ó	ô	õ	ö	œ	ø	ù	ú	û	ü	ý	þ	ß

A.3. Latin Modern Math — 8-bit fonts

The traditional Latin Modern Math Italic, Symbol and Extension fonts. The symbol font (`lmsy10`) has two characters added to the Computer Modern symbol repertoire, seen in the last row of the table. Commands used:

```
\displayfonttable*[color=none]{lmmi10}
\displayfonttable*[color=none]{lmsy10}
\displayfonttable*[color=none]{lmex10}
```

Table 3. lmmi10

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0000 - 000F	Γ	Δ	Θ	Λ	Ξ	Π	Σ	Υ	Φ	Ψ	Ω	α	β	γ	δ	ϵ
U+0010 - 001F	ζ	η	θ	ι	κ	λ	μ	ν	ξ	π	ρ	σ	τ	υ	ϕ	χ
U+0020 - 002F	ψ	ω	ε	ϑ	ϖ	ϱ	ς	φ	$\leftharpoonup$	$\leftharpoondown$	$\rightharpoonup$	$\rightharpoondown$	$\lhook$	$\rhook$	$\triangleright$	$\triangleleft$
U+0030 - 003F	0	1	2	3	4	5	6	7	8	9	.	,	<	/	>	$\star$
U+0040 - 004F	∂	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
U+0050 - 005F	P	Q	R	S	T	U	V	W	X	Y	Z	$\flat$	$\natural$	$\sharp$	$\smile$	$\frown$
U+0060 - 006F	ℓ	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
U+0070 - 007F	p	q	r	s	t	u	v	w	x	y	z	$\imath$	$\jmath$	$\wp$	$\vec{}$	$\frown$

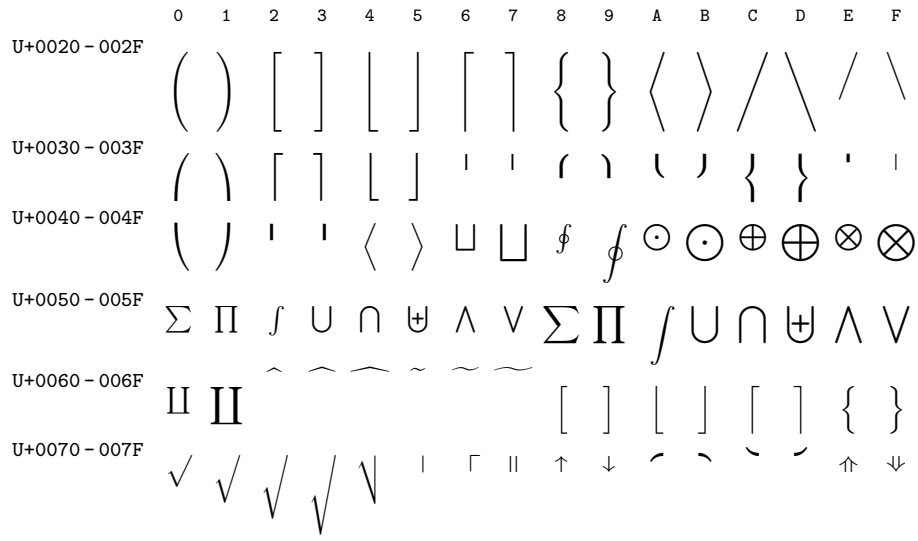
Table 4. lmsy10

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0000 - 000F	$-$	$\cdot$	$\times$	$*$	$\div$	$\diamond$	$\pm$	$\mp$	$\oplus$	$\ominus$	$\otimes$	$\oslash$	$\odot$	$\bigcirc$	$\circ$	$\bullet$
U+0010 - 001F	$\asymp$	$\equiv$	$\subseteq$	$\supseteq$	$\leq$	$\geq$	$\preceq$	$\succeq$	$\sim$	$\approx$	$\subset$	$\supset$	$\ll$	$\gg$	$\prec$	$\succ$
U+0020 - 002F	$\leftarrow$	$\rightarrow$	$\uparrow$	$\downarrow$	$\leftrightarrow$	$\nearrow$	$\searrow$	$\simeq$	$\Leftarrow$	$\Rightarrow$	$\Uparrow$	$\Downarrow$	$\Leftrightarrow$	$\nwarrow$	$\swarrow$	$\propto$
U+0030 - 003F	$\prime$	∞	$\in$	$\ni$	$\triangle$	$\triangledown$	$/$	$-$	$\forall$	$\exists$	$\neg$	$\emptyset$	$\Re$	$\Im$	$\top$	$\bot$
U+0040 - 004F	$\aleph$	$\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$	$\mathcal{D}$	$\mathcal{E}$	$\mathcal{F}$	$\mathcal{G}$	$\mathcal{H}$	$\mathcal{I}$	$\mathcal{J}$	$\mathcal{K}$	$\mathcal{L}$	$\mathcal{M}$	$\mathcal{N}$	$\mathcal{O}$
U+0050 - 005F	$\mathcal{P}$	$\mathcal{Q}$	$\mathcal{R}$	$\mathcal{S}$	$\mathcal{T}$	$\mathcal{U}$	$\mathcal{V}$	$\mathcal{W}$	$\mathcal{X}$	$\mathcal{Y}$	$\mathcal{Z}$	$\cup$	$\cap$	$\uplus$	$\wedge$	$\vee$
U+0060 - 006F	$\vdash$	$\dashv$	$\lfloor$	$\rfloor$	$\lceil$	$\rceil$	$\{$	$\}$	$\langle$	$\rangle$	$\mid$	$\parallel$	$\updownarrow$	$\Updownarrow$	$\backslash$	$\wr$
U+0070 - 007F	$\surd$	$\amalg$	∇	$\int$	$\sqcup$	$\sqcap$	$\sqsubseteq$	$\sqsupseteq$	$\S$	$\dagger$	$\ddagger$	$\P$	$\clubsuit$	$\diamondsuit$	$\heartsuit$	$\spadesuit$
U+00A0 - 00AF	-	-	-	-	-	-	-	-	-	-	-	-	$\leqslant$	$\geqslant$	-	-

Table 5. lmex10

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0000 - 000F	$($	$)$	$[$	$]$	$\lfloor$	$\rfloor$	$\lceil$	$\rceil$	$\{$	$\}$	$\langle$	$\rangle$	$\vert$	$\Vert$	$/$	$\backslash$
U+0010 - 001F	$\big($	$\big)$	$\Big($	$\Big)$	$\Big[$	$\Big]$	$\Big\lfloor$	$\Big\rfloor$	$\Big\lceil$	$\Big\rceil$	$\Big\{$	$\Big\}$	$\Big\langle$	$\Big\rangle$	$\Big/$	$\Big\backslash$

Table 5. lmex10 *cont.*



A.4. Latin Modern Math compared to New Computer Modern Math

This example shows the extra symbols available in New Computer Modern Math in comparison to Latin Modern Math as the base font. We use the following setup (including settings for the greyscaled ArsTEXnica output, as an example of color overrides):

```
\displayfonttable[hex-digits=head+foot, range-end=1FFFF,
                  compare-with=NewComputerModernMath-Book,
                  title-format=\caption{Latin Modern Math compared
                                        to New Computer Modern Math},
                  title-format-cont=\caption{LM Math vs.\ NewCM Math,
                                        \emph{cont.}},
                  compare-color=black,  compare-bgcolor=black!10,
                  missing-glyph-color=black!50, color=black!75]
                 {Latin Modern Math}
```

That is, glyphs only in NewCM are shown with a light grey background. We also extended the range to cover U+10000 to U+1FFFF in order to include the Unicode Math alphabets.

Table 6. Latin Modern Math compared to New Computer Modern Math

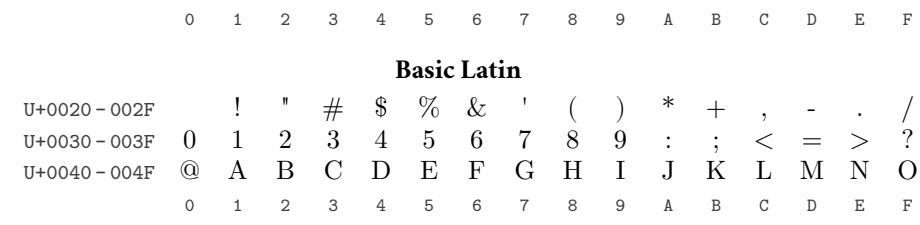


Table 6. LM Math vs. NewCM Math, *cont.*

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0050 – 005F	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
U+0060 – 006F	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
U+0070 – 007F	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Latin-1 Supplement

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+00A0 – 00AF		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬		®	¯
U+00B0 – 00BF	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
U+00C0 – 00CF	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
U+00D0 – 00DF	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
U+00E0 – 00EF	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
U+00F0 – 00FF	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Latin Extended-A

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0100 – 010F	Ā	ā	Ă	ă	Ą	ą	Ć	ć	Ĉ	ĉ	Ċ	ċ	Č	č	Ď	ď
U+0110 – 011F	Đ	đ	Ē	ē	Ĕ	ĕ	Ė	ė	Ę	ę	Ě	ě	Ĝ	ĝ	Ğ	ğ
U+0120 – 012F	Ġ	ġ	Ģ	ģ	Ĥ	ĥ	Ħ	ħ	Ĩ	ĩ	Ī	ī	Ĭ	ĭ	Į	į
U+0130 – 013F	İ	ı	IJ	ij	Ĵ	ĵ	Ķ	ķ	ĸ	Ĺ	ĺ	Ļ	ļ	Ľ	ľ	Ŀ
U+0140 – 014F	l·	Ł	ł	Ń	ń	Ņ	ņ	Ň	ň	ʼn	Ŋ	ŋ	Ō	ō	Ŏ	ŏ
U+0150 – 015F	Ő	ő	Œ	œ	Ŕ	ŕ	Ŗ	ŗ	Ř	ř	Ś	ś	Ŝ	ŝ	Ş	ş
U+0160 – 016F	Š	š	Ţ	ţ	Ť	ť	Ŧ	ŧ	Ũ	ũ	Ū	ū	Ŭ	ŭ	Ů	ů
U+0170 – 017F	Ű	ű	Ų	ų	Ŵ	ŵ	Ŷ	ŷ	Ÿ	Ź	ź	Ż	ż	Ž	ž	ſ

Latin Extended-B

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0180 – 018F	ƀ	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
U+01A0 – 01AF	Ơ	ơ	–	–	–	–	–	–	–	–	–	–	–	–	–	Ư
U+01B0 – 01BF	ư	–	–	–	–	Ƶ	–	–	–	–	–	–	–	–	–	–
U+0210 – 021F	–	–	–	–	–	–	–	–	Ș	ș	Ț	ț	–	–	–	–
U+0230 – 023F	–	–	–	–	–	–	–	ȷ	–	–	–	–	–	–	–	–

Spacing Modifier Letters

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+02C0 – 02CF	–	–	–	–	–	–	ˆ	ˇ	–	–	–	–	–	–	–	–
U+02D0 – 02DF	–	–	–	–	–	–	–	–	˘	˙	˚	˛	˜	˝	–	–

Combining Diacritical Marks

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0300 – 030F	̀	́	̂	̃	̄	̅	̆	̇	̈	̉	̊	̋	̌		̑	̒
U+0310 – 031F	̓	̂	̔	–	–	̕	–	–	–	–	̬	–	–	–	–	–
U+0320 – 032F	–	–	–	̣	–	–	̦	–	–	–	–	–	̳	̂	̳	̂

Table 6. LM Math vs. NewCM Math, *cont.*

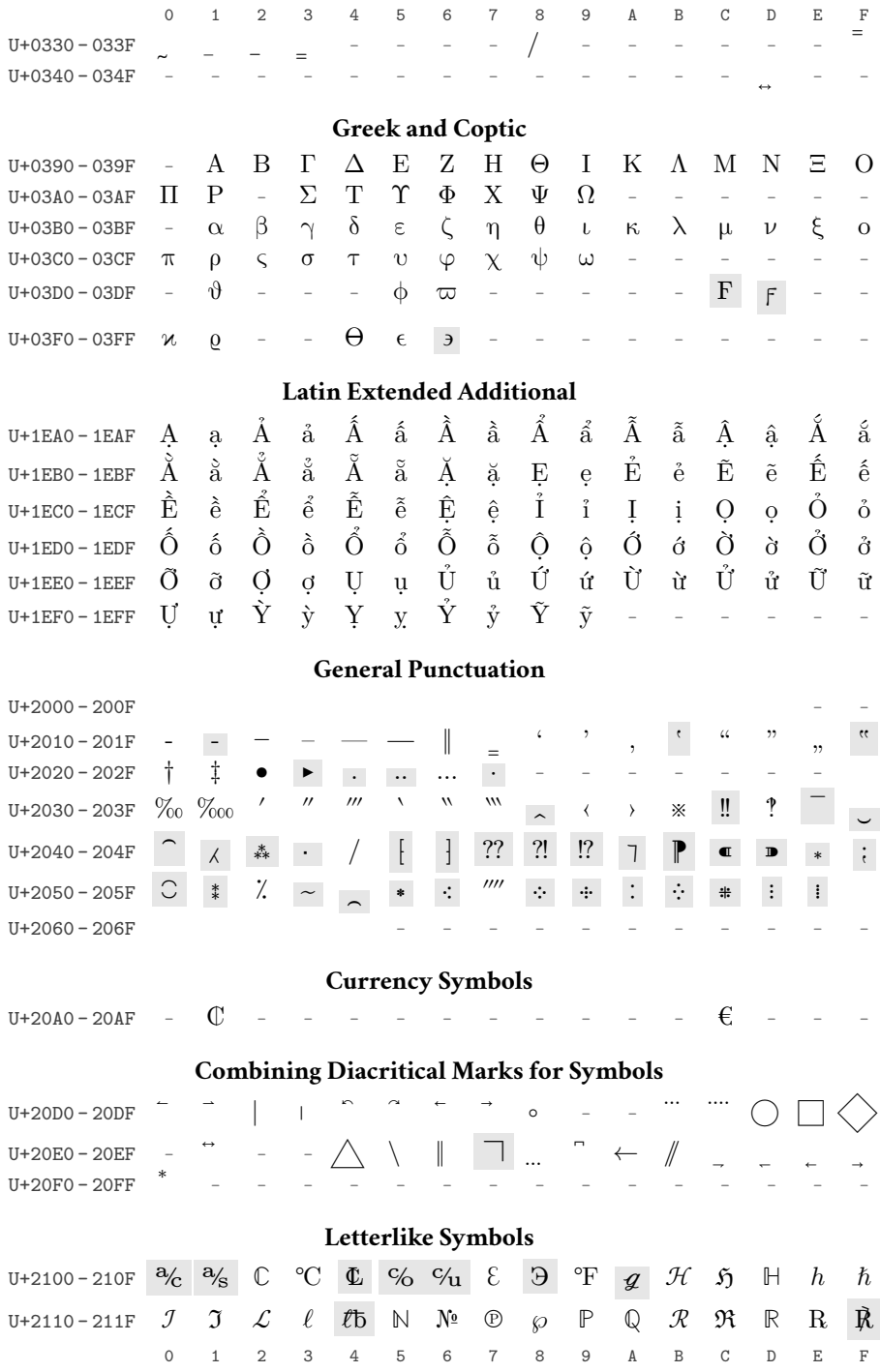
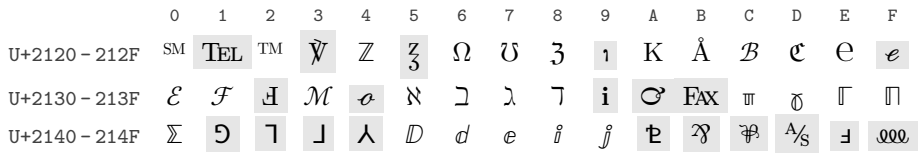
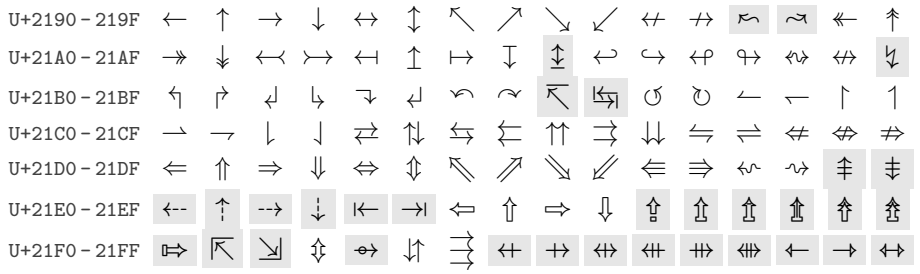


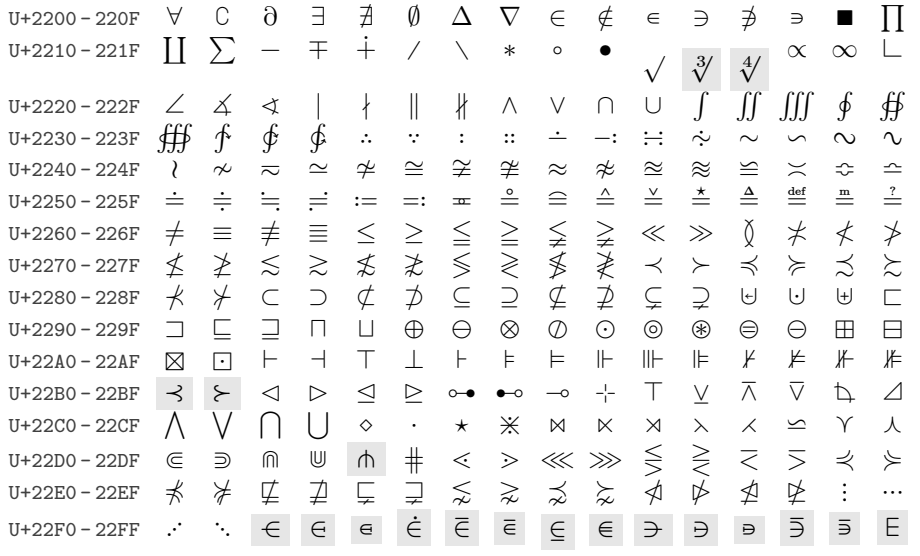
Table 6. LM Math vs. NewCM Math, *cont.*



Arrows



Mathematical Operators



Miscellaneous Technical

Table 6. LM Math vs. NewCM Math, *cont.*

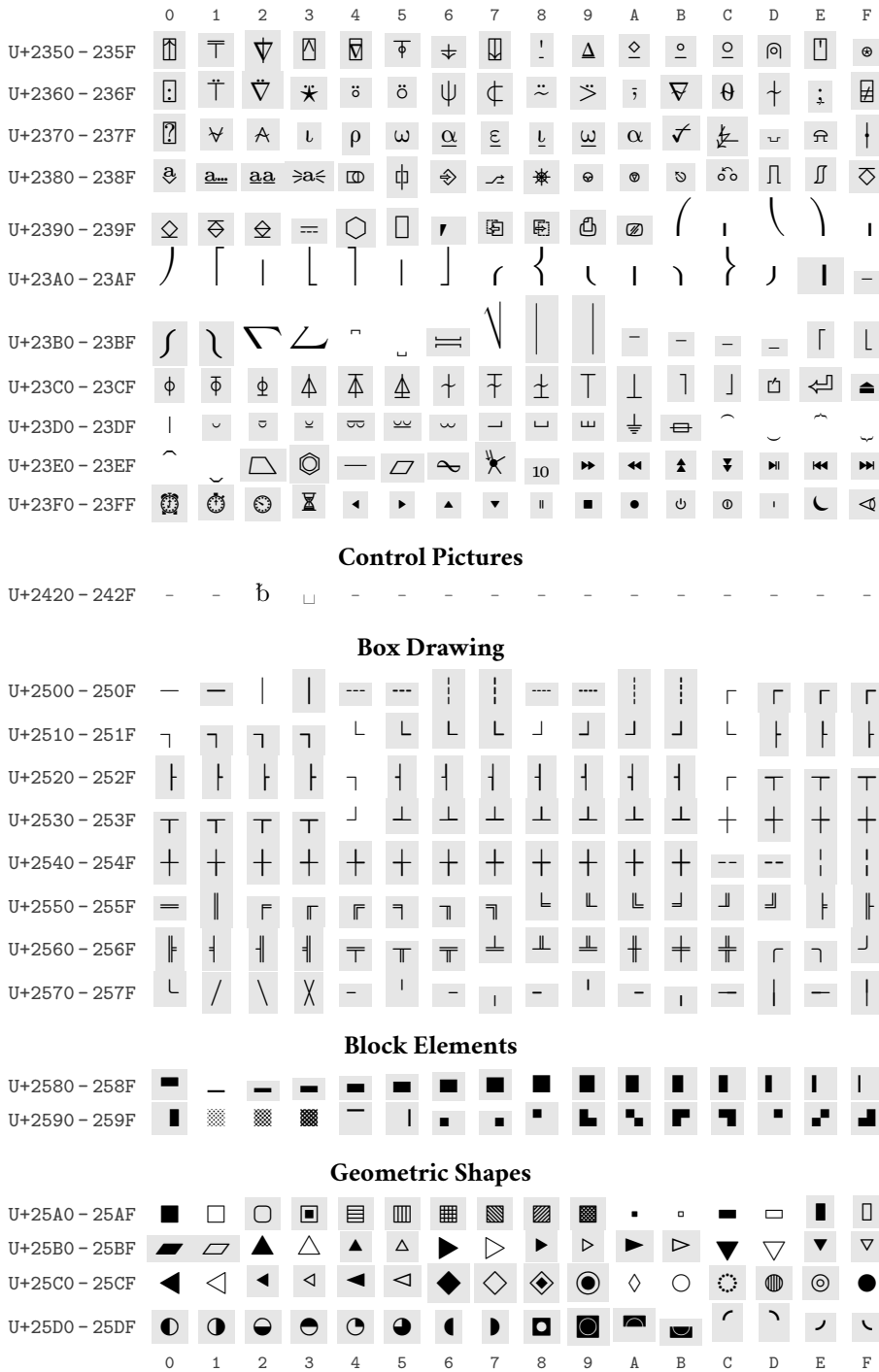


Table 6. LM Math vs. NewCM Math, *cont.*

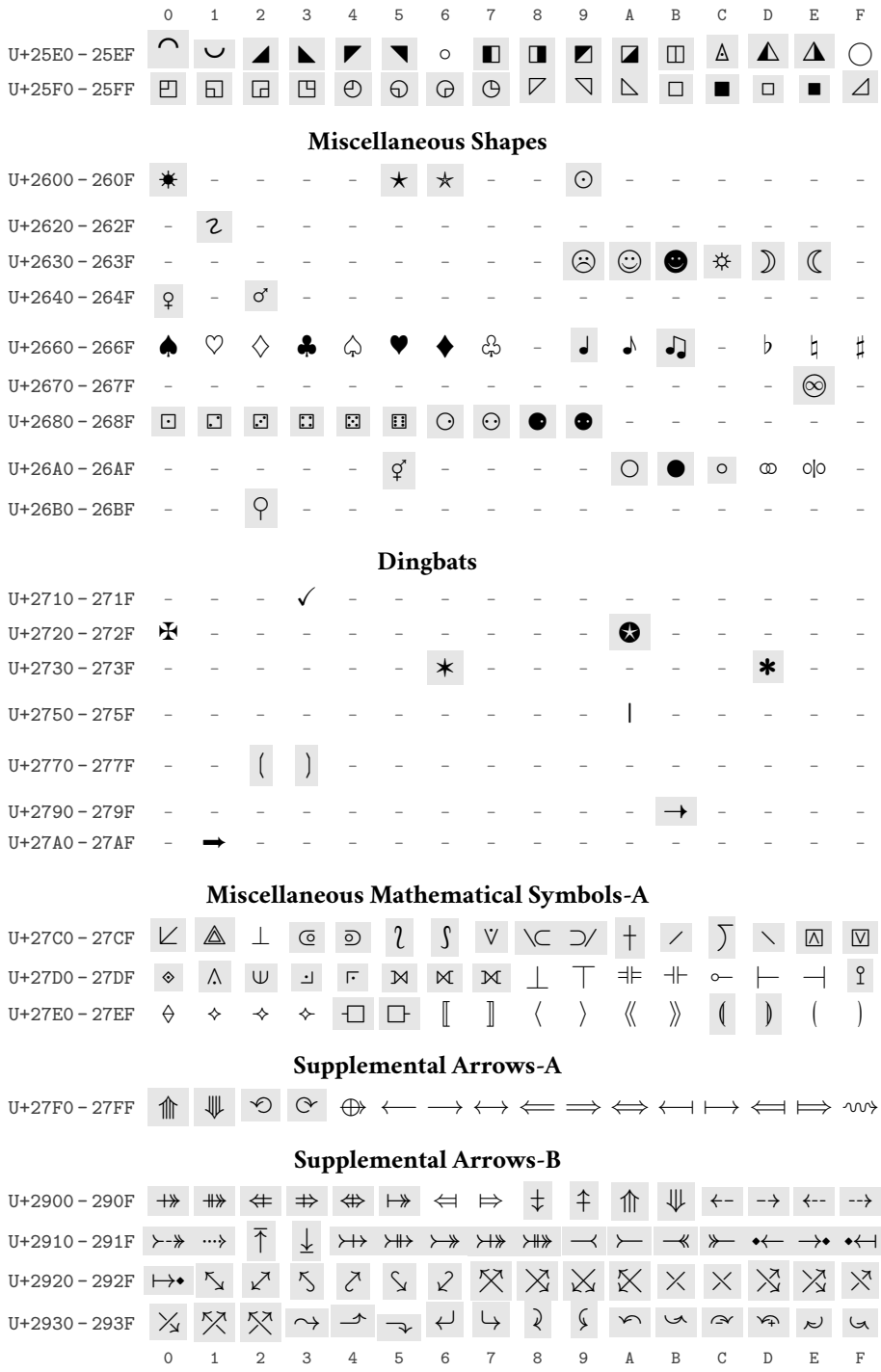
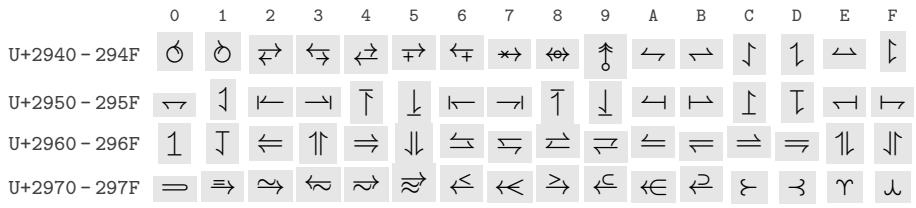
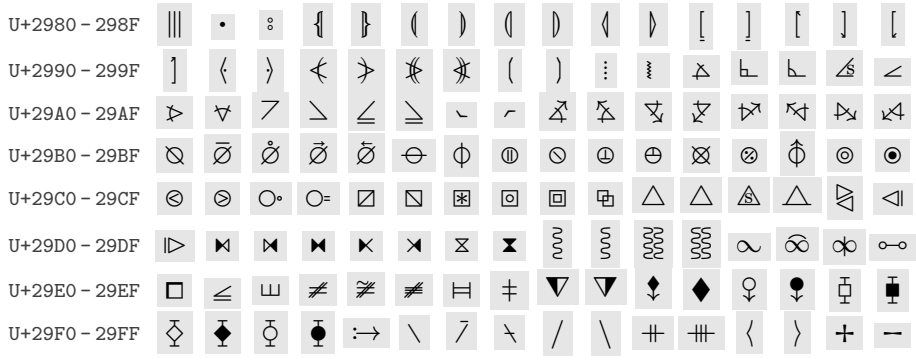


Table 6. LM Math vs. NewCM Math, *cont.*



Miscellaneous Mathematical Symbols-B



Supplemental Mathematical Operators

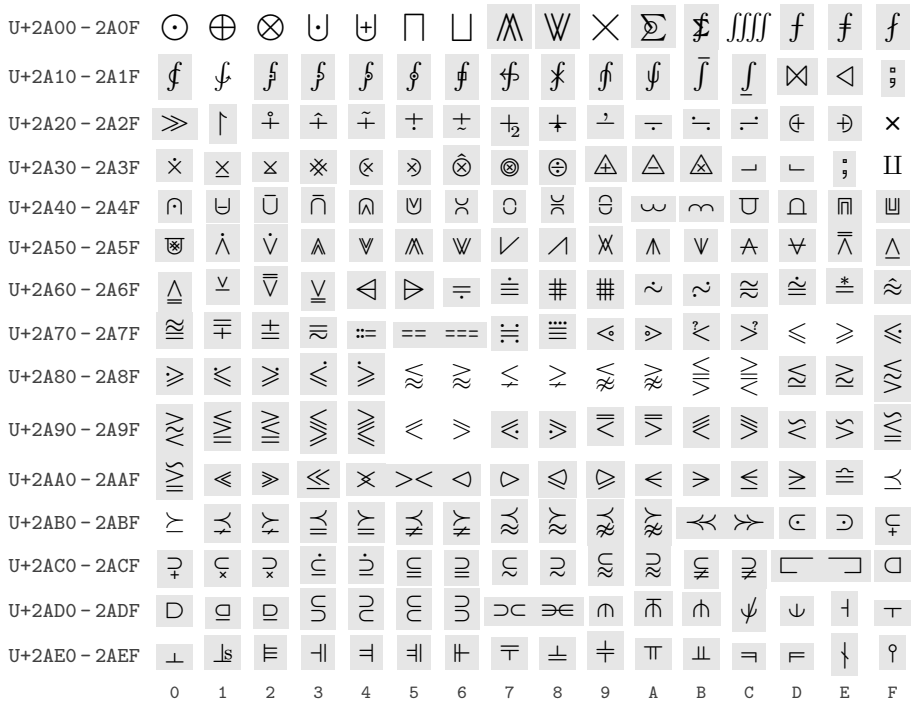


Table 6. LM Math vs. NewCM Math, *cont.*

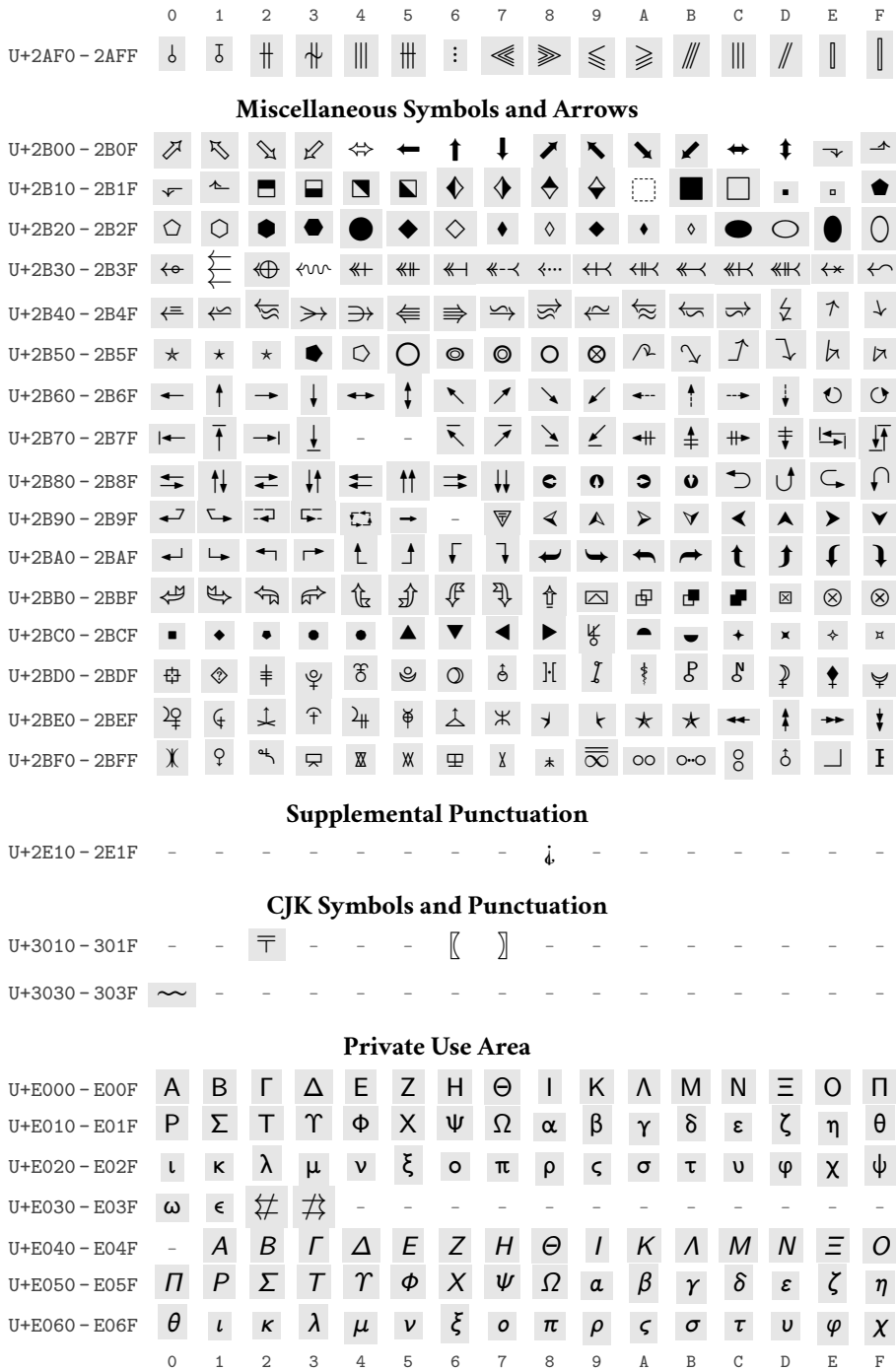


Table 6. LM Math vs. NewCM Math, *cont.*

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+E070 – E07F	ψ	ω	ϵ	–	–	–	–	–	–	–	–	–	–	–	–	–
U+E370 – E37F	–	–	–	–	–	–	$\oint$	$\oint$	–	–	–	–	–	–	–	–
U+E390 – E39F	–	–	–	–	–	$\oint$	–	$\oint$	$\oint$	$\oint$	$\oint$	$\oint$	–	–	–	–
U+E3D0 – E3DF	–	–	–	$\oint$	–	–	–	–	–	–	–	–	–	–	–	–
U+EA50 – EA5F	–	–	–	–	–	–	–	$\oint$	–	–	–	–	–	–	–	–

Alphabetic Presentation Forms

U+FB00 – FB0F	ff	fi	fl	ffi	ffl	–	–	–	–	–	–	–	–	–	–	–

Arabic Presentation Forms-B

U+FEF0 – FEFF	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	

Mathematical Alphanumeric Symbols

U+1D400 – 1D40F	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**
U+1D410 – 1D41F	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	**a**	**b**	**c**	**d**	**e**	**f**
U+1D420 – 1D42F	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**	**p**	**q**	**r**	**s**	**t**	**u**	**v**
U+1D430 – 1D43F	**w**	**x**	**y**	**z**	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*
U+1D440 – 1D44F	*M*	*N*	*O*	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*	*a*	*b*
U+1D450 – 1D45F	*c*	*d*	*e*	*f*	*g*	–	*i*	*j*	*k*	*l*	*m*	*n*	*o*	*p*	*q*	*r*
U+1D460 – 1D46F	*s*	*t*	*u*	*v*	*w*	*x*	*y*	*z*	***A***	***B***	***C***	***D***	***E***	***F***	***G***	***H***
U+1D470 – 1D47F	***I***	***J***	***K***	***L***	***M***	***N***	***O***	***P***	***Q***	***R***	***S***	***T***	***U***	***V***	***W***	***X***
U+1D480 – 1D48F	***Y***	***Z***	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***	***k***	***l***	***m***	***n***
U+1D490 – 1D49F	***o***	***p***	***q***	***r***	***s***	***t***	***u***	***v***	***w***	***x***	***y***	***z***	$\mathcal{A}$	–	$\mathcal{C}$	$\mathcal{D}$
U+1D4A0 – 1D4AF	–	–	$\mathcal{G}$	–	–	$\mathcal{J}$	$\mathcal{K}$	–	–	$\mathcal{N}$	$\mathcal{O}$	$\mathcal{P}$	$\mathcal{Q}$	–	$\mathcal{S}$	$\mathcal{T}$
U+1D4B0 – 1D4BF	$\mathcal{U}$	$\mathcal{V}$	$\mathcal{W}$	$\mathcal{X}$	$\mathcal{Y}$	$\mathcal{Z}$	$\mathcal{a}$	$\mathcal{b}$	$\mathcal{c}$	$\mathcal{d}$	–	$\mathcal{f}$	–	$\mathcal{h}$	$\mathcal{i}$	$\mathcal{j}$
U+1D4C0 – 1D4CF	$\mathcal{k}$	$\mathcal{l}$	$\mathcal{m}$	$\mathcal{n}$	–	$\mathcal{p}$	$\mathcal{q}$	$\mathcal{r}$	$\mathcal{s}$	$\mathcal{t}$	$\mathcal{u}$	$\mathcal{v}$	$\mathcal{w}$	$\mathcal{x}$	$\mathcal{y}$	$\mathcal{z}$
U+1D4D0 – 1D4DF	$\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$	$\mathcal{D}$	$\mathcal{E}$	$\mathcal{F}$	$\mathcal{G}$	$\mathcal{H}$	$\mathcal{I}$	$\mathcal{J}$	$\mathcal{K}$	$\mathcal{L}$	$\mathcal{M}$	$\mathcal{N}$	$\mathcal{O}$	$\mathcal{P}$
U+1D4E0 – 1D4EF	$\mathcal{Q}$	$\mathcal{R}$	$\mathcal{S}$	$\mathcal{T}$	$\mathcal{U}$	$\mathcal{V}$	$\mathcal{W}$	$\mathcal{X}$	$\mathcal{Y}$	$\mathcal{Z}$	$\mathcal{a}$	$\mathcal{b}$	$\mathcal{c}$	$\mathcal{d}$	$\mathcal{e}$	$\mathcal{f}$
U+1D4F0 – 1D4FF	$\mathcal{g}$	$\mathcal{h}$	$\mathcal{i}$	$\mathcal{j}$	$\mathcal{k}$	$\mathcal{l}$	$\mathcal{m}$	$\mathcal{n}$	$\mathcal{o}$	$\mathcal{p}$	$\mathcal{q}$	$\mathcal{r}$	$\mathcal{s}$	$\mathcal{t}$	$\mathcal{u}$	$\mathcal{v}$
U+1D500 – 1D50F	$\mathcal{w}$	$\mathcal{x}$	$\mathcal{y}$	$\mathcal{z}$	$\mathfrak{A}$	$\mathfrak{B}$	–	$\mathfrak{D}$	$\mathfrak{E}$	$\mathfrak{F}$	$\mathfrak{G}$	–	–	$\mathfrak{J}$	$\mathfrak{K}$	$\mathfrak{L}$
U+1D510 – 1D51F	$\mathfrak{M}$	$\mathfrak{N}$	$\mathfrak{O}$	$\mathfrak{P}$	$\mathfrak{Q}$	–	$\mathfrak{S}$	$\mathfrak{T}$	$\mathfrak{U}$	$\mathfrak{V}$	$\mathfrak{W}$	$\mathfrak{X}$	$\mathfrak{Y}$	–	$\mathfrak{a}$	$\mathfrak{b}$
U+1D520 – 1D52F	$\mathfrak{c}$	$\mathfrak{d}$	$\mathfrak{e}$	$\mathfrak{f}$	$\mathfrak{g}$	$\mathfrak{h}$	$\mathfrak{i}$	$\mathfrak{j}$	$\mathfrak{k}$	$\mathfrak{l}$	$\mathfrak{m}$	$\mathfrak{n}$	$\mathfrak{o}$	$\mathfrak{p}$	$\mathfrak{q}$	$\mathfrak{r}$
U+1D530 – 1D53F	$\mathfrak{s}$	$\mathfrak{t}$	$\mathfrak{u}$	$\mathfrak{v}$	$\mathfrak{w}$	$\mathfrak{x}$	$\mathfrak{y}$	$\mathfrak{z}$	$\mathbb{A}$	$\mathbb{B}$	–	$\mathbb{D}$	$\mathbb{E}$	$\mathbb{F}$	$\mathbb{G}$	–
U+1D540 – 1D54F	$\mathbb{I}$	$\mathbb{J}$	$\mathbb{K}$	$\mathbb{L}$	$\mathbb{M}$	–	$\mathbb{O}$	–	–	–	$\mathbb{S}$	$\mathbb{T}$	$\mathbb{U}$	$\mathbb{V}$	$\mathbb{W}$	$\mathbb{X}$
U+1D550 – 1D55F	$\mathbb{Y}$	–	$\mathbb{a}$	$\mathbb{b}$	$\mathbb{c}$	$\mathbb{d}$	$\mathbb{e}$	$\mathbb{f}$	$\mathbb{g}$	$\mathbb{h}$	$\mathbb{i}$	$\mathbb{j}$	$\mathbb{k}$	$\mathbb{l}$	$\mathbb{m}$	$\mathbb{n}$
U+1D560 – 1D56F	$\mathbb{o}$	$\mathbb{p}$	$\mathbb{q}$	$\mathbb{r}$	$\mathbb{s}$	$\mathbb{t}$	$\mathbb{u}$	$\mathbb{v}$	$\mathbb{w}$	$\mathbb{x}$	$\mathbb{y}$	$\mathbb{z}$	$\mathbf{\mathfrak{A}}$	$\mathbf{\mathfrak{B}}$	$\mathbf{\mathfrak{C}}$	$\mathbf{\mathfrak{D}}$
U+1D570 – 1D57F	$\mathbf{\mathfrak{E}}$	$\mathbf{\mathfrak{F}}$	$\mathbf{\mathfrak{G}}$	$\mathbf{\mathfrak{H}}$	$\mathbf{\mathfrak{I}}$	$\mathbf{\mathfrak{J}}$	$\mathbf{\mathfrak{K}}$	$\mathbf{\mathfrak{L}}$	$\mathbf{\mathfrak{M}}$	$\mathbf{\mathfrak{N}}$	$\mathbf{\mathfrak{O}}$	$\mathbf{\mathfrak{P}}$	$\mathbf{\mathfrak{Q}}$	$\mathbf{\mathfrak{R}}$	$\mathbf{\mathfrak{S}}$	$\mathbf{\mathfrak{T}}$
U+1D580 – 1D58F	$\mathbf{\mathfrak{U}}$	$\mathbf{\mathfrak{V}}$	$\mathbf{\mathfrak{W}}$	$\mathbf{\mathfrak{X}}$	$\mathbf{\mathfrak{Y}}$	$\mathbf{\mathfrak{Z}}$	$\mathbf{\mathfrak{a}}$	$\mathbf{\mathfrak{b}}$	$\mathbf{\mathfrak{c}}$	$\mathbf{\mathfrak{d}}$	$\mathbf{\mathfrak{e}}$	$\mathbf{\mathfrak{f}}$	$\mathbf{\mathfrak{g}}$	$\mathbf{\mathfrak{h}}$	$\mathbf{\mathfrak{i}}$	$\mathbf{\mathfrak{j}}$
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Table 6. LM Math vs. NewCM Math, *cont.*

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+1D590 – 1D59F	𝔈	𝔉	𝔊	𝔋	𝔬	𝔭	𝔮	𝔯	𝔰	𝔱	𝔲	𝔳	𝔴	𝔵	𝔶	𝔷
U+1D5A0 – 1D5AF	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
U+1D5B0 – 1D5BF	Q	R	S	T	U	V	W	X	Y	Z	a	b	c	d	e	f
U+1D5C0 – 1D5CF	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
U+1D5D0 – 1D5DF	w	x	y	z	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**
U+1D5E0 – 1D5EF	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	**a**	**b**
U+1D5F0 – 1D5FF	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**	**p**	**q**	**r**
U+1D600 – 1D60F	**s**	**t**	**u**	**v**	**w**	**x**	**y**	**z**	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*
U+1D610 – 1D61F	*I*	*J*	*K*	*L*	*M*	*N*	*O*	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*
U+1D620 – 1D62F	*Y*	*Z*	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*	*l*	*m*	*n*
U+1D630 – 1D63F	*o*	*p*	*q*	*r*	*s*	*t*	*u*	*v*	*w*	*x*	*y*	*z*	***A***	***B***	***C***	***D***
U+1D640 – 1D64F	***E***	***F***	***G***	***H***	***I***	***J***	***K***	***L***	***M***	***N***	***O***	***P***	***Q***	***R***	***S***	***T***
U+1D650 – 1D65F	***U***	***V***	***W***	***X***	***Y***	***Z***	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***
U+1D660 – 1D66F	***k***	***l***	***m***	***n***	***o***	***p***	***q***	***r***	***s***	***t***	***u***	***v***	***w***	***x***	***y***	***z***
U+1D670 – 1D67F	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
U+1D680 – 1D68F	Q	R	S	T	U	V	W	X	Y	Z	a	b	c	d	e	f
U+1D690 – 1D69F	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
U+1D6A0 – 1D6AF	w	x	y	z	*ı*	*ȷ*	–	–	**Α**	**Β**	**Γ**	**Δ**	**Ε**	**Ζ**	**Η**	**Θ**
U+1D6B0 – 1D6BF	**Ι**	**Κ**	**Λ**	**Μ**	**Ν**	**Ξ**	**Ο**	**Π**	**Ρ**	**ϴ**	**Σ**	**Τ**	**Υ**	**Φ**	**Χ**	**Ψ**
U+1D6C0 – 1D6CF	**Ω**	**∇**	**α**	**β**	**γ**	**δ**	**ε**	**ζ**	**η**	**θ**	**ι**	**κ**	**λ**	**μ**	**ν**	**ξ**
U+1D6D0 – 1D6DF	**ο**	**π**	**ρ**	**ς**	**σ**	**τ**	**υ**	**φ**	**χ**	**ψ**	**ω**	**∂**	**ϵ**	**ϑ**	**ϰ**	**ϕ**
U+1D6E0 – 1D6EF	**ϱ**	**ϖ**	*Α*	*Β*	*Γ*	*Δ*	*Ε*	*Ζ*	*Η*	*Θ*	*Ι*	*Κ*	*Λ*	*Μ*	*Ν*	*Ξ*
U+1D6F0 – 1D6FF	*Ο*	*Π*	*Ρ*	*ϴ*	*Σ*	*Τ*	*Υ*	*Φ*	*Χ*	*Ψ*	*Ω*	*∇*	*α*	*β*	*γ*	*δ*
U+1D700 – 1D70F	*ε*	*ζ*	*η*	*θ*	*ι*	*κ*	*λ*	*μ*	*ν*	*ξ*	*ο*	*π*	*ρ*	*ς*	*σ*	*τ*
U+1D710 – 1D71F	*υ*	*φ*	*χ*	*ψ*	*ω*	*∂*	*ϵ*	*ϑ*	*ϰ*	*ϕ*	*ϱ*	*ϖ*	***A***	***B***	***Γ***	***Δ***
U+1D720 – 1D72F	***E***	***Z***	***H***	***ϴ***	***I***	***K***	***Λ***	***M***	***N***	***Ξ***	***O***	***Π***	***P***	***ϴ***	***Σ***	***T***
U+1D730 – 1D73F	***Υ***	***Φ***	***X***	***Ψ***	***Ω***	***∇***	***α***	***β***	***γ***	***δ***	***ε***	***ζ***	***η***	***θ***	***ι***	***κ***
U+1D740 – 1D74F	***λ***	***μ***	***ν***	***ξ***	***ο***	***π***	***ρ***	***ς***	***σ***	***τ***	***υ***	***φ***	***χ***	***ψ***	***ω***	***∂***
U+1D750 – 1D75F	***ϵ***	***ϑ***	***ϰ***	***ϕ***	***ϱ***	***ϖ***	**A**	**B**	**Γ**	**Δ**	**E**	**Z**	**H**	**Θ**	**I**	**K**
U+1D760 – 1D76F	**Λ**	**M**	**N**	**Ξ**	**O**	**Π**	**P**	**ϴ**	**Σ**	**T**	**Υ**	**Φ**	**X**	**Ψ**	**Ω**	**∇**
U+1D770 – 1D77F	**α**	**β**	**γ**	**δ**	**ε**	**ζ**	**η**	**θ**	**ι**	**κ**	**λ**	**μ**	**ν**	**ξ**	**ο**	**π**
U+1D780 – 1D78F	**ρ**	**ς**	**σ**	**τ**	**υ**	**φ**	**χ**	**ψ**	**ω**	**∂**	**ϵ**	**ϑ**	**ϰ**	**ϕ**	**ϱ**	**ϖ**
U+1D790 – 1D79F	***A***	***B***	***Γ***	***Δ***	***E***	***Z***	***H***	***ϴ***	***I***	***K***	***Λ***	***M***	***N***	***Ξ***	***O***	***Π***
U+1D7A0 – 1D7AF	***P***	***ϴ***	***Σ***	***T***	***Υ***	***Φ***	***X***	***Ψ***	***Ω***	***∇***	***α***	***β***	***γ***	***δ***	***ε***	***ζ***
U+1D7B0 – 1D7BF	***η***	***θ***	***ι***	***κ***	***λ***	***μ***	***ν***	***ξ***	***ο***	***π***	***ρ***	***ς***	***σ***	***τ***	***υ***	***φ***
U+1D7C0 – 1D7CF	***χ***	***ψ***	***ω***	***∂***	***ϵ***	***ϑ***	***ϰ***	***ϕ***	***ϱ***	***ϖ***	**F**	*F*	–	–	**0**	**1**
U+1D7D0 – 1D7DF	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	𝟘	𝟙	𝟚	𝟛	𝟜	𝟝	𝟞	𝟟
U+1D7E0 – 1D7EF	𝟠	𝟡	0	1	2	3	4	5	6	7	8	9	**0**	**1**	**2**	**3**
U+1D7F0 – 1D7FF	**4**	**5**	**6**	**7**	**8**	**9**	0	1	2	3	4	5	6	7	8	9

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Table 6. LM Math vs. NewCM Math, *cont.*

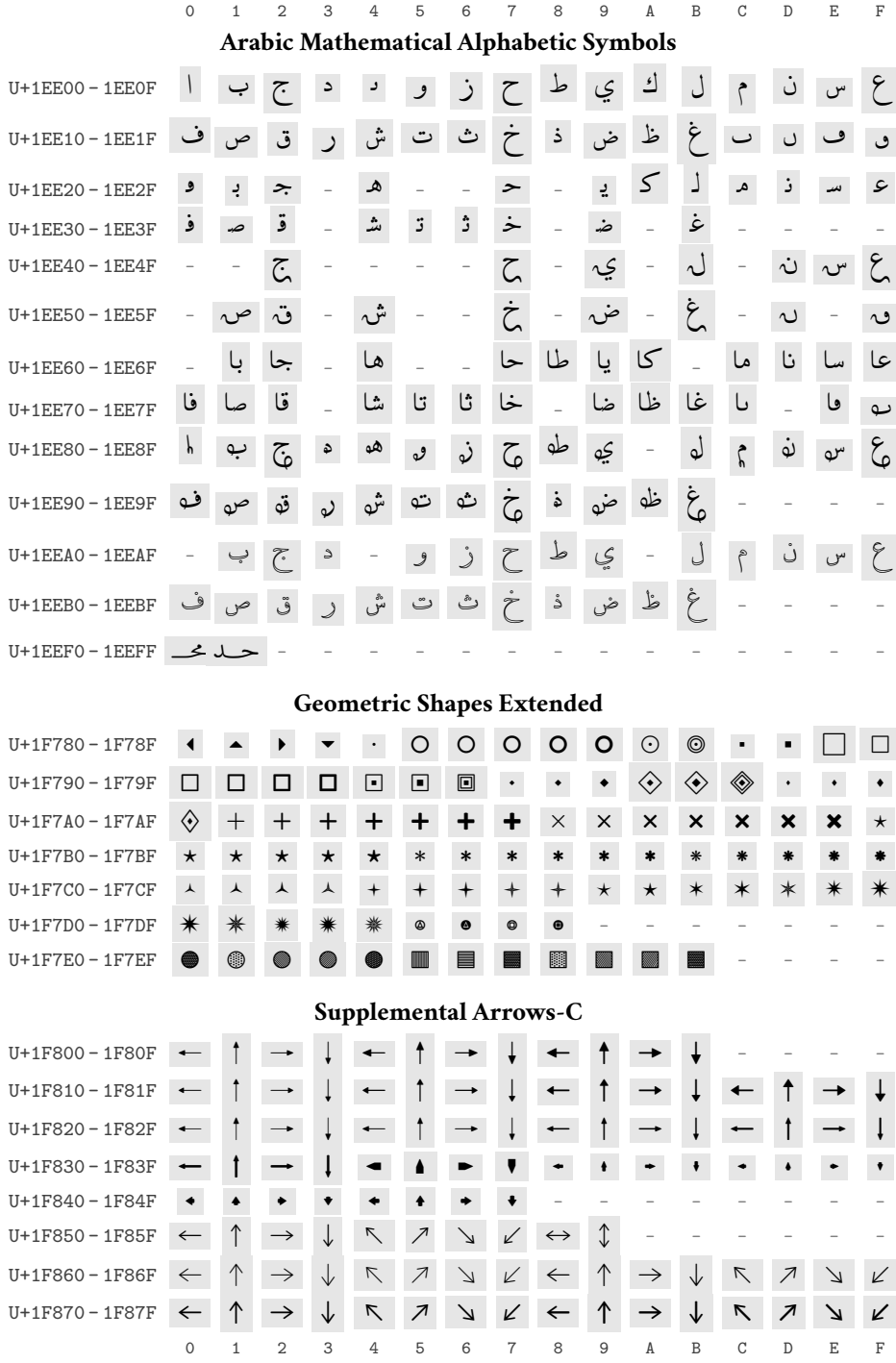
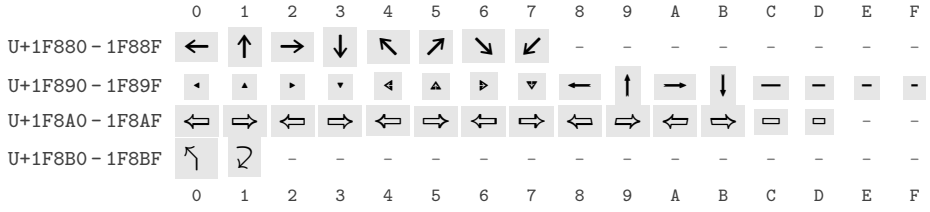


Table 6. LM Math vs. NewCM Math, *cont.*



Total number of glyphs in `Latin Modern Math`: 2045
Comparison font `NewComputerModernMath-Book` has 0 missing and 1959 extra glyphs

A.5.

Garamond Libre's Byzantine Musical symbols

As a final example we exhibit the Byzantine Musical Symbols as provided by Garamond Libre. Command used:

```
\displayfonttable[range-start=1D000, range-end=1D0FF,
                  hex-digits=block,
                  missing-glyph-color=black!50, color=black!75,
                  statistics-format=Total number of glyphs in
                    this block of #1 is #2]
                 {Garamond Libre}
```

Note that we have altered the text produced by the statistics, because the default is somewhat misleading if only a portion of the font is displayed. This produces the following table:

Table 7. Garamond Libre

Byzantine Musical Symbols

Table 7. Garamond Libre *cont.*

U+1D0E0 – 1D0EF	𝃠	𝃡	𝃢	𝃣	𝃤	𝃥	𝃦	𝃧	𝃨	𝃩	𝃪	𝃫	𝃬	𝃭	𝃮	𝃯
U+1D0F0 – 1D0FF	𝃰	𝃱	𝃲	𝃳	𝃴	𝃵	–	–	–	–	–	–	–	–	–	–

Total number of glyphs in this block of Garamond Libre is 246

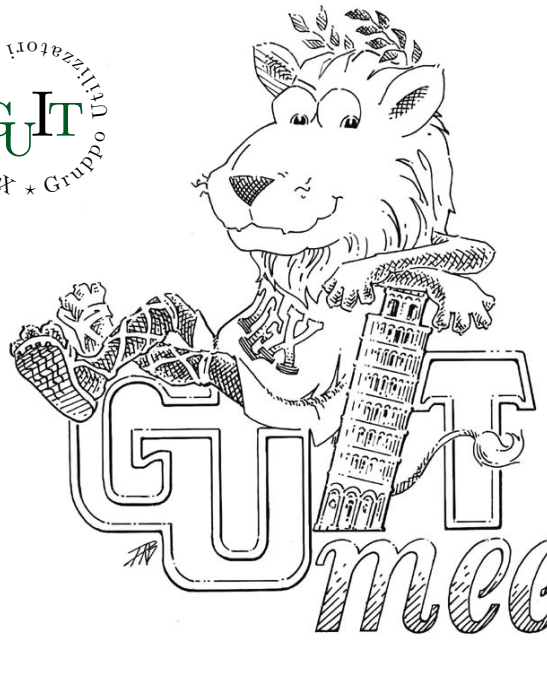
Frank Mittelbach
Mainz, Germany
https://www.latex-project.org
https://ctan.org/pkg/unicodefonttable

Progetto grafico di copertina
a cura di Ivan Valbusa

Immagine di copertina:
Typesetting in wood
(Raphael Schaller, 2016)
https://unsplash.com/photos/GkinCd2enIY

Questa rivista è stata prodotta
dal Gruppo Utilizzatori Italiani di TEX
usando esclusivamente software libero.

Versione elettronica per la diffusione via web.



Diciannovesimo convegno nazionale su TEX, LATEX e tipografia digitale

Online
ottobre 2022

Call for Papers

Si terrà di nuovo per via telematica il diciannovesimo Convegno annuale su TEX, LATEX e tipografia digitale organizzato dal Gruppo Utilizzatori Italiani di TEX. Il Convegno sarà un momento di ritrovo e di confronto per la comunità LATEX italiana, tramite una serie di interventi atti sia a contribuire all'arricchimento sia a supportarne lo sviluppo.
Maggiori informazioni sul Convegno e sulle modalità di presentazione degli interventi saranno disponibili all'indirizzo:

https://www.guitex.org/home/guit-meeting-2022/


20033
9 771828 236001