Numero 10
Ottobre
2010

# ArsTEXnica

Rivista italiana di TEX e LATEX

Atti del GUIT *meeting 2010*

GUIT

http://www.guit.sssup.it/arstexnica

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 ed 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 quasi trent'anni fa da Donald Knuth.

Le uscite avranno, almeno inizialmente, cadenza semestrale e verranno pubblicate nei mesi di Aprile e Ottobre. In particolare, la seconda uscita dell'anno conterrà gli Atti del Convegno Annuale del GUIT, che si tiene in quel periodo.

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.

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

`arstexnica@sssup.it`.

**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 Ufficio Statistica
Scuola Superiore Sant'Anna
Piazza Martiri della Libertà 33
56127 Pisa, Italia.
`http://www.guit.sssup.it`
`guit@sssup.it`

*Redazione ArsTEXnica*:
`http://www.guit.sssup.it/arstexnica/`
`arstexnica@sssup.it`

Codice ISSN 1828-2369

Stampata in Italia
Pisa: 15 Ottobre 2010

## Programma del Convegno

### Sessione Mattutina

**09:00** – **Registrazione**

**09:30** – **Installare TEX Live su Ubuntu**
*Enrico Gregorio*, Dipartimento di Informatica, Università di Verona

**10:00** – **Le graffe queste sconosciute**
*Claudio Beccari*

**10:30** – `illumino`: **An XML document production system with a TEX core**
*Matteo Centonza*, Metatype

**11:00** – **Coffee break**

**11:30** – **PDF/A-1a in ConTEXt-mkiv**
*Luigi Scarso*, Logo S.r.l.

**12:00** – **Presentazioni animate in LATEX**
*Gianluca Pignalberi*

**12:30** – **Managing Printed and On-Line Versions of Big-Sized Educational Documents**
*Jean Michel Hufflen*, Université de Franche Comté

**13:15** – **Pausa Pranzo**

### Sessione Pomeridiana

**14:30** – **Discussione sulle prospettive del gruppo**
*Gianluca Pignalberi*

**16:00** – **Chiusura dei lavori**

# ArsTEXnica

Rivista italiana di TEX e LATEX

*Numero 10, Ottobre 2010*



Gruppo Utilizzatori Italiani di TEX

# Editoriale

*Gianluca Pignalberi, Massimiliano Dominici*

Questo è il mio ultimo editoriale in qualità di direttore di ArsTEXnica. È arrivato il momento di rimettere il mio mandato. Sono intenzionato a ricandidarmi alla stessa carica, ma auspico che ci siano altri candidati, perché è importante che il nostro giornale viva delle esperienze di tutti.

Sono altresí convinto che i membri del GUIT, che pure mi hanno sempre manifestato il loro apprezzamento, siano per il cambiamento, e nel gruppo ci sono esperti validissimi in grado di migliorare un prodotto nato molto bene.

L'eventuale nuovo direttore sarà il benvenuto e avrà da me tutto l'appoggio e aiuto di cui riterrà aver bisogno.

Siccome devo lasciare la parola a Massimiliano Dominici per l'*EuroTEX* e altro, vi accenno brevemente gli articoli di questa uscita di ArsTEXnica, relativa al GUIT *meeting 2010*.

Iniziamo con Enrico Gregorio. Il suo articolo sull'installazione di TEXLive 2010 e relativo affiancamento (non *sostituzione*) a una precedente versione presente in Ubuntu è molto interessante ed esaustivo e, al solito, ricco di spunti di riflessone.

Segue Claudio Beccari, con la sua guida autorevole, a spiegarci come e quando usare le parentesi graffe nella loro multiformità.

Matteo Centonza e Vito Piserchia ci portano fin dentro realtà lavorative in cui LATEX ha un ruolo centrale. Ci illustrano inoltre `illumino`, il loro sistema di produzione XML costruito intorno a TEX.

Luigi Scarso ci erudisce su come ConTEXt-`mkiv` riesce a essere compatibile con lo standard di archiviazione di documenti ISO 19500-1 livello A, meglio noto come PDF/A 1a.

Gianluca Pignalberi ci mostra come ha usato beamer per ottenere una presentazione autopresentante. Un po' di parlato, qualche effetto di transizione e un temporizzatore e la "magia" è fatta.

Infine Jean-Michel Hufflen ci parla di un pacchetto (sviluppato dal suo gruppo) indispensabile per quei corsi di insegnamento sia frontale che a distanza, spiegando problematiche, esigenze e voncoli dei tre.

La parola passa ora a Massimiliano Dominici per il suo discorso di commiato. Da parte mia ringrazio tutti, e vi auguro un lungo e felice TEXing. Alla prossima.

*Il futuro del* GUIT

Queste poche righe sono l'ultimo editoriale che firmo in qualità di presidente del GUIT. Il mio mandato scade in concomitanza con il convegno annuale e ho deciso di non ricandidarmi né per un posto nel consiglio direttivo, né, di conseguenza, per la presidenza. Avrei ritenuto un motivo sufficiente per questa decisione il fallimento di EuroTEX 2010, ma in realtà la decisione era maturata già nei mesi precedenti, quando mi ero reso conto che cominciavo ad avvertire l'impegno come un peso. Continuare in queste condizioni sarebbe faticoso per me e dannoso per il GUIT allo stesso tempo. Ho deciso, quindi, di lasciare il passo a chi, in questo momento, ha più entusiasmo e voglia di fare di me. Naturalmente non abbandonerò il GUIT e continuerò a dare il mio contributo in termini di discussioni sulla mailing list, e di tempo dedicato alle attività del gruppo come redattore della rivista e amministratore del Forum. Ma non più in posizioni che prevedano responsabilità decisionali.

Il rinnovamento nel gruppo dirigente del GUIT sarà comunque ancora più esteso. Per una serie di circostanze fortuite scade il mandato di ben sette membri del Consiglio Direttivo e, quando leggerete queste note, saranno già stati eletti i nuovi membri, che dovranno poi provvedere all'elezione di presidente, vicepresidente e tesoriere.

Il nuovo gruppo dirigente si troverà a dover risolvere una serie di questioni attualmente in discussione sulla mailing list generale degli iscritti e che riguardano la struttura amministrativa del gruppo. In questi anni si è avuta una commistione, a mio avviso poco naturale, di compiti "di indirizzo" e di altri più propriamente "burocratici" nelle attività di presidente, vicepresidente e tesoriere. In pratica ci siamo occupati di questioni (ordini e spedizioni del materiale, monitoraggio delle iscrizioni, della posta del GUIT, ecc.) che sarebbero più propriamente di competenza di una segreteria, a scapito anche della parte di indirizzo delle attività del gruppo.

Questa scelta è un retaggio funzionale alla prima organizzazione dell'associazione, in cui coloro che gestivano GUIT erano anche quelli che svolgevano tutte le attività (manutenzione del portale, spedizione buste, gestione quote, ecc.). Adesso questo modello non è più proponibile, sia perché il numero di soci (e conseguentemente delle attività) è cresciuto, sia perché in base allo statuto vigente i membri del consiglio direttivo sono scelti per le loro competenze su LATEX ma (come è giusto che sia) non per la manutenzione di siti web e/o di gestione della segreteria.

Inoltre a gestire queste attività è stato finora sostanzialmente un ristretto numero di membri con

base a Pisa. Questo stato di cose ha garantito un periodo di crescita all'associazione, ma le risorse che un gruppo locale può mettere in campo sono limitate e in questo momento in via di esaurimento.

La discussione in atto tra i membri, quindi, ha principalmente il duplice scopo di svincolare il gruppo dirigente da compiti strettamente amministrativi e di rendere l'associazione indipendente da una qualunque collocazione geografica. Questi non sono però gli unici due temi discussi. Il dibattito è anche l'occasione per lanciare nuove idee e riproporne altre già presentate in passato. Stilo qui un elenco incompleto dei temi trattati:

- idee per ottenere un maggiore coinvolgimento degli iscritti alle attività del gruppo;
- migrazione del sito e sua completa ristrutturazione;
- possibilità di attivare sul sito strumenti più avanzati di quelli attuali, come una *wiki* o *repositories* per progetti collaborativi sponsorizzati dal GUIT;
- ruolo dell'attuale mailing list e creazione di una mailing list pubblica dedicata ad argomenti "TEXnici".

Insomma, la carne al fuoco è molta; scopriremo in futuro come è venuto l'arrosto.

Un paio di osservazioni, infine, sul fallimento di EuroTEX 2010. A mio avviso i principali fattori del cattivo esito sono stati due. Il primo è stato il forte ritardo con cui ho dato il via all'organizzazione dell'evento e, probabilmente, una pubblicizzazione non sufficiente dello stesso. Il secondo è la sottovalutazione delle potenzialità attrattive del ConTEXt Meeting, insieme a una scarsa coordinazione con gli altri gruppi europei. Se fino a quattro anni fa, quando venne accettata la nostra candidatura all'organizzazione di EuroTEX, questo evento non aveva rivali, nel corso degli anni il ConTEXt meeting si è attestato come alternativa di indubbio valore, potendo vantare la presenza dei maggiori sviluppatori del mondo TEX. La sua forza è stata in parte mascherata dal fatto che nel 2009 era abbinato a EuroTEX, mentre l'anno precedente lo stesso EuroTEX si svolgeva in concomitanza con il convegno TUG. I maggiori eventi internazionali legati a TEX sono quindi stati solo due, in quegli anni, e non tre come nel 2010 dove si è vista la reale capacità di attrazione del ConTEXt meeting. Una maggiore capacità di prevedere questo scenario e un'integrazione migliore con gli altri gruppi europei avrebbero probabilmente evitato l'esito negativo.

▷ Gianluca Pignalberi
`g dot pignalberi at alice dot it`

▷ Massimiliano Dominici
`mlgdominici at interfree dot it`

# Installare TEX Live 2010 su Ubuntu

*Enrico Gregorio*


## Sommario

È possibile far convivere la distribuzione TEX Live 2010 con la versione Ubuntu del sistema operativo GNU/Linux? Sì, ecco come si può fare.

## Abstract

Is it possible to have TEX Live 2010 along with the Ubuntu version of the GNU/Linux operating system? Yes, here's how it may be done.


## 1 Introduzione

Uno dei difetti principali della distribuzione TEX Live su sistemi UBUNTU è che, per precisa scelta degli sviluppatori, manca il gestore di aggiornamento e manutenzione tlmgr. Da un lato questo difende l'utente da possibili danni al sistema, dall'altro impedisce un costante aggiornamento della distribuzione TEX per ovviare a *bug* o per avere a disposizione nuove funzioni che escono ogni giorno.

È naturalmente possibile installare la distribuzione *normale* anche su sistemi che prevedano la gestione delle applicazioni tramite un programma dedicato, nel caso di Ubuntu è Synaptic o, dalla linea di comando, apt-get e simili.

La procedura che descriviamo, con opportune modifiche, può essere adattata anche ad altre distribuzioni come Fedora o simili. Per tutte quelle basate su Debian dovrebbe essere del tutto identica.

## 2 Breve introduzione al terminale

Tutto ciò che segue richiede una certa pratica con il terminale, cioè l'interfaccia per la linea di comando. Chi non ha idea di che cosa sia il terminale, lasci perdere; ma non è poi così difficile copiare i comandi così come sono scritti. Si può trovare una guida iniziale all'indirizzo

```
http://wiki.ubuntu-it.org/
AmministrazioneSistema/RigaDiComando
```

Nel seguito, una riga come

```
$ ls -l
```

indica un comando da dare sul terminale, che va inviato con l'apposito tasto di invio (quello per andare a capo, per capirsi). Il simbolo `$` rappresenta il sistema che attende un comando, non va copiato. Nel terminale effettivo può essere diverso, per esempio qualcosa come

```
enrico@ubuntu:~$
```

e, normalmente, dopo questi caratteri iniziali c'è un rettangolo lampeggiante. Si copino i comandi dal segno di `$` escluso in poi. In certe situazioni i comandi sono troppo lunghi per stare su una riga di questo documento e saranno qui resi con

```
$ comando a b c \
  d e f
```

La barra rovescia indica dunque che il comando prosegue sulla riga di stampa successiva. Gli spazi prima della barra rovescia sono significativi.

Eventuali risposte del sistema saranno rappresentate senza il simbolo `$`, per esempio

```
bash: tix: command not found
```

dice che il sistema ha ricevuto il comando di eseguire il programma tix, che però non esiste. Il prefisso `bash:` indica chi sta cercando di eseguire i comandi, in questo caso la *shell*, ignorate questi dettagli.

Quasi sempre non è necessario copiare del tutto le varie componenti di una riga di comando: si preme il tasto di tabulazione e, se il completamento della parte che si sta scrivendo è unico, il terminale provvederà da sé a farlo.

Ultimi avvisi: se la vostra tastiera non ha `~`, trovate il modo di inserire questo carattere (e procuratevi al più presto una tastiera internazionale); su parecchie tastiere, con la configurazione normale di Ubuntu per l'italiano, il simbolo `~` si ottiene con la combinazione AltGr + ì. Le parti di testo in corpo ridotto sono riservate agli utenti più esperti.

## 3 Preliminari

Con Synaptic installate i moduli Perl-Tk e Perl-doc. Poi aprite una sessione di terminale e preparatevi una cartella di lavoro, per esempio

```
$ mkdir ~/texlive-install
$ cd ~/texlive-install
```

Il secondo comando vi fa entrare nella cartella creata con il primo.

Ora descriveremo brevemente i due modi principali per procurarsi la distribuzione TEX Live, il primo esclusivamente via rete, il secondo anche *offline*.

## 4 Procurarsi la distribuzione (1)

Il modo più semplice di installare la TEX Live è via rete. Si scarica

```
http://mirror.ctan.org/systems/texlive/
tlnet/install-tl-unx.tar.gz
```

per esempio tramite wget, curl oppure un browser. L'ultimo modo non richiede commenti, se non che il file scaricato va trasferito nella cartella di lavoro; per il primo il comando è

```
$ wget http://mirror.ctan.org/systems/\
  texlive/tlnet/install-tl-unx.tar.gz
```

mentre per curl si dovrà scrivere

```
$ curl -O http://mirror.ctan.org/systems/\
  texlive/tlnet/install-tl-unx.tar.gz
```

Qui c'è subito un problema. Il sistema di *mirror*, cioè la rete di macchine che si distribuiscono gli accessi agli archivi della TEX Live, ha un nodo in Italia che è tremendamente lento. Ci sono alternative più veloci: sperimentate con la vostra connessione quale sia più rapida scegliendo una delle seguenti sostituzioni

```
mirror.ctan.org
      ↓
mirror.switch.ch/ftp/mirror/tex/
bo.mirror.garr.it/mirrors/CTAN/
```

Nel seguito useremo mirror.ctan.org, ricordatevi la sostituzione scelta.

A questo punto dovete decomprimere il file scaricato. Il comando è

```
$ tar zxf install-tl-unx.tar.gz
```

che produrrà una nuova cartella, nella quale ci sposteremo:

```
$ cd install-tl-20100914
```

La parte finale del nome è la data in cui il programma di installazione è stato prodotto, quindi può essere diversa e cade a fagiolo la possibilità di completare con il tasto di tabulazione. Passate alla sezione 6.

## 5 Procurarsi la distribuzione (2)

Se non avete un collegamento di rete efficiente, potete scaricare un'immagine ISO della distribuzione, cioè un file che equivale a un DVD, oppure procurarvi il DVD fisico. L'indirizzo Web a cui rivolgersi è uno fra i due seguenti

```
http://mirror.switch.ch/ftp/mirror/tex/
systems/texlive/Images/texlive2010.iso
```

```
http://bo.mirror.garr.it/mirrors/CTAN/
systems/texlive/Images/texlive2010.iso
```

Si tratta di un file di 1.9 GiB, trasferitelo su una chiave e copiatelo sulla macchina dove desiderate installare TEX Live; oppure trasformate l'immagine in un DVD che inserirete nella macchina.

Si può ricorrere anche a un 'torrent' scegliendo il relativo collegamento alla pagina `http://www.tug.org/texlive/acquire-iso.html`; dovrebbe partire automaticamente il programma Transmission che scaricherà l'immagine del DVD.

Un doppio clic sul file (o sul DVD) dovrebbe permettere di accedere al disco virtuale (o fisico). Aprite una sessione di terminale, create una cartella di lavoro con

```
$ mkdir ~/texlive-install
```

e spostatevi nella cartella `texlive` del disco virtuale (o fisico). Questa manovra dovrebbe essere

```
$ cd /cdrom/TeXLive/texlive
```

ma al posto di "`TeXLive`" potrebbe esserci qualcos'altro; usate il completamento automatico. Passate alla sezione 6.

## 6 Installare la distribuzione

Ora dovrete dare il comando di installazione:

```
$ sudo ./install-tl -gui -lang it \
  -repository http://mirror.ctan.org/\
  systems/texlive/tlnet
```

(ricordate di sostituire mirror.ctan.org). Il sistema chiederà la password di amministratore e comparirà una finestra simile a quella che vedete nella figura 1. Se omettete "`-lang it`", le scritte saranno in inglese.

In basso, al centro, c'è il pulsante Installa TeX Live. Premetelo e attendete fiduciosi che l'installazione sia completa. In realtà sarebbe possibile personalizzare l'installazione in vari modi, ma quello proposto è sicuro e completo.

*Non modificate* la scelta standard per "Crea i collegamenti nelle directory di sistema": accanto *deve* comparire "No".

Quando l'installazione è terminata, passate alla sezione 8.

## 7 Se qualcosa va storto

Nel caso l'installazione non vada a buon fine, prima di riprovare occorre eliminare quanto eventualmente scritto sul proprio sistema:

```
$ sudo rm -rf /usr/local/texlive/2010
$ sudo rm -rf ~/.texlive2010
```

Poi si riprovi.

## 8 Perfezionare l'installazione

Ora viene la parte difficile, cioè far capire al sistema dove trovare i programmi della distribuzione

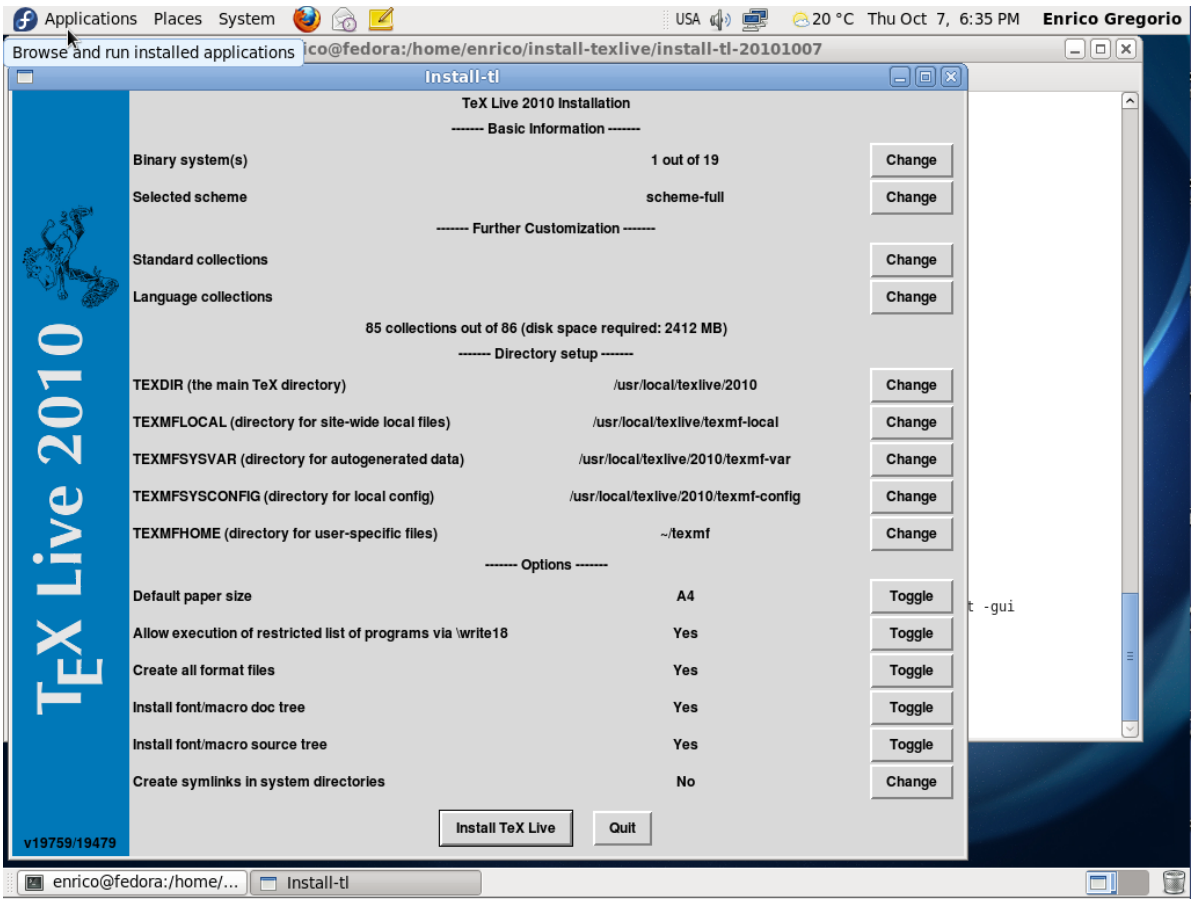


Figura 1: Finestra di installazione

TEX. Si consiglia di non toccare quella standard su Ubuntu, in modo da non aver problemi a installare programmi che da essa dipendono, come Kile.

Torniamo nella nostra cartella di lavoro con

```
$ cd ~/texlive-install
```

e affrontiamo il sistema operativo. Diamo i comandi misteriosi

```
$ echo 'export PATH=/opt/texbin:${PATH}' \
  > texlive.sh
$ sudo cp texlive.sh /etc/profile.d/
$ sudo mkdir -p /opt
```

Questo crea un file `texlive.sh` contenente il testo che abbiamo scritto fra apici singoli e lo copia in una delle directory di sistema. Ora il passo decisivo, che richiede una scelta dipendente dall'architettura hardware della propria macchina; si dovrà dare uno (e uno solo) dei comandi qui riportati

```
$ sudo ln -s /usr/local/texlive/2010/bin/\
  i386-linux /opt/texbin
$ sudo ln -s /usr/local/texlive/2010/bin/\
  x86_64-linux /opt/texbin
$ sudo ln -s /usr/local/texlive/2010/bin/\
  powerpc-linux /opt/texbin
```

Non è possibile per l'autore delle note sapere quale dei tre: solo voi potete stabilire se la vostra macchina è basata su un processore Intel (o AMD) a 32 bit, su un Intel a 64 bit o su un PowerPC. Potete però scoprirlo con il comando

```
$ ls /usr/local/texlive/2010/bin
```

che darà come risposta una delle tre possibilità.

L'utente smaliziato si potrebbe chiedere perché non inserire direttamente in `texlive.sh` il nome della directory con gli eseguibili, per esempio

```
export PATH=/usr/local/texlive/2010/\
  bin/i386-linux:${PATH}
```

L'idea è che quando uscirà la 2011 sarà sufficiente dare il comando

```
$ sudo ln -s /usr/local/texlive/2011/bin/\
  i386-linux /opt/texbin
```

dopo l'installazione, *senza toccare nient'altro e senza nemmeno un logout.*

A questo punto *si deve eseguire il logout*, perché il sistema deve digerire la modifica. Rifatto il *login*, riapriamo una sessione del terminale e controlliamo che tutto sia a posto; il comando

```
$ which tex
```

dovrebbe dare come risposta

```
/opt/texbin/tex
```

Se così è, siamo a cavallo e possiamo procedere con l'aggiornamento della distribuzione, altrimenti

cercare di capire che cos'è andato storto con l'aiuto di un guru.

Ci sono due modi per mettersi in condizione di usare tlmgr, consiglio di usare entrambi. Il primo farà partire tlmgr dal terminale. Si dia il comando

```
$ gedit ~/.bashrc
```

e nella finestra che compare si aggiunga, in fondo,

```
...

# Addition for TeX Live
function sutlmgr () {
  if [[ -z "$@" ]]
  then
    sudo /opt/texbin/tlmgr -gui
  else
    sudo /opt/texbin/tlmgr "$@"
  fi
}
alias mktexlsr=\
  'sudo /opt/texbin/mktexlsr'
alias updmap-sys=\
  'sudo /opt/texbin/updmap-sys'
alias fmtutil-sys=\
  'sudo /opt/texbin/fmtutil-sys'
```

I tre puntini rappresentano ciò che sta già nel file `.bashrc` e che non va modificato.

Chi lo preferisce (probabilmente perché ha già aggiunto altri *alias* al suo ambiente personale) può inserire quella modifica nel file `.bash_aliases`.

Si registri la modifica con l'apposito menù e al terminale si scriva

```
$ . ~/.bashrc
$ sutlmgr
```

Apparirà la finestra di tlmgr che non richiede particolari commenti (si legga la documentazione). D'ora in poi il comando sutlmgr farà partire tlmgr con privilegi di amministratore in interfaccia grafica. Un comando come

```
$ sutlmgr show --list xyz
```

eseguirà direttamente l'azione richiesta come argomento di tlmgr.

In alternativa si può creare una piccola applicazione sulla scrivania. Puntando sulla scrivania, si prema il tasto destro del mouse e si scelga "Create Launcher ...". Apparirà una finestra con alcuni spazi dove scrivere. Nello spazio 'Name' scrivere per esempio "TeX Live Manager" e nello spazio 'Command' scrivere (è una sola riga, sia chiaro)

```
gksu -d -S -D "TeXLive Manager"
  '/opt/texbin/tlmgr -gui'
```

Creata la piccola applicazione, potremo fare doppio clic e dopo la richiesta della password comparirà la finestra di tlmgr.

Ci manca ancora una cosa: rendere noti al sistema i font OpenType forniti con TEX Live, in particolare per poterli usare con XETEX e XELATEX che si appoggiano sulle librerie FreeType:

```
$ sudo cp \
  $(kpsewhich -var-value TEXMFSYSVAR)\
  /fonts/conf/texlive-fontconfig.conf \
  /etc/fonts/conf.d/09-texlive.conf
$ sudo fc-cache -fsv
```

I comandi per la procedura via rete, uno dopo l'altro in una tipica installazione di Ubuntu 10, sono riportati nella tabella finale 2; in rosso le parti che potrebbero (o dovrebbero) essere diverse; le righe con la sottolineatura rappresentano risposte del terminale, quella in rosso dice che cosa sostituire al posto di `i386-linux` nella riga successiva; le righe con azioni descritte tra parentesi ad angolo descrivono manovre che si devono fare fuori dal terminale.

## 9 SUSELinux

La procedura descritta funziona perfettamente con SUSELinux, purché si sia installato il modulo per Perl-Tk. Non occorre definire la funzione per accedere a tlmgr. Occorre invece ricordarsi di usare l'opzione `-E` a sudo. Quindi

```
$ sudo -E tlmgr -gui
$ sudo -E updmap-sys
$ sudo -E mktexlsr
```

La procedura per crearsi un'applicazione che lanci tlmgr funziona allo stesso modo.

## 10 Fedora

La procedura va bene anche per Fedora, almeno per la versione 13. Per l'installazione con l'interfaccia grafica è necessario installare Perl-Tk che però non fa parte della dotazione standard. Lo si può recuperare da

```
http://koji.fedoraproject.org/koji/
buildinfo?buildID=151517
```

Il file `/etc/profile.d/texlive.sh` deve essere diverso e contenere il testo che si trova nella tabella 1.

Altra differenza è che in Fedora si lavora come `root` per l'amministrazione, quindi i comandi che vengono indicati per Ubuntu con il prefisso sudo vanno eseguiti dopo un comando iniziale su (e senza sudo, ovviamente).

# Appendici

## A Installare un pacchetto personale

Supponiamo di aver bisogno di un pacchetto LATEX che non è nella TEX Live; può succedere per questioni di licenza o perché è una versione sperimentale non ancora negli archivi ufficiali. Ci sono due posti fra cui scegliere dove sistemare i file necessari. Prima di tutto scarichiamo l'archivio da dove è

Tabella 1: Il file `texlive.sh` per Fedora 13.

```
#!/bin/bash
pathmunge () {
  if ! echo $PATH | /bin/egrep -q "(^|:)$1($|:)" ; then
    if [ "$2" = "after" ] ; then
      PATH=$PATH:$1
    else
      PATH=$1:$PATH
   fi
  fi
}
pathmunge /opt/texbin
unset pathmunge
```

ospitato (CTAN o altro sito) e decomprimiamolo in una cartella di lavoro. Per fissare le idee, il pacchetto sarà `padua` e la cartella conterrà i file `README`, `padua.ins`, `padua.dtx` e `padua.pdf` (le parti scritte in rosso saranno quelle da sostituire con il nome effettivo). Apriamo una sessione di terminale e diamo il seguente comando:

```
$ tex padua.ins
```

Può capitare che questo file con estensione `.ins` non ci sia; in tal caso il comando

```
$ tex padua.dtx
```

sarà quello necessario. In entrambi i casi saranno generati alcuni file che dovremo sistemare nel posto corretto. Prima di passare all'installazione, cancelleremo il file `padua.log`:

```
$ rm padua.log
```

Naturalmente queste istruzioni sono generiche; ci sono pacchetti con struttura più complessa e andranno seguite le istruzioni degli sviluppatori, ma i passi sono analoghi.

Ora decidiamo se servirci dell'albero personale o di quello 'locale'. La differenza fondamentale è che nel secondo caso il pacchetto sarà disponibile a tutti gli utenti della macchina; ovviamente sarà necessario essere amministratori per poterlo fare.

### A.1 Installare nell'albero personale

L'albero personale ha la sua radice in `~/texmf` (su sistemi GNU/Linux), ma non occorre sapere dove sia di preciso e le istruzioni che seguono valgono in realtà per qualsiasi sistema Unix. Proseguiamo da dove c'eravamo interrotti; per prima cosa definiamo un'abbreviazione che ci risparmia lavoro, poi creiamo le cartelle necessarie.

```
$ Local=$(kpsewhich -var-value TEXMFHOME)
$ mkdir -p $Local/source/latex/padua
$ cp README padua.ins padua.dtx \
  $Local/source/latex/padua
$ mkdir -p $Local/doc/latex/padua
$ cp padua.pdf $Local/doc/latex/padua
$ mkdir -p $Local/tex/latex/padua
$ cp * $Local/tex/latex/padua
```

### A.2 Installare nell'albero locale

L'albero locale ha la sua radice in `/usr/local/texlive/texmf-local` (su sistemi GNU/Linux), ma non occorre sapere dove sia di preciso e le istruzioni che seguono valgono in realtà per qualsiasi sistema Unix. Proseguiamo da dove c'eravamo interrotti; per prima cosa definiamo un'abbreviazione che ci risparmia lavoro, poi creiamo le cartelle necessarie.

```
$ Local=$(kpsewhich -var-value TEXMFLOCAL)
$ sudo mkdir -p $Local/source/latex/padua
$ sudo cp README padua.ins padua.dtx \
  $Local/source/latex/padua
$ sudo mkdir -p $Local/doc/latex/padua
$ sudo cp padua.pdf $Local/doc/latex/padua
$ sudo mkdir -p $Local/tex/latex/padua
$ sudo cp * $Local/tex/latex/padua
$ mktexlsr
```

## B Installare una famiglia di font

Ci sono varie istruzioni su come installare nuovi font che siano stati acquistati o, se gratuiti, non abbiano una licenza che ne permette l'inclusione nella TEX Live. L'installazione di font nell'albero personale è sconsigliata, perché richiede un costante lavoro dell'utente nel caso in cui gli aggiornamenti a TEX Live contengano anche quelli ai font distribuiti.

È opportuno seguire le istruzioni contenute nell'opuscolo "The font installation guide" di Philipp Lehman, disponibili nella TEX Live con il comando da terminale

```
$ texdoc fontinstallationguide
```

Queste istruzioni, volutamente, terminano allo stadio della preparazione dei file necessari. Facciamo l'ipotesi che la famiglia di font si chiami 'Padua', con nome di famiglia `zpd`. In seguito le parti da sostituire con i nomi effettivi saranno in rosso. La procedura descritta da Lehman crea un certo numero di file nella cartella di lavoro, con varie estensioni:

```
.tfm .vf .pfb .afm .map .sty .fd
```

che andranno inserite al posto giusto nella gerarchia del sistema TEX. Il posto giusto è il cosiddetto *albero locale* che, nella TEX Live, ha la sua radice in `/usr/local/texlive/texmf-local`. In realtà non è necessario saperlo, perché il sistema è capace di conoscere sé stesso. Prima di tutto costruiamo le cartelle necessarie, definendo una variabile che ci risparmi lavoro:

```
$ Local=$(kpsewhich -var-value TEXMFLOCAL)
$ sudo mkdir -p \
  $Local/fonts/{afm,tfm,type1,vf}/padua
$ sudo cp zpd*.afm $Local/fonts/afm/padua
$ sudo cp zpd*.tfm $Local/fonts/tfm/padua
$ sudo cp \
  zpd*.pfb $Local/fonts/type1/padua
$ sudo cp zpd*.vf $Local/fonts/vf/padua
$ sudo mkdir -p $Local/tex/latex/padua
$ sudo cp *.sty *.fd $Local/tex/latex/padua
$ sudo mkdir -p \
  $Local/fonts/map/dvips/padua
$ sudo cp padua.map \
  $Local/fonts/map/dvips/padua
$ mktexlsr
```

Così abbiamo terminato di sistemare i mobili. Ora dobbiamo fornire al sistema TEX la chiave di accesso. Ci sono due casi: o è la prima volta che si aggiunge una famiglia di font oppure l'abbiamo già fatto seguendo questa stessa procedura.

Nel primo caso dobbiamo generare un nuovo file e inserirlo al posto giusto:

```
$ echo "Map padua.map" > updmap-local.cfg
$ mkdir -p $Local/web2c
$ sudo mv updmap-local.cfg $Local/web2c
$ sutlmgr generate --rebuild-sys updmap
```

Nel secondo caso dobbiamo solo aggiungere una riga al file già esistente:

```
$ cp $Local/web2c/updmap-local.cfg .
$ echo "Map padua.map" >> updmap-local.cfg
$ sudo mv updmap-local.cfg $Local/web2c
$ sutlmgr generate --rebuild-sys updmap
```

L'ultima azione, così come la chiamata di mktexlsr, può essere eseguita dall'interfaccia grafica di tlmgr. Eseguendola possiamo stare certi che la chiave di accesso non sarà persa con gli aggiornamenti di TEX Live.

Se per caso abbiamo anche a disposizione le versioni OpenType dei nostri font, si aggiunga anche la coppia di righe

```
$ sudo mkdir -p $Local/fonts/opentype/padua
$ sudo cp *.otf $Local/fonts/opentype/padua
```

a quelle analoghe viste prima.

▷ Enrico Gregorio
Dipartimento di Informatica
Università di Verona
enrico dot gregorio at univr dot it

Tabella 2: L'intera procedura per Ubuntu

⟨ *Installare* Perl-Tk *con* Synaptic ⟩
⟨ *Avviare una sessione di terminale* ⟩
```
$ mkdir ~/texlive-install
$ cd ~/texlive-install
$ wget http://mirror.ctan.org/systems/texlive/tlnet/install-tl-unx.tar.gz
$ tar zxf install-tl-unx.tar.gz
$ cd install-tl-20100914
$ sudo ./install-tl -gui -lang it \
  -repository http://mirror.ctan.org/systems/texlive/tlnet
```
⟨ *Premere "Installa TeX Live"* ⟩
⟨ *Attendere che l'installazione finisca; bersi un caffè, forse due* ⟩
⟨ *Premere "Fine"* ⟩
```
$ cd ~/texlive-install
$ echo 'export PATH=/opt/texbin:${PATH}' > texlive.sh
$ sudo cp texlive.sh /etc/profile.d/
$ sudo mkdir -p /opt
$ ls /usr/local/texlive/2010/bin
  i386-linux
$ sudo ln -s /usr/local/texlive/2010/bin/i386-linux /opt/texbin
```
⟨ *Eseguire il logout* ⟩
⟨ *Dopo il login, aprire un terminale* ⟩
```
$ which tex
  /opt/texbin/tex
```
⟨ *Se la risposta non è quella, gridare forte 'Aiuto'* ⟩
```
$ gedit ~/.bashrc
```
⟨ *Aggiungere in coda al file* ⟩
```
    # Additions for TeX Live
    function sutlmgr () {
      if [[ -z "$@" ]]
      then
        sudo /opt/texbin/tlmgr -gui
      else
        sudo /opt/texbin/tlmgr "$@"
      fi
    }
    alias mktexlsr='sudo /opt/texbin/mktexlsr'
    alias updmap-sys='sudo /opt/texbin/updmap-sys'
    alias fmtutil-sys='sudo /opt/texbin/fmtutil-sys'
```
⟨ *Registrare e uscire da* gedit ⟩
```
$ sudo cp $(kpsewhich -var-value TEXMFSYSVAR)/fonts/conf/texlive-fontconfig.conf \
  /etc/fonts/conf.d/09-texlive.conf
$ sudo fc-cache -fsv
```
⟨ *Rilassarsi e godersi la TEX Live 2010* ⟩

Note.
(1) Sostituire "`mirror.ctan.org`" con uno tra "`mirror.switch.ch/ftp/mirror/tex`" e "`bo.mirror.garr.it/mirrors/CTAN`" scegliendo il sito più veloce.
(2) La data `20100914` è indicativa, potrebbe essere diversa.
(3) `i386-linux` corrisponde a una delle possibili architetture, potrebbe essere `x86_64-linux` o, meno probabilmente, `powerpc-linux`.

# Le graffe: queste sconosciute

*Claudio Beccari*

## Sommario

Le parentesi graffe, quadre e tonde sono usate nella sintassi dei comandi LaTeX molto frequentemente, le graffe molto più delle tonde. Tuttavia mentre le parentesi quadre e quelle tonde hanno degli usi molto ristretti, quelle graffe hanno almeno tre interpretazioni diverse e dispongono anche di nomi alternativi. Talvolta si confondono i significati delle graffe o non è chiarissima la loro funzione. Questo tutorial vorrebbe fare chiarezza in questo campo.

## Abstract

Braces, brackets and common (round) parentheses play an important rôle in LaTeX syntax; braces in particular are used more often than the other kinds. Nevertheless while brackets and round parentheses play restricted rôles, braces have at least three different interpretations and even some alias names are given to them. Sometimes their rôles are misinterpreted by the user or sometimes their rôles are not so evident. This tutorial is intended to clarify the braces' rôles.

## 1 Introduzione

Chiunque legga il file sorgente di un qualunque documento LaTeX ne trova alcune parti fitte di parentesi, quasi sempre graffe, talvolta quadre; all'interno dell'ambiente picture si incontrano anche le parentesi tonde. Queste ultime vengono usate solo per delimitare le coordinate dei punti o i coefficienti angolari delle linee oblique nella sintassi di quell'ambiente. Praticamente è difficile trovare altri punti di LaTeX o pacchetti di estensione nei quali si siano usate le parentesi tonde; ovviamente le si incontra spesso nel testo, ma questa loro funzione non ha nulla a che fare con la sintassi di LaTeX.

Le parentesi quadre si incontrano più sovente delle parentesi tonde; esse, nella sintassi di LaTeX, servono per racchiudere le liste delle opzioni per le classi, per molti pacchetti e per alcuni comandi.

Le funzioni delle parentesi graffe sono generalmente descritte come quelle di delimitazione dei gruppi, ma non è proprio così, è solo una mezza verità.

Si consideri la seguente lista di funzioni:

1. {⟨*testo*⟩}
2. ⟨*macro*⟩{⟨*testo*⟩}
3. ⟨*comando*⟩{⟨*testo*⟩}

In questa lista ⟨*testo*⟩ può consistere di qualunque cosa: testo vero e proprio, sequenze di controllo o comandi generici, scatole o qualunque oggetto che abbia senso nel contesto racchiuso dalle graffe.

Il caso 1 rappresenta un gruppo; eventuali definizioni o assegnazioni eseguite dentro il gruppo in modalità non globale[1] perdono la loro efficacia non appena l'interprete dei comandi (tex, ma più sovente pdftex) ha esaurito di elaborare quanto è contenuto nel gruppo e ne esce.

Il caso 2 è forse il caso più diffuso; serve per passare argomenti *non delimitati*[2] a macro definite sia dall'utente stesso sia nel nucleo di LaTeX o nei vari pacchetti. Come è noto gli argomenti possono essere al massimo nove. Gli argomenti non delimitati *devono* essere racchiusi fra graffe se sono composti da più *token* (oggetti), tipicamente da un testo vero e proprio composto da più lettere. Se l'argomento è composto da un solo token le graffe possono essere omesse. In questo senso le graffe sembrano svolgere anche l'azione di raggruppamento visto nel caso 1, ma non è così: quanto viene passato come argomento ad una macro viene sviluppato nel momento in cui la macro viene espansa e se l'argomento conteneva delle assegnazioni o delle definizioni queste possono essere "globali" anche se non sono esplicitamente definite come tali. In ogni caso bisogna rendersi conto che raggruppare dei token per passarli come un unico argomento ad una macro non implica affatto le funzioni del *gruppo* come definito al punto 1.

Il caso 3 con la sintassi LaTeX si presenta assai raramente in modo esplicito, ma è lì in "agguato" pronto per essere utilissimo o, al contrario, per porre in difficoltà l'utente LaTeX che cerca di definire delle macro, ma si trova impacciato da un comportamento che non si aspetta. In realtà questa sintassi è esplicita per molti comandi TeX "primitivi", quelli cioè che non sono definiti come macro, ma appartengono a quei circa 300 comandi del nucleo dell'interprete. Sono quelli in termini dei quali direttamente o indirettamente sono definite tutte le macro.

Questo ultimo caso 3 accetta che le graffe siano sostituite da sinonimi, mentre negli altri casi i

1. Per il significato di comando globale, di assegnazione globale, di definizione globale si vede il manuale di Knuth (1996). Alternativamente si veda la documentazione di Gregorio (2009).
2. Per i comandi delimitati, oltre che nel testo di Knuth (1996), si può anche vederne una applicazione nell'articolo di Beccari (2008b).

sinonimi non sono accettabili o lo sono in certe circostanze come si vedrà più avanti; spesso si ha la sensazione che i comandi primitivi vogliano gli oggetti su cui operare raccolti come nel caso 2 o formino un gruppo come nel caso 1, ma in realtà non è così sia perché loro o i loro sinonimi sono sempre obbligatori, anche nel caso di un solo oggetto, sia perché la funzione di limitare la validità delle definizioni o delle assegnazioni può essere assente.

## 2 I sinonimi delle graffe

Per vari motivi le graffe possono essere indicate con i loro sinonimi; il nucleo di LaTeX esegue le associazioni:

```
\let\bgroup {
\let\egroup }
```

Il comando `\let` è un comando primitivo che serve per associare un token ad un altro token[3]; questi token associati prendono anche il nome di *token impliciti* ma la terminologia non è così importante; il fatto che il token associato, detto gergalmente *alias* del token vero, ne conserva tutte le caratteristiche e spesso può essere usato in alternativa al token vero. Non è sempre vero, però: quando l'interprete legge dal file di ingresso i singoli caratteri che lo costituiscono, esegue la *tokenizzazione*; generalmente ogni segno viene definito come un token che rappresenta il segno stesso corredato da un codice di categoria; così avviene per le lettere dell'alfabeto, per le cifre, per i segni di interpunzione, per le varie parentesi, per lo spazio, per i caratteri proibiti, per i caratteri che l'interprete sa essere attivi perché dichiarati tali prima di leggere il file sorgente, eccetera. Quando l'interprete incontra una *contol sequence*, una fila di lettere precedute dal backslash o un solo carattere diverso da una lettera preceduto dal backslash, allora forma un solo token formato dal nome della control sequence a cui associa il codice di categoria specifico dei comandi o delle macro.

In questo modo `\bgroup` e `\egroup` hanno lo stesso codice e la stessa categoria rispettivamente della graffa aperta e di quella chiusa, ma restano ancora un pochino diversi. Infatti quando l'interprete legge il file di ingresso esso tiene conto delle graffe aperte e di quelle chiuse in modo da assicurare che esse siano correttamente bilanciate. In questo caso `\bgroup` e `\egroup` non partecipano al conteggio delle graffe aperte e chiuse e quindi possono apparire anche scompagnate all'interno di un altra coppia di graffe esplicite correttamente accoppiate. Vedremo più avanti quanto sia utile questa caratteristica.

3. Per il significato di token, oltre al testo di Knuth (1996) si può vere anche l'articolo di Beccari (2008a).

## 3 I gruppi

Una espressione racchiusa fra graffe, senza che queste servano per passare dei token raggruppati a una macro o a un comando primitivo, come si è detto, forma un gruppo. I gruppi possono anche essere racchiusi fra

`\begingroup` ... `\endgroup`

oppure fra

`\bgroup` ... `\egroup`

ma questa seconda scrittura appare barocca, lunga da scrivere e, se non ci sono i motivi particolari che vedremo più avanti, sarebbe meglio evitarla.

Dentro il gruppo, senza che l'utente debba preoccuparsi di questi dettagli, viene costituito uno *stack* nel quale vengono memorizzati tutti i valori dei registri, delle *control sequence* e dei caratteri attivi esistenti via via che si attribuisce loro un nuovo significato; chiudendo il gruppo, lo stack viene usato per ripristinare i valori precedenti alle "variabili" che sono state modificate dentro il gruppo. Questo avviene sempre quando le ridefinizioni o le assegnazioni non sono globali; se una assegnazione o una definizione è globale, il valore precedente della variabile viene perso perché, appunto, l'operazione è globale e nulla viene immesso nello stack. L'uso dei gruppi serve proprio per ottenere questa funzionalità.

Merita segnalare che la coppia `\begingroup` e `\endgroup` mantiene l'interprete nella stessa modalità operativa, modo testo, modo matematico, modo verticale, modo orizzontale, eccetera. Invece le coppie di graffe esplicite o implicite possono, non necessariamente devono, anche prevedere un cambiamento di modo. In generale gli ambienti di LaTeX ricorrono a `\begingroup` incluso nella definizione di `\begin` e `\endgroup` incluso nella definizione di `\end`.

Eseguendo qualunque tipo di definizione mediante uno dei comandi LaTeX oppure TeX, quali `\newcommand`, `\providecommand`, `\renewcommand`, `\def`, `\edef`, la definizione resta locale al gruppo; vale a dire che quando si esce dal gruppo la definizione si perde.

Quando si esegue qualche assegnazione non globale mediante comandi TeX, quali ad esempio `\count...=`, `\advance`, `\multiply`, `\divide`,... (e analogamente per le assegnazioni a tutti gli altri registri di TeX e LaTeX, come le lunghezze, le scatole, eccetera), questa assegnazione resta in vigore solo dentro il gruppo ma, uscendo dal gruppo, ogni registro viene ristabilito allo stato che esso aveva prima di entrare nel gruppo.

Quando si devono eseguire assegnazioni o definizioni complesse usando registri interni o macro interne di TeX o LaTeX, è conveniente eseguire tutte le manipolazioni all'interno di un gruppo, proprio per ristabilire tutto, ogni registro e ogni

macro interna, allo *statu quo ante* cosicché nulla sia alterato nella memoria dell'interprete.

Dentro il gruppo possono venire "scritte" del cose che vanno a finire nella scatola 255, quella dove si accumula tutto quanto è stato composto fino a quel momento e dal quale ogni tanto la routine di output estrae l'equivalente di una pagina da accodare al file di uscita. In questo caso il gruppo serve per riportare tutto allo statu quo ante, tranne che per lo stato della scatola 255 che per la sua funzione è sempre un po' speciale e assolutamente intoccabile direttamente dall'utente.

Può succedere però che dentro il gruppo si eseguano delle elaborazioni il cui risultato finale debba essere usato fuori dal gruppo. Si hanno due possibilità:

1. L'assegnazione o la definizione del risultato finale va eseguito con i comandi TEX che a loro volta vanno preceduti dal comando primitivo `\global`; oppure
2. l'assegnazione o la definizione va eseguita ricorrendo al procedimento che vedremo fra poco.

Nel primo caso l'assegnazione o la definizione diventano globali in assoluto e restano valide anche quando l'interprete sia uscito da ogni qualsivoglia gruppo nel quale si trovava al momento della definizione o assegnazione. Questo fatto potrebbe non essere quello che si desidera; in questo caso bisogna ricorrere al secondo procedimento che fa uso degli alias.

Mi spiego con un esempio; supponiamo di voler calcolare quanto sia lungo l'alfabeto minuscolo latino con un determinato font, che supporremo essere quello in vigore al momento del comando che esegue questa determinazione (ovviamente facciamo finta non non sapere che esiste il comando LATEX `\settowidth`); definiremo allora:

```
\newlength{\alfabeto}

\newcommand*\lunghezzalfabeto{%
\bgroup \setbox0
  \hbox{abcdefghijklmnopqrstuvwxyz}%
\edef\x{\noexpand\egroup
  \noexpand\alfabeto=\the\wd0}
\x}
```

in modo che dopo aver dato il comando `\lunghezzalfabeto` possiamo sapere che l'alfabeto corrente minuscolo è lungo `127.58365pt`.

La definizione della macro in questione comincia con l'apertura di un gruppo mediante l'alias della graffa aperta; imposta poi la scatola numero 'zero' con l'alfabeto minuscolo corrente; poi, e qui comincia il bello, esegue una definizione espansa del comando `\x` dove in realtà espande solo il comando `\the`, che è quello che serve per scrivere il contenuto di un registro o di un comando relativo a un registro; in questo caso si tratta dell'espressione `\wd0` che indica la larghezza (width) della scatola 'zero'; questo valore, proprio perché si tratta di una definizione espansa non viene ancora assegnato al registro dimensionale `\alfabeto` (definito poco sopra) ma viene semplicemente messo nel testo della definizione; i comandi primitivi `\noexpand` servono per evitare che durante l'espansione vengano espansi prematuramente i comandi `\egroup` e l'equivalente in termini di registri dimensionali corrispondente ad `\alfabeto`. Alla fine dell'espansione è come se la macro `\x` fosse stata definita con

```
\def\x{\egroup\alfabeto=127.58365pt}
```

Completata questa definizione la macro `\x` viene eseguita; questo significa che nel buffer d'entrata dell'interprete il nome della macro viene sostituito con la sua definizione e questa a sua volta viene eseguita; per prima cosa si chiude il gruppo (e con ciò si restituisce allo statu quo ante sia lo stato della scatola 'zero' – usata molto spesso nel nucleo di LATEX per gli scopi più diversi – sia il significato della macro `\x` stessa) poi, ormai fuori dal gruppo, il valore della lunghezza dell'alfabeto viene effettivamente assegnato al registro dimensionale `\alfabeto`.

Questo espediente garantisce che il valore calcolato possa essere "lanciato" oltre la chiusura del gruppo, senza lasciare tracce dietro di sé. È superfluo evidenziare che questo espediente è usato spesso all'interno del nucleo di LATEX e di numerosissimi pacchetti di estensione.

## 4 Gli argomenti delle macro

Ogni argomento obbligatorio non delimitato formato da più di un token deve venire passato alla macro racchiuso fra parentesi graffe esplicite. In questo caso non si possono assolutamente usare le graffe implicite mediante i comandi `\bgroup` e `\egroup` perché l'interprete riconosce un argomento di macro proprio riconoscendo le graffe esplicite e contando quante se ne aprono e quante se ne chiudono perché l'argomento composto di molti token, che può comprendere altre possibili graffe adeguatamente appaiate, termina quando l'interprete arriva a riconoscere la parentesi graffa chiusa che si appaia alla prima graffa aperta dopo il nome della macro.

Questo implica un fatto importante: se si crea una macro nella cui definizione compare un'altra macro che agisce sui suoi argomenti, questa deve avere la graffa esplicita aperta e quella chiusa entrambe contenute nella definizione della macro che la contiene. Sembra del tutto ovvio, diamine!

Al momento opportuno si cade in questo trabocchetto e lo dimostro con un esempio: vogliamo creare un ambiente che permetta di incorniciare un testo qualunque. Sappiamo che `\framebox` può

incorniciare un testo che si sviluppa su una sola riga; per incorniciare un testo che si svolge su più righe bisogna passare attraverso un ambiente intermedio come minipage; proviamo un po' e scopriamo che il seguente codice, già abbastanza elaborato, funziona:

```
\framebox{\textwidth}{%
\begin{minipage}{%
\dimexpr\textwidth-2\fboxsep
-2\fboxrule}
Testo di qualunque lunghezza che comporti
un numero di righe maggiore di uno.
\end{minipage}}
```

Bene: allora possiamo creare una macro che possa ricevere come parametri la specificazione della larghezza e il testo; la specificazione della larghezza sia facoltativa, ma il valore di default sia la larghezza della riga corrente:

```
\newcommand\cornice[2][\linewidth]
\framebox[#1]{%
\begin{minipage}{%
    \dimexpr#1-2\fboxsep-2\fboxrule}
#2\end{minipage}}
```

Collaudiamo il nuovo comando e constatiamo funziona con il testo di prova: "Testo di qualunque lunghezza che comporti un numero di righe maggiore di uno."

Benissimo, siamo a cavallo; allora possiamo creare un ambiente:

```
\newenvironment{cornice}[1][\linewidth]{%
%                                  apertura
\framebox{\begin{minipage}{%
    \dimexpr#1-2\fboxsep-2\fboxrule}}
{%                                 chiusura
\end{minipage}}}
```

Verifichiamo l'ambiente e riceviamo subito un messaggio d'errore al quale non sappiamo rispondere. Che cosa è successo? Semplicemente che le graffe che racchiudono l'argomento di `\framebox` sono in comandi diversi; la graffa aperta si trova dentro la definizione del comando di apertura dell'ambiente, mentre la graffa chiusa si trova dentro la definizione del comando di chiusura dell'ambiente. Ciò non è possibile nel modo più assoluto e non c'è verso di risolvere il problema con gli alias delle graffe.

Non solo ma ripiegando sul comando `\cornice`, definito sopra e che aveva passato il collaudo, scopriamo che se il testo non è di un solo capoverso, il comando cessa di funzionare. Diamine, che cosa non va? Lo vedremo nel prossimo paragrafo.

## 5 I delimitatori degli oggetti

I comandi primitivi di TEX spesso ricevono i loro argomenti o gli oggetti su cui devono operare senza fare ricorso alle graffe. I comandi di definizione usano le graffe per racchiudere la definizione di nuove macro e di nuovo queste graffe devono essere esplicite come per le macro definite. Ma usano le graffe anche i comandi per usare le scatole. A questi ci riferiremo in particolare, perché le loro graffe possono essere anche implicite.

LATEX consente all'utente solo di usare scatole orizzontali, dentro le quali esso opera in "LR mode", cioè scrivendo solo in orizzontale, tipicamente da sinistra (L = left) a destra (R = right); in realtà anche LATEX apparentemente offre altre scatole, oltre a quelle definibili con `\newsavebox` e usabili con `\sbox`, `\savebox`, `\usebox`: `\mbox`, `\makebox`, `\fbox`, `\framebox`. In realtà si tratta quasi solamente di comandi di gestione di scatole che in generale vengono riempite, talvolta incorniciate, ma spesso usate immediatamente senza conservarle in memoria. Tutte però sono sostanzialmente scatole orizzontali, anche se talvolta si riesce a caricarle con scatole verticali non accessibili direttamente.

La loro limitazione è che comunque il loro argomento non può essere altro che una linea di testo o anche un capoverso già composto ma ben nascosto dentro una scatola verticale usata tramite il comando `\parbox` o l'ambiente minipage. L'argomento di quei comandi non deve avere nessun comando che possa indurre la macro a credere che il suo argomento sia costituito da più di un capoverso. Ecco come si dà una risposta all'interrogativo posto alla fine del paragrafo precedente.

TEX offre all'utente una varietà di scatole e di comandi per gestirle. Le scatole sono sostanzialmente `\hbox` (che coincide con il tipo di scatola disponibile agli utenti di LATEX), `\vbox`, `\vtop` e `\vcenter`, più una notevole varietà di comandi per gestire, copiare, svuotare quelle scatole. Tutti questi comandi primitivi di caricamento delle scatole richiedono che gli oggetti da caricare siano racchiusi fra graffe esplicite o implicite. In altre parole se l'oggetto da inserire in una scatola fosse fatto da un solo token, non sarebbe possibile omettere le graffe esplicite o implicite che siano; le graffe sono funzionali a questi comandi primitivi e l'interprete le cerca esplicitamente o implicitamente perché in loro assenza esso non sarebbe in grado di usare la scatola.

Le tre scatole verticali disponibili con TEX (e usate internamente da LATEX per costruire il risultato di `\parbox` e dell'ambiente minipage) differiscono per la posizione del punto di riferimento: `\vbox` ha il suo punto di riferimento coincidente con quello dell'ultimo oggetto verticale (in basso) che ha impilato, tipicamente il punto di riferimento dell'ultima riga del suo contenuto; `\vtop`, invece, ha il suo punto di riferimento coincidente con quello del primo oggetto verticale (in alto) che ha impilato, tipicamente il punto di riferimento della prima riga che la scatola contiene. `\vcenter` è un po' speciale perché il suo punto di riferimento, a metà della

sua altezza complessiva, viene collocato sull'asse matematico della riga dove la scatola viene a trovarsi; per cui questa scatola deve venire usata solo in ambiente matematico, anche se il suo contenuto è testuale.

Ecco allora che con le graffe implicite e un po' di programmazione di basso livello possiamo ora risolvere il problema di incorniciare un testo formato di più capoversi. Si esamini il codice:

```
\newenvironment{cornice}[1][\linewidth]{%
\fboxsep=6pt \setbox0\vbox\bgroup
\hsize=\dimexpr#1-2\fboxsep-2\fboxrule
\@parboxrestore
}{\egroup\framebox{\box0}}
```

Ecco subito un esempio dell'uso di questo ambiente:

> Questo è un testo abbastanza lungo per occupare più di due righe in questo articolo composto su due colonne.
> Come si vede, infatti, il secondo capoverso, ancora composto di più righe, si accoda al primo dentro la cornice.

Esso risulta prodotto dal seguente testo nel file sorgente:

```
Ecco subito un esempio dell'uso di questo
ambiente:  \begin{center}
\begin{cornice}[0.9\columnwidth]
Questo è un testo abbastanza lungo per
occupare più di due righe in questo
articolo composto su due colonne.

Come si vede, infatti, il secondo
capoverso, ancora composto di più righe,
si accoda al primo dentro la cornice.
\end{cornice}\end{center}
Esso risulta prodotto dal seguente testo
nel file sorgente:
```

Il commento al codice ora non dovrebbe presentare misteri: siccome stiamo parlando di un ambiente, esso forma implicitamente un gruppo, quindi tutto quello che vi definiamo dentro o le variabili a cui assegniamo dei valori o dei contenuti vengono ripristinate dopo l'uscita dall'ambiente. Pertanto l'assegnazione di un nuovo valore alla variabile dimensionale `\fboxsep` non ha conseguenze sul resto del documento quando l'ambiente verrà chiuso. Si apre poi una scatola verticale del tipo `\vbox` da conservare nella scatola 'zero'; il suo contenuto comincia subito dopo una graffa implicita aperta; siccome dobbiamo comporre il testo in modo verticale (dobbiamo costruire uno o più capoversi) dobbiamo fissare la giustezza della composizione che, a livello delle scatole di TEX, si chiama `\hsize`; in particolare, qualunque larghezza vogliamo che abbia la cornice, quella di default o quella che noi specifichiamo come argomento opzionale all'apertura dell'ambiente, dobbiamo tenere conto che attorno al testo composto apparirà uno spazio attorno ai quattro lati (la cui larghezza è stabilita mediante `\fboxsep`) e, per quanto piccolo, bisogna tenere conto dello spessore del filetto che forma la cornice (il cui valore è contenuto in `\fboxrule`); ecco perché si chiede a TEX di determinare il risultato di una espressione dimensionale con `\dimexpr`, che esegue i calcoli necessari. Fatto questo si completano i comandi di apertura con un comando `\@parboxrestore`, che serve per impostare i parametri compositivi di default all'interno della scatola verticale. Bisogna ora provvedere ai comandi di chiusura: questi cominciano con una graffa chiusa implicita che chiude anche la scatola verticale; il suo contenuto è ora nella scatola 'zero'; finalmente il comando `\framebox` incornicia la scatola 'zero'.

Come ci si rende conto, queste operazioni non sarebbero state possibili se le graffe di apertura e di chiusura delimitanti il contenuto della scatola verticale non fossero state delle parentesi implicite, quella aperta dentro i comandi di apertura e quella chiusa dentro i comandi di chiusura.

## 6 Conclusioni

Con questo tutorial si è cercato di chiarire la funzione delle parentesi graffe aperte e chiuse a seconda di come vengano usate e quindi a seconda del contesto; si è anche mostrata la differenza fra i comandi di gruppo espliciti e impliciti. È chiaro che si potrebbero fare molti altri esempi; il lettore curioso non ha altro da fare se non aprirsi il suo pacchetto preferito oppure il file `latex.ltx` per trovare altri esempi, lo studio dei quali non può che giovargli se desidera costruirsi delle macro che usino le coppie di graffe nei vari modi descritti in questo articolo.

## Riferimenti bibliografici


Beccari, C. (2008a). «I registri token: questi sconosciuti». *ArsTEXnica*.

— (2008b). «Macroistruzioni con argomenti delimitati». *ArsTEXnica*.

Gregorio, E. (2009). «Appunti di programmazione in TEX e LATEX».

Knuth, D. E. (1996). *The TEXbook*. Addison Wesley, Reading, Mass., 16ª edizione.



▷ Claudio Beccari
Villarbasse (TO)
`claudio dot beccari at gmail dot com`

# illumino: An XML document production system with a TEX core

*Matteo Centonza, Vito Piserchia*

## Abstract

XML is the state of the art in publishing technology. Publishers, through the "one source, multiple output" paradigm, are able to publish the same content to multiple media without much effort. In this paper we'll investigate current scenarios for publishers adopting a LATEX workflow and introduce illumino, our fulltext XML production system built around TEX.

## Sommario

XML è lo stato dell'arte delle tecnologie editoriali. Gli editori, usando tecnologie XML, possono pubblicare lo stesso contenuto su diversi media usando come base un unico file sorgente. In questo articolo investigheremo gli scenari attuali per gli editori che adottano un flusso di lavoro basato su LATEX e introdurremo illumino, il nostro sistema di produzione XML Fulltext, costruito intorno a TEX.

## 1 Introduction

XML publishing in scholarly publications is nothing new. Publishers, through content/format separation, can leverage the many benefits of XML:

- Publish the same content to multiple media
- Store production data in a neutral format, the "*lingua franca*" of Internet applications
- Use XML as a neutral format for long-term archival of content
- Disseminate content through syndication
- Have content ready for data harvesting/mining (discussed in sect. 4.3)

With the term "XML publishing", we are referring to procedures and methods generating final output media from XML sources. XML sources are authored to produce final output, ready to be published. On the other hand, XML publishing is a complex task since content should be structured to be valid XML, *i.e.*:

- Encoded with correct metadata granularity
- Follow an XML grammar

XML publishing tools are often complex content management systems (CMS). Users need to perform content authoring according to tool specifications. Import tools may be provided, but imported content needs to be reviewed. This is a time-consuming task.

Publishers interested in XML publishing and adopting a LATEX based workflow, are either supposed to develop complex in-house solutions or outsource most of the publishing chain. There are many outsource facilities more or less ready to do the job but the price to pay is losing control of the work.

In this paper we'd like to present illumino, our fulltext XML production system that is trying to change this scenario. We'll present the ideas behind this technology, system capabilities and discuss future development.

### 1.1 illumino

illumino is a fulltext XML production system, built around (LA)TEX, which integrates international standards such as:

- DocBook 5.0
- MathML 2.0
- SVG Tiny 1.2
- Unicode 5.0

illumino converts LATEX sources into its internal XML format (DocBook) and the publishing chain, starts from XML sources.

The process is similar to the one described in the seminal article by E. Gurari and S. Rahtz Gurari e Rahtz but uses different XML technologies. For a graphical representation of the full process, please see figure 1.

illumino is a multiplatform application built around TEX (TEX Live and the embedded TEX4ht), XSLT 2.0, Java, git (as XSLT) and once configured, has native support for publisher LATEX classes and generates publishers' native production files as output. It is able to run unmodified in the old LATEX workflow.

illumino aims at integrating as smoothly as possible with any LATEX workflow, minimizing production changes to obtain fulltext XML publishing.

To achieve this goal, illumino performs automatic metadata enrichment through heuristic methods to match content granularity needed by a given XML grammar. In order to guarantee content safety while heuristically enriching unstructured information, illumino has been designed to produce output that perfectly matches that of the LATEX production source file the system is processing: we test for equal checksums of source and production

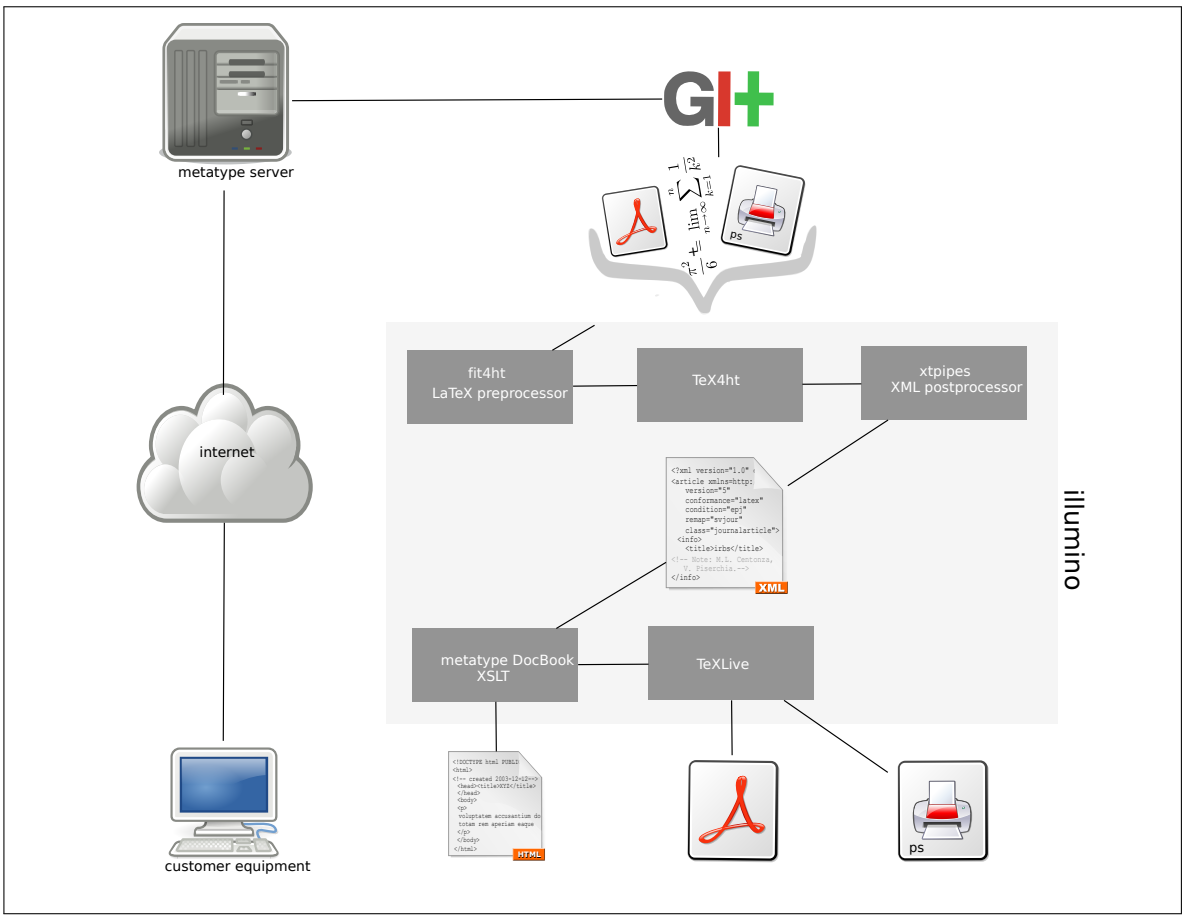


FIGURE 1: The `illumino` architecture

output (currently PostScript output) to ensure this. When this perfect match ("*equivalence*") applies we are sure that the system has not introduced any modification to document content, so there's no need to review the article content.

`illumino` has embedded content checking (via SHA checksums) and the user is warned when the system outcome is not the perfect equivalence; in those cases, `illumino` is able to visually highlight differences found, so that visual validation can take place.

`illumino` is an incremental (à la Apache `ant`), client/server application and is able to run through the network with speed similar to that of a conventional LaTeX workflow. By integrating XSLT technologies, `illumino` can be used concurrently in a safe way. The complete list of features is given on the main `illumino` web page at http://www.metatype.it/illumino.html.

## 2 `illumino` architecture

Figure 1 shows current `illumino` client/server interaction. `illumino` uses standard components and implements standard and open protocols.

`illumino` has its foundations on just two main components: TeX Live and Java.

From a technical point of view, `illumino` is based on Apache `ant` and is implemented as several custom `ant` tasks, through our `illuminant` library (`antlib`). By using `ant`, `illumino` is an incremental (through dependencies and timestamps calculations) and multithreaded application (Java).

The system is completely standalone[1] and `ant`, used also to build the whole stack, is able to update and rebuild all upstream dependencies.

What follows is a description of high-level processes of which `illumino` is made.

### 2.1 fit4ht

This part of `illumino`, as its name may suggest, is responsible for making the initial LaTeX source file "fit" to be run under TeX4ht. This workflow segment parses LaTeX document and by using heuristic algorithms performs:

- Automatic document cleanup (*e.g.* standardize misused TeX primitives and sloppy constructs to LaTeX)
- Enrich document metadata structure (split and tag content according to information patterns)
- Make some constructs ready to be correctly interpreted by TeX4ht

1. With the exception of the Apache Tomcat servlet container (used to implement the caching XSLT engine) and `git` XSLT program.

From a low-level point of view, fit4ht is implemented as an ant filterreader.

### 2.2 TEX4ht

TEX4ht is the heart of illumino and is the component taking care of LATEXto-XML transformations.

We'll not delve into TEX4ht internals since this is out of the scope of this article. For a more in-depth explanation of how TEX4ht works, the reader may refer to Cevolani; Goossens *et al.* (1999).

TEX4ht's most notable difference with other similar technologies is the use of the real thing, the TEX parser, when converting a TEX file to another format.

For simplicity, we'll condense the TEX4ht workflow to three main steps:

1. Seed configurable (at the control sequence level) hooks in DVI output
2. Harvest the seeded hooks to generate a given markup representation
3. Post-process the outcome to undergo validation

We have heavily customized TEX4ht[2] mainly to:

- Implement a native backend for DocBook 5.0 output.
- Add support in the TEX4ht core for editorial fine tuning control sequences (*e.g.* supporting all tuning `toks`, vertical, horizontal, and math spaces, ...) as XML processing instructions.
- Enrich control sequence mapping in order to go from LATEX→XML and back without degradation in information quality.

By pre-processing input files and slightly modifying some TEX4ht internals, we have made the LATEX→XML conversion a completely automated process.

We have developed custom "(LA)TEX4ht compile" `ant` tasks to have automated compilation of sources. Compile reruns are handled automatically (*e.g.* TEX4ht, for complex tables have to run several times, and LATEX needs to be rerun when labels are modified).

Through TEX4ht's power and flexibility we've been able to have fine-grained content resolution and exactly remap a LATEX file into its corresponding DocBook 5.0 counterpart, producing the same output (we call it "*equivalence*" and their outputs have identical checksums).

`illumino` testcases are made of "*equivalences*" with research papers in physics from different scholarly publications. This approximately 400 pages and 30 articles test suite is `illumino`'s internal certification system and is used to avoid regressions and to spot inconsistencies in the whole `illumino` application stack (including upstream dependencies). For every build, `illumino` must pass these test cases that are constantly updated as soon as we implement new features or fix bugs.

At present, `illumino` has been tested on approximately 4k pages of content from hard sciences.

2. Thanks to the invaluable help of Eitan Gurari.

### 2.3 XML transformation phase

`illumino` uses XSLT to transform the raw XML document generated by the previous phase (TEX4ht).

In more detail, `illumino`'s XML transformation phase is currently using XSLT 2.0 and takes advantage of its features, *e.g.* by using `xsl:function`, `xsl:character-map`, regular expressions and pattern matching features extensively.

The XSLT 2.0 phase must be seen as a multi-stage stack of stylesheets, where every filter accomplishes a different task.

XSLT stylesheets are organized in two main sets:

- `xtpipes`, an XSLT pre-phase, which takes care of space rearrangement and element positioning, and produces an enriched and valid DocBook document;
- Metatype DocBook XSLT, transforming the resulting DocBook document to all supported formats (including LATEX with publisher's class).

#### *2.3.1 `xtpipes` stylesheets*

In this stage, the filter performs:

- Space rearrangements
- Element reordering and structure enrichment
- Validation fixes

Space rearrangements are strictly related to the design decision of aiming at full equivalence with source output. LATEX and XML spaces obey completely different sets of rules in determining the output. In LATEX spaces can appear almost anywhere in the source document but may be relevant to output in only some cases; conversely, an XML grammar strictly controls the allowed spaces in the document tree.

In order to achieve "*equivalence*" between source and production output, we have handled all corner situations in which the meaning of spaces from LATEX and XML differ.

Regarding element reordering and enriching, we have to face the different nature of semi-structured and structured data. For example, in LATEX documents, many commands can change the properties of the entire group or environment when specified inside that group. Almost all the alignment commands have this behaviour (*e.g.* `\centering` inside a floating environment). On the other hand, on the XML side we have to specify this

behaviour with the tag that represents the LaTeX environment, with permitted attributes, if any (*i.e.* `align="center"` inside the CALS table element).

Keeping in mind that seeding of TeX4ht hooks is sequential and happens when TeX sees the commands, we have two possibilities:

- using elements and attributes suggested by the XML schema, when meaningful and close to LaTeX counterparts (*e.g.* alignment in table environments)
- using a powerful transclusion and linking technique

`xtpipes` stylesheets follow the first approach where possible and in the remaining cases revert to using a built-in `xlink`/`xpointer` processor, implemented with XSLT function extensions.

For example, the `xpointer` scheme can be used to link other elements in the document and the `xinclude` syntax can be used to transclude from other documents.

We have been able, with our XSLT 2.0 `xpointer` implementation, to point to any other element in the document and *e.g.* change attribute values. In short, we have XSLT transformations driven by the XML content, so in the final analysis governed by TeX4ht.

When the latter method is not applicable, we resort to bare XML processing instructions to render the construct.

Validation techniques are discussed in sect. 4.1.

#### *2.3.2 metatype DocBook stylesheets*

This phase produces the supported output formats, starting from valid DocBook 5.0 sources. Leveraging XML's strengths, we can generate several output documents (*e.g.* simple text, HTML, LaTeX or documents in other XML markup languages) from the same XML source.

### 2.4 DocBook version 5

DocBook, developed by the OASIS consortium, is a semantic markup language for technical documentation. As a semantic language, DocBook is focused on content and meaning (DocBook has not been designed to visually format content).

DocBook offers several advantages over competing markup languages:

- Long history and schema stability
- Wide adoption and great availability of tools that support authoring of DocBook documents
- Capability to generate output files in a wide variety of formats (HTML, XSL-FO and LaTeX for later conversion into PDF or other document markup languages), lately `epub`
- Semantic similarities with LaTeX commands
- Modular structure including widely adopted XML grammars (*e.g.* MathML and SVG)

For a more in-depth explanation of DocBook concepts, the reader may refer to Walsh.

### 2.5 `illumino-remote`

`illumino` is a client/server application built upon open protocols. `illumino` leverages XSLT technologies, and the backend system exposes `git` (`http://git-scm.com/`) interfaces.

`illumino-remote`, the system client, interacts with the remote `illumino` server through the `git` protocol.

Whenever the `git` daemon receives new changesets (deltas) for a given article from a client, a new local (server) workflow run will be launched on the updated sources and results (*e.g.* XML, PDF deltas) will be sent back to the client.

Normally `git` roundtrips are very fast[3] in comparison to other XSLT technologies and we are able, in combination with `ant` behind the scenes, to have `illumino` processing time be on the same order as a LaTeX workflow run.

`illumino-remote` is a Java application with JMS message passing between client and server. We are waiting for the pure Java `git` (`jgit`) implementation to mature, in order to have a pure Java client.

`illumino-remote` can control all remote backend behaviour such as:

- Repository operations (add, delete article resources)
- Enable/disable output formats
- Choose the PDF output engine (pdfTeX, Adobe Distiller, `ghostscript`)
- Show output differences[4]
- Enforce output equivalence[5]
- Choose a secondary XML output format

## 3 Usage caveats

`illumino` has been designed to integrate as smoothly as possible into any existing LaTeX workflow.

XML publishing, starting from unmodified LaTeX production sources, is a complex software task, while being a cost-effective way for publisher to enable a full text XML workflow. Aspects of this complexity are:

- Automatic enrichment of semi-structured content to a more structured form
- Proper separation of content from presentational elements.

What follows is a list of production caveats.

3. Deltas (differences) for storing changesets and fast merging/indexing algorithms let `git` compete with some native filesystem operations.

4. Visual differences are presented when the transformation does not end with output equivalence.

5. `illumino` will fail the transformation if the result is not equivalence.

### 3.1 Automated content tagging

Often LaTeX sources are not sufficiently structured to permit a 1:1 mapping with the majority of XML schemata. To be able to fill all the data structures provided by an XML schema, we have to properly resolve pieces of information adhering to specific patterns. These patterns are able to take care of most of the production scenarios we have seen during the heavy test phase our product has undergone.

Out of the box, `illumino` is able to resolve correctly and to split various sparse information that in other semi-automatic systems users tag manually.

This process is by no means perfect since it is completely heuristic. In some corner cases, this approach may not be completely satisfactory and manual tagging is needed. If a new content pattern is found or highlighted, it will be added to existing filters.

In other cases, heuristic treatment is simply ineffective (such as affiliation splitting) and users must manually tag content to get the needed granularity (*e.g.* split into organization name, division, etc.).

Our long-term aim is to integrate `illumino` with the `UIMA` framework and leverage Bayesian annotators to automatically split what currently is done manually (see sect. 4.2).

### 3.2 Content/presentation separation

LaTeX has a plethora of commands, environments and class infrastructures which allow for a very high fraction of content separated from presentation.

Authors strictly adhering to LaTeX and class instructions will provide a very good source base to transform to XML. Unfortunately this is not always true, and non-standard environments, low level TeX code instead of standard LaTeX, TeX font primitives, etc., are easily found.

We have done our best to automatically transform non-standard code to a more conformant form, preserving its original meaning. This again will probably not cover all possible cases. In a few cases, users should manually convert the non-standard code.

### 3.3 XML validation

A document, to be valid according to an XML grammar, should be checked not only at the structural level but also at the element content level (*i.e.* not only how elements nest but also what elements contain).

This streamlines further processing to other formats and *e.g.* long term archiving of content (one of the most interesting parts of an XML workflow).

This (not surprisingly) comes at a cost: content sometimes should be rearranged in order to adhere to a given XML schema. The upside is that document overall quality will be increased.

In most situations, TeX4ht is able to produce valid XML documents, but some problematic cases exist. In our experiments, we have found at least two classes of problems in which validation should be refined at a later XML post-processing stage.

As already mentioned, this is due to the strict rules imposed on an XML document when compared with the weak structure imposed by the LaTeX grammar: LaTeX to XML transformation can produce XML chunks that do not fit in the XML structure (*e.g.* elements outside the allowed parent).

We have solved these validation problems by using XSL context-aware `xpath` expressions, rearranging the offending chunk and folding it with the most appropriate parent element, whenever the XML schema allows this. With this approach we are able to solve most validation problems. In some remaining cases, users must resort to recoding LaTeX sources to solve validation problems; a high fraction of problems come from offending XML chunks generated from a sloppy or invalid use of LaTeX constructs.

## 4 "*What the future brings*"...

### 4.1 XML validation

Currently we validate XML documents through the Namespace-based Validation Dispatching Language (NVDL).

NVDL is able to route content coming from a given namespace in order to be validated by the correct namespace grammar. In this way, we are able (by using DocBook) to intermix content validated through DTD, RelaxNG, and XML Schema.

`oNVDL`, an open-source NVDL implementation based on `Jing`, is our choice.

In the future, we want to explore the opportunity to take advantage of other XML validation languages. In particular our attention and future efforts are focused on the Schematron validation language. By using Schematron rules we will be able to deal more easily with current validation constraints.

### 4.2 Improving unstructured content parsing through the `UIMA` framework

In section 2.1 we introduced `fit4ht` filters taking care of document metadata structure enrichment, information tagging and code cleanup.

`fit4ht` is a set of specialized modules taking care of enriching information structure by adding context metadata. The nature of `fit4ht` modules is heuristic: whenever document excerpts adhere to a given pattern, information can be split (safely, since "*equivalence*" or visual validation comes to help).

Since one of `illumino`'s tasks is to treat unstructured/partially structured information to con-

vert it into a more structured form, in the long term we'll port fit4ht modules to Apache UIMA (`http://uima.apache.org/`).

Unstructured Information Management applications are software systems that analyze large volumes of unstructured information in order to discover knowledge relevant to an end user. An example UIM application might ingest plain text and identify entities, such as persons, places, organizations; or relations, such as works-for or located-at.

The UIMA frameworks support configuring and running pipelines of Annotator components. These components do the actual work of analyzing the unstructured information. Users can write their own annotators, or configure and use pre-existing annotators. Some annotators are available as part of the UIMA project; others are contained in various repositories on the Internet.

By integrating `illumino` with the framework we will be able to leverage the software ecosystem built around UIMA and *e.g.* split information based on Bayesian inference or address other editorial tasks such as normalization of inflected forms.

### 4.3 Knowledge mining

Another interesting field for which scientific XML content is particularly suited is *knowledge mining*.

Several advances in computer science have been brought together under the rubric of "data mining" Langley e Simon (1995). Techniques range from simple pattern searching to advanced data visualisation and neural networks. Since our aim is to extract comprehensible and communicable scientific knowledge, our approach should be characterised as "knowledge mining".

Our idea is to create a network of links between research articles from various fields of science and accelerate research, scientific discovery and innovation.

The key point is that scientific papers, especially from the hard sciences, encode most of their content using mathematical expressions. Every mathematical expression has a unique meaning.

By indexing all occurrences of mathematical expressions present in research papers, it would be possible to build a network of links between research articles. Analyzing links between different fields of knowledge would make it possible to deduce symmetries, patterns, and even similarities that could be used as research targets.

### 4.4 `illumino` GUI

We plan to develop a graphical interface in order to have a smooth interaction with the system. This graphical interface should integrate a LaTeX editor and will handle remote interaction with the system.

In our plans, this will be done by developing an Eclipse plugin, in order to leverage the Eclipse ecosystem to have advanced functionalities such as:

- Real-time shared editing
- Context sensitive editing
- Seamless remote interaction
- Versioning and change management (à la `git`).

## References

Cevolani, G. «Introduzione a TeX4ht». *Proceedings of the 2004 Italian GuIT meeting (in Italian)*. URL `http://www.guit.sssup.it/guitmeeting/2005/articoli/cevolani.pdf`.

Goossens, M., with E. Gurari, S. R., Moore, R. e Sutor, R. (1999). *The LaTeX Web Companion*. Addison-Wesley.

Gurari, E. e Rahtz, S. «From LaTeX to MathML and back with TeX4ht and passiveTeX». URL `http://www.cse.ohio-state.edu/~gurari/docs/mml-00/mml-00.html`.

Langley, P. e Simon, H. (1995). «Applications of machine learning and rule induction». *Communications of the Association for Computing Machinery*, **38** (11), pp. 54–64.

Walsh, N. *DocBook: The Definitive Guide*. O'Reilly & Associates. URL `http://www.docbook.org/tdg/`.

▷ Matteo Centonza
metatype, Via Santacroce 13/5, I-40122 Bologna, Italy
`matteo at metatype dot it`

▷ Vito Piserchia
metatype, Via Santacroce 13/5, I-40122 Bologna, Italy
`vito at metatype dot it`

# PDF/A-1a in ConTEXt-mkiv

*Luigi Scarso*

## Abstract

I present some considerations on electronic document archiving and how ConTEXt-mkiv supports the ISO Standard 19500-1 Level A Conformance (PDF/A-1a:2005), an ISO standard for long-term document archiving.

## Abstract

Alcune considerazioni sulle problematiche dell'archiviazione di documenti elettronici e come ConTEXt-mkiv supporta lo standard ISO 19500-1 Level A (PDF/A-1a:2005) relativo all'archiviazione digitale documentale.

## 1 Introduction

In this paper I will try to illustrate some aspects of electronic documents archiving starting from the position that it's fundamentally a typographic language problem, and also, in the contest of information technology, a programming language one. After some theoretical considerations, I will show some important typographic languages that are used, and then I will briefly talk about the ISO Standard PDF/A-1 and how ConTEXt-mkiv tries to adhere to its requirements. About the typographic style of this paper, I will follow these simple rules: I will avoid footnotes and citations on running text, and I will try to limit lists (e.g. only itemize and enumerate) and figures; the last section before the References one will collect all citations.

## 2 Some theoretical considerations

The main problem of electronic document archiving is to find a good answer to the question: "Will we be able to read it again in the next $x$ years or, better, forever?" There are basically two kinds of requests: just to be able to understand the document contents, or to be able to fully and faithfully reproduce the original document. With electronic documents we aim at the second one.

It should be clear that it is not a problem to make an identical copy of an electronic document, but to obtain a reasonable confidence that we will be able to read the present document as originally intended. So the problem is to define an electronic format which is clear enough to allow a correct implementation of a consumer program both now and in the future, hence robust against accidental errors and independent of external resources, i.e. *self contained*, and semantically adequate for the purpose of document archiving.

The first requirement calls for a widely accepted standard; the semantically adequacy calls for a (formal) language, and precisely a *Typographic Language*. Some authors also distinguish between digital typography (the design and rendering of a "character"), micro typography (which covers aspects of type and spacing, such as kerning between letters, ligatures, line breaking etc.), and macro typography (that covers the visual quality of the document, hence the design of headings, lists, pages but also colors and images). A formal typographic language ideally covers all these aspects while being also a programming language.

We actually need to preserve both the content and the visual appearance of the document and inevitably this demands for a language that is able to talk of fonts, colors, images and *animations* in an abstract fashion and easily connects these abstractions to the concrete world of printing (and not only printing on paper), i.e. it must be a *Page Description Language*.

But this is not enough: we need another more abstract connection to the world of semantic and structure. As of today, we are still unable to exhibit a practical algorithm that checks the semantic correctness of a document in a human language, and there are some problems with the syntax: we can only hope to check the semantic of the typographical unit, the "character". This seems more affordable: just a (possibly unlimited) list of `(glyph, id, semantic)`, where `glyph` is the pictorial representation of a "character" and `id` is a unique identifier to prevent misunderstanding (for example a space is also a "character" and there are several kinds of spaces that we need to distinguish). The typographic language hence can use the `id` both to display the "character" (its `glyph`) and as a reference to its `semantic`. But here we enter into a cultural and linguistic domain: for example the `semantic` require a standardized metalanguage and more often than not a glyph is intimately tied with the literature of a given cultural area (of the present as of the past): it's not unusual to see the same glyph in completely different contexts that must be clearly distinguished. It's hence reasonable to expect practical/contingent conventions hard to implement or to respect — and perhaps also meaningless in the future. Even the concept of lists may be impractical: some glyphs are combi-

nations of others "characters" (basic signs and/or other glyphs) and the rules of combinations can lead to infinite possibilities: that means that we need a standard *Character Language*.

The structure of the document calls for a *Markup Language*. This field is better known and well established: the key points are a clear separation between structure and contents, and the usage of a standard metalanguage. This is a peculiar property of an electronic document: the structure in a paper document can be inferred from the physical copy, but it's not embedded in it. Markup languages permit computational classification of electronic documents, high degree of content reuse (e.g. automatic speaking), efficiency in storage, hyper link capabilities, while preserving the original document unaltered — things that are difficult or impossible to achieve in traditional documents.

The controversial point is about the semantic: given a markup language, what is its domain? Is it wide enough to cover all the aspects of our document? Is it infinite or finite? For example, a markup language about the book's structure can cover only a part of all the aspects of a technical drawing or an invoice document. We can think of a sort of a universal infinite markup language that covers all the aspects, but sentences of an infinite language can be long enough to become a practical problem, and, most important, markup languages are hard to design. Ideally, we would like a language defined by a compact grammar, with infinite sentences and a wide semantic; practically we must rely on the human common sense of measure and adaptability.

So far we have seen that a typographic language alone is not sufficient to honor the contents. We need other kinds of languages. But even with them we need something else to be able to reproduce the document as originally intended: we must be reasonably certain that the current document *is* the original document. Traditional paper documents are always tied to a physical support: we have only one original and one or more copies. Sometimes the copies are so faithful that the original seems unimportant (think of a newspaper) and sometimes things can be arranged to produce two or more "originals" — but it is another way to say that there is one original and one o more copies that are "indistinguishable from the original for practical purpose".

Things change radically with the electronic documents: the copy is indistinguishable from the original, and it's easy to verify if two documents are equal — just a comparison between bytes — as to keep track of changes (the history of the document). But the negative side is that it's now more difficult to decide the right one between two documents when both claim to be the "original one".

Again, it's another peculiar property of any electronic document that solves the problem: the property to be seen as a *number*. Each stream of bits can be easily analyzed and transformed by means of a particular kind of mathematical functions, among which the *one-way function* plays a central role: it is a function that is *easy* ("cheap enough for the legitimate users") to calculate for every input but *hard* ("prohibitively expensive for any malicious agents") to invert given the image of a random input. A trivial example: it is easier to multiply two large numbers than to find the divisors of a large number.

A one-way function can be used to encode a stream of bits so that only who knows the key can decode it, but it can also be used to detect and prevent the modifications of the stream itself — and this is what we need for the document archiving. It's the modern edition of the signet ring, in facts it is called *digital signature*, but with a subtle difference: up today a formal proof does not exist that inverting an one-way function will *always* be a hard task. Unfortunately the history just says the opposite: with the raising of computational power and the advancement of modern algebra some of the early one-ways-functions were "easily" (for a malicious agent) inverted so that literally every day the security experts must check the news. With Internet there are more potential malicious agents today than in the past, software and documentation are available for free, more computational power can be achieved with clustering (often abusively). And, finally, for a casual user managing a secret key can be problematic.

It's important to note that the digital signature depends only on the digital nature of the content, and not from any typographic property: every stream of bits (i.e. every kind of file) can be encrypted.

## 3 Some typographic languages

In the previous section we delineated some properties of a typographic language for document archiving: now let's see some real candidates.

### 3.1 SVG and XSL-FO

The Scalable Vector Language is a W3C recommendation for describing two-dimensional graphics both static and dynamic. It's a vector graphics markup language in XML format and hence it can express concepts like fonts, colors, curves but it's not a strictly page description language — every SVG graphics has exactly one page, even if multiple layers are possible. A W3C draft extending the standard SVG with the notion of pages has been written, but development efforts are now directed to the next release of SVG, so it's unlikely that this draft will have further influences in the present recommendation. Moreover, there are no ways to

express the logical structure of a page.

Another W3C recommendation that is related to typography is the Extensible Stylesheet Language, an XML application that defines a language for transforming XML documents (XSLT) and an XML vocabulary to specify formatting semantics, informally referred to as XSL formatting objects or XSL-FO.
The semantic of XSL-FO is mostly related to the layout of a document; there are some structural elements especially for book-like documents (i.e. tables, lists, but not sectioning), and hence it looks like a natural companion of SVG because, thanks to the namespace, we can compose documents with fragments from different markup languages, while preserving the syntactic and semantic correctness of each language.

SVG and XSL have some good points: they are standards according to a widely recognized world organization, they are free from royalties and freely available, and there are software tools to check the syntactic correctness. At least for SVG, the Inkscape program is a quite good editor for an average use and the last release has also the interesting feature of embedding the JavaScript language into a graphic, so that it's possible to consider SVG + JavaScript as a full typographic language. As for any XML application, they both use the Unicode standard for the character encoding.

But as of today their diffusion is still limited: one of the most important browser, Internet Explorer 8, still lacks support for SVG (it should be supported in the next version 9) and XSL-FO is quite simple in its typographic capabilities to gain popularity *per se* even if we consider that XSLT is a programming language — but intended to transform generic XML documents, not specific for typography. For the document archiving purpose a document markup language with a richer semantic like DocBook should be preferable, but managing three different standardized languages is not a good solution, given that each one evolves independently; on the other side just two of them are not enough.

### 3.2 TEX

TEX is a *typographic macro-programming Turing-complete language* by D. Knuth. The original language is described by the author in The TEXbook, and it consists of about 300 primitive commands that cover, in essence, how to organize the characters in rows, the rows in lines and the lines in pages, the indexing and the tables. In the same book it is also described the *plain format*, a collection of almost 600 macros built on top of the primitive commands, that provides sectioning and a few other typographic constructs (and used to typeset the book itself). It has only a basic notion of graphics, but the macro `\special` permits to implement extensions: particularly important is the extension to the PostScript language, a page description language which is also a full-featured programming language.
The key point is the Turing-completeness. With TEX we can build an arbitrary format that is, basically, a document markup *and programming* language and this definitely solves the problem to choose the right markup language for document structure because new structures can be build remaining inside the format. The macro nature of the language is also well suited to process the input, so that it's possible to build macros that manage a particular text encoding — theoretically all computable *character languages*. Knuth also designed a compact page description language, the *DeVice Independent* format, or DVI, that represents the output of a `tex` file as processed by the TEX program, and the METAFONT programming language to design fonts (i.e. it's a *digital typographic programming language*), which is the companion program to produce bitmap fonts. Finally he also described the complete implementation in PascalWeb of all these programs. TEX was so accurately designed and so deeply tested on a wide range of hardware/software (still today) that we can consider it as practically bug free.

On the other side, TEX was not designed for archiving purpose. The DVI format is not self contained (for example, the fonts aren't included) and this is true even if we consider the TEX source file as an electronic format. There is no standard organization behind TEX and the original program today is surpassed by new implementations: the most important are pdfTEX, XETEX and luaTEX. Finally, TEX is almost unknown outside the scientific community, even if its hyphenation algorithm is widely used.

### 3.3 PDF

The Adobe Portable Document Format (PDF) is the successor of the PostScript language, a well known and established *page description language* for printing documents. Basically it extends the PostScript model by adding interactive features as navigation and annotations (these are quite similar to a static `html` page with a script language similar to php or JavaScript; in fact PDF uses JavaScript), tree dimensional images, movies and animations (all for screen documents), a complete support for Unicode, a new font technology, digital signature, a document meta-markup language and a simple html-like markup language and a radically different electronic format — a binary format instead of textual.

Adobe started PDF around 1993 and until now, following the same line of conduct of PostScript all specifications are publicly available, as there is a pdf reader free for downloading (the full featured pdf editor Acrobat is available as a commercial

product). In 2008 the PDF version 1.7 became the ISO standard 32000-1:2008, and as of today Adobe has promulgated two extensions (for Rich Media contents and forms) which probably will be included into the next ISO standard 32000-2 under development.

There are many good points in PDF. Practically all electronic documents can be converted in a self contained PDF. Thanks to PDF binary format, exchanging and archiving are more reliable because even the modification of only one byte may entail an error when reading. Editing is also difficult (but nowadays less than in the past) and so accidental modifications; conversely producing PDF is increasingly simple and there are now better pdf readers other than the Adobe one (but currently very few commercial products can compete with the Adobe pdf editor). The print quality of PDF is the same of PostScript but a pdf file can also embed the logical structure and finally, most important, pdf documents are enormously widespread all around the world, fitting well with the local typographic traditions. It's a winning ISO Standard.

But there are also some limitations. PDF is not a programming language: unlike PostScript, for example, it needs another language (the *Job Description Format*) to describe a print job, and unlike TEX we cannot use it as a typographic language. In the end, it's not so different from SVG: even Adobe has started the *Mars* project to give "an XML-friendly representation for PDF documents called PDFXML", and it's not necessary to invent a sort of binary SVG because the digital signature can be used to detect and prevent document modifications. Finally it seems that by the end of this year all the most important browsers will support the SVG format to some extent, so they can be insidious competitors for the pdf reader (not for printing, anyway).

## 4 The PDF/A-1 ISO Standard

Probably one of most known PDF version is `PDF 1.4` (around 2001, almost ten years ago) maybe because the companion Acrobat 5.0 was a robust programs and the pdf Reader was freely available for several platforms both as a program and as plugin for browsers. We keep having a huge amount of electronic documents that are in `PDF 1.4`, hence we should not be surprised if Adobe pushed it as reference for document archiving. What follows is a verbatim copy from http://www.digitalpreservation.gov/formats/fdd/fdd000125.shtml and it's a good description:

PDF/A-1 is a constrained form of Adobe PDF version 1.4 intended to be suitable for long-term preservation of page-oriented documents for which PDF is already being used in practice. The ISO standard [ISO 19005-1:2005] was developed by a working group with representatives from government, industry, and academia and active support from Adobe Systems Incorporated. Part 2 of ISO 19005 (as of September 2010, an ISO Draft International Standard) extends the capabilities of Part 1. It is based on PDF version 1.7 (as defined in ISO 32000-1) rather than PDF version 1.4 (which is used as the basis of ISO 19005-1).

PDF/A attempts to maximize device independence, self-containment, self-documentation. The constraints include: audio and video content are forbidden, JavaScript and executable file launches are prohibited, All fonts must be embedded and also must be legally embeddable for unlimited, universal rendering, colorspaces specified in a device-independent manner, encryption is disallowed, use of standards-based metadata is mandated.

The PDF/A-1 standard defines two levels of conformance: conformance level A satisfies all requirements in the specification; level B is a lower level of conformance, "encompassing the requirements of this part of ISO 19005 regarding the visual appearance of electronic documents, but not their structural or semantic properties".

In essence the standard wants to ensure that every typographic element, from the low level character to the high level logical structure is unambiguously defined and unchangeable — and it does, it achieves its purpose: every character must be identified by a Unicode id, which is an international standard and a *character language*, every color must be device independent by means of a color profile or output intent, there must be precise metadata informations for classifications and the document must have a logical structure described by a (possible ad-hoc) markup language.

Unfortunately the pdf version 1.4 is quite old: animations and 3D pictures cannot be embedded, the font format cannot be OpenType, JavaScript programs are not permitted at all, even if they don't modify the document in any way as, for example, a calculator. Ten years ago it was very important to guarantee that the document would always be printed as intended, nowadays screen is slowly replacing paper and animations play a fundamental role: PDF/A-1 is good for paper but less than optimal for "electronic paper".

## 5 PDF/A-1a in ConTEXt-`mkiv`

Given that it is still under heavy development, ConTEXt-`mkiv` has the opportunity to be developed on two fronts: the "low level" luaTEX (CWEB code and Lua primitives) and the "high level" macros that build the format itself. One of this year results is the implementation of "tagged PDF", the Adobe document markup language for PDF documents, and the development of color macros for the PDF/X specifications. As a consequence, it was

possible to use these results to test some real code for producing PDF/A-1a compliant documents. Let's start with an example explained step-by-step.

```
%% Debug
\enabletrackers[backend.format,
                backend.variables]
%% For PDF/A
\setupbackend[
format={pdf/a-1a:2005},
profile={default_cmyk.icc,
         default_rgb.icc,default_gray.icc},
intent={%
ISO coated v2 300\letterpercent\space (ECI)}
]
%% Tagged PDF
%% method=auto ==> default tags by Adobe
\setupstructure[state=start,method=auto]

\definecolor[Cyan][c=1.0,m=0.0,y=0.0,k=0.0]
\starttext
\startchapter[title={Test}]
\startparagraph
\input tufte
%% Some ConTeXt env. are already mapped:
%% colors
\color[red]{OK}
\color[Cyan]{OK}
%% figures
\externalfigure[rgb-icc-sRGB_v4_ICC.jpg]
               [width=0.4\textwidth]
\stopparagraph
%% Natural tables
\bTABLE
\bTR\bTD 1 \eTD \bTD 2 \eTD \eTR
\bTR \bTD[nx=2] 3 \eTD\eTR
\eTABLE
\stopchapter
\stoptext
```

As usual the file is processed with

```
#>context test.tex
```

and it doesn't hurt to enable some debug information with

```
\enabletrackers[backend.format,
                backend.variables]
```

### 5.1 Enable the PDF/A-1a

To enable PDF/A-1a we must setup the backend with the appropriate variant of PDF/A. From the very beginning ConTEXt has had a backend system that permits to use almost the same macro-format for different outputs (i.e. DVI and PDF), and with luaTEX this system is increasingly enhanced, as we'll see later on.

With `format={pdf/a-1a:2005}` we select the `1a` variant of PDF/A standard and the label is mandatory because it also puts some default metadata into the output (see `lpdf-pda.xml`; a complete list of formats is currently in `lpdf-fmt.lua` and also as a `Lua` table `lpdf.formats`).

Next comes the colors part, and we must pay attention here. The key concept is:

*every color must be independent of any device.*

In a pdf we usually have two sources of colors: the colors specified by the author, e.g something like `\definecolor[orange][r=1.0,g=0.5,b=0.0]` and the images. The 3 most used color spaces `DeviceGray`, `DeviceRGB`, `DeviceCMYK` are device dependent because the reproduction of a color from these color spaces *depends* on the particular output device; but the real output devices are all different due both to the different nature (screen vs. printer, for example) and different technologies (CRT vs. LCD screen, or inkjet vs laser printer, for example). Every device can be classified by means of a *color profile* which maps an input color (rgb, cmyk or gray) to an *independent color space* so that we achieve two goals: such maps assure that each device will correctly reproduce the color, and the independent color space permits to compare colors from different color spaces. With

```
profile={default_cmyk.icc,
         default_rgb.icc,default_gray.icc},
```

we associate all the document colors with the corresponding color profile by mean of a filename (the file `colorprofiles.xml` has a list of predefined profiles). Of course it's wrong to tie a rgb color space to, for example, a cmyk profile, and not all profiles are good too.

There is a second way to specify colors, but it's a bit tricky. We must specify that all the colors *without profile* are intended to be used with a common output profile, i.e. we must impose an *output intent*: this is the meaning of

```
intent={%
ISO coated v2 300\letterpercent\space (ECI)}
```

which is a cmyk profile for coated paper. Note that we are using a name and not a filename to avoid clashing with the values of the `profile` key. By doing so we accept these implicit limitations and color space conversions:

- if the output intent is a cmyk profile then the document can have only cmyk and gray colors;
- if the output intent is a rgb profile then the document can have only rgb and gray colors;
- if the output intent is a gray profile then the document can have only gray colors.

They are reasonable: in general we cannot use a rgb color with a cmyk profile because there are rgb colors without equivalent cmyk ones (that is to say that screens display more colors than printers). We can convert a gray color to rgb or cmyk because usually gray color spaces are a subset of the former (otherwise we have a really poor device). It's not an error if we specify both profiles and output intent: at least if all color spaces have their own profiles, as in the example, then the output intent is simply ignored by a PDF/A compliant pdf reader.

Finally the images: we must be sure that every image has its color profile but as of today ConTEXt-mkiv cannot help here. There are some good programs like MagickWand that are really useful for these tasks.

### 5.2 Tagged PDF

Next we must enable the tagging system with `\setupstructure[state=start,method=auto]`. ConTEXt-mkiv permits the author to define his own document markup language (the tags used inside the pdf document) but of course we also need the associated TEX macros. This naturally leads to start with a sort of XML document:

```
\setupstructure[state=start,method=none]
\starttext
\startelement[document]
\startelement[chapter]
opes
\startelement[p]\input ward\stopelement \par
\stopelement
\stopelement
\stoptext
```

The internal tag names are `<document>`, `<chapter>` and `<p>` as we see in fig. 1 from Acrobat 9.0, but we still need to put the appropriate typographic elements into the PDF.

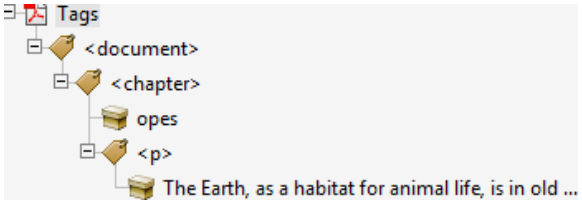


FIGURE 1: The tags structure of a simple document

In the context of PDF/A, a validation program expected the tags as defined by Adobe and this leads to some "syntactic sugar" macros, i.e instead of

`\startelement[chapter]...\stopelement`

it's better to use

`\startchapter[title={Test}]...\stopchapter`

which puts the correct tags and also typesets the chapter title Test as expected.

The complete list of tags can be found in `strc-tag.mkiv` and of course ConTEXt-mkiv permits to redefine the default mapping. In fig. 2 our document shows that ConTEXt-mkiv had already mapped some predefined typographic objects like figures and tables to the appropriate tags.

We can use this mechanism to embed an XML document into a tagged pdf document, which opens quite interesting perspectives, but we can also start from a "structured TEX" document and end into an XML one, and this is more interesting because it's a matter of backend only — and because it's already implemented:

```
\setupbackend[export=yes]
```

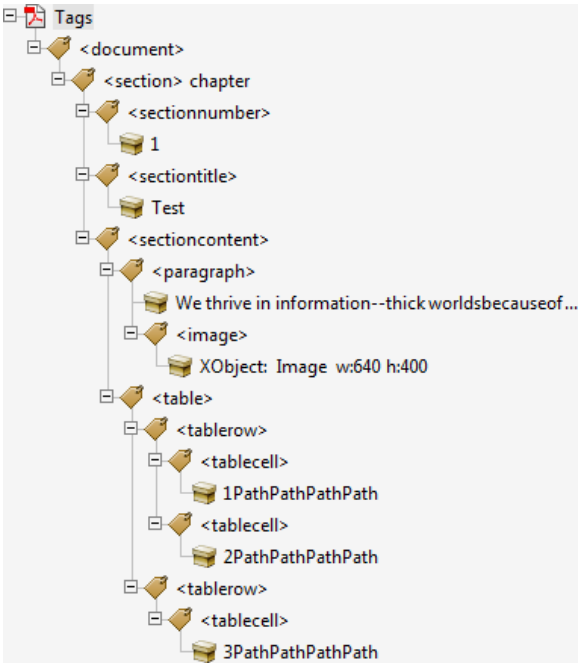


FIGURE 2: The tags structure of a more complex document

```
\setupstructure[state=start,method=none]
\starttext
\startelement[document]
\startelement[chapter][title=Test]
opes
\startelement[p]\input ward\stopelement \par
\stopelement
\stopelement
\stoptext
```

produces a `<tex-file>.export` like this (original XML spaces are not preserved in this listing)

```
<?xml version='1.0' standalone='yes' ?>
<!-- input filename   : test-2            -->
<!-- processing date  : 10/09/10 15:28:48 -->
<!-- context version  : 2010.09.24 11:40  -->
<!-- exporter version : 0.10              -->
<document language='en'
   file='test-2' date='10/09/10 15:29:04'
   context='2010.09.24 11:40'
   version='0.10'>
<chapter title="Test">opes
<p>
The Earth, as a habitat for animal life, is
in old age and has a fatal illness. Several,
in fact. It would be happening whether humans
had ever evolved or not. But our presence is
like the effect of an old-age patient who
smokes many packs of cigarettes per day
------ and we humans are the cigarettes.
</p>
</chapter>
</document>
```

### 5.3 Fonts and encoding

In the previous subsection we have seen that with simple macro we can have a *valid* (i.e validated by Acrobat 9.0) PDF/A-1a pdf document. We still didn't talk about fonts.

The default fonts used by ConTEXt-mkiv are the OpenType version of LatinModern, and, as of now, they cannot be embedded into PDF/A documents because OpenType isn't supported in version 1.4; this is not a problem because, in essence, ConTEXt-mkiv strips the OpenType part and embeds a valid Type1 or TrueType font. Given an OpenType font, ConTEXt-mkiv is also able to map each glyph to its Unicode id, so even this side is not problematic.

Unfortunately, it's already known that typesetting mathematics with the Computer Modern fonts easily leads to invalid PDF/A documents due to misleading dimensional information of some fonts. As widely noted by C. Beccari, just the simple `$a\not=b$` invalidates the whole document, due the wrong dimension of the `\not` sign. What are the solutions? There are two of them, both unsatisfactory:

1. choose another (valid) math family;
2. make a high resolution (more than 300dpi) bitmap of each invalid formula.

Of course it's possible to edit the fonts, but it's not a general solution: there are copyright limitations and we should subset a modified copy of the font when the original version is different — an error prone situation because modifications of PDF/A-1a document are permitted, and an editor uses the system fonts. The problem remains even if ConTEXt-mkiv can patch the font on the fly. A way out is the complete embedding of the patched fonts, so that the editor uses the document fonts, but it's not a robust solution — some editors can still use the original system fonts. Sometimes just a change of glyph is sufficient:

```
\startluacode
function desc2utf8(desc)
 local us =''
 local plane = 0
 for i,v in pairs(characters.data) do
  if v.description == desc then
   us = v.unicodeslot
   break
  end
 end
 return tex.sprint(tex.ctxcatcodes,
              unicode.utf8.char(us))
end
\stopluacode
\def\N#1{\ctxlua{desc2utf8("#1")}}
\starttext
\startTEXpage
$a\not=b$
$a\N{NOT EQUAL TO}b$ %% Use Unicode names!
\stopTEXpage
\stoptext
```

leads to

$a \neq b$ $a \neq b$

where the first inequality makes an invalid PDF/A-1 while the second does not. But the aesthetic result is very different.

## 6 Conclusion

The PDF/A1-a is a good standard for document archiving: it's quite complete *Page Description Language*, it relies on Unicode which is also quite good *Character Language* and on Type1 and TrueType as *digital typography formal language*; it has also a good *Document Markup Language*. The binary electronic format and the digital signature for detection and prevention of document modifications complete the picture. The restrictions (e.g. profiles for colors) together with a freely available PDF/A-1a compliant pdf reader lead to an concrete self-contained format.

PDF/A-1a support in ConTEXt-mkiv is still experimental because it needs more tests, but programming in luaTEX is easier than in pdfTEX, and the 1.4 is a well known pdf version. The colors management can be probably improved by permitting to specify a color and its profile for a single object and not for the whole document, as it currently is.

On the other hand, the model of PDF/A-1 is the traditional paper. The exclusion of animations and 3D pictures is questionable and perhaps also the scripting languages should be allowed if they don't change the document.

The ISO standard is not freely available and the PDF/A-1a validators are expensive and complex to implement: this is an obstacle for the diffusion of PDF/A.

In this sense the SVG seems to have more chances than PDF. For example by defining an XML schema that is a subset of full SVG but tailored for document archiving, we automatically gain validation, because free XML validators already exist. From the TEX world we learned that a typographic Turing complete programming language is more powerful than a simple description language, and perhaps SVG can use JavaScript for this — and maybe it will end in a TEX-like language. But even if this scenario will become reality, ConTEXt-mkiv-users can still program typographic tasks as they do today: it will be only a matter of designing a new backend.

## 7 Notes on References

In section 2, for *digital/micro/macro typography* see Richy and Mittelbach. Unicode Unicode is an example of *Character Language* and Wikipedia Markup is a good starting point for *Markup Languages*; the W3C site XML maintains the specifications for XML. One-way functions are described in OneWayFunction and the pdf reference PDFRef on section "Digital Signatures" shows how it's implemented in PDF.

In section 3, for SVG see the W3C site at SVG; for XSL-FO see Wikipedia at XSLFOWiki and th W3C site at XSLFO. For DocBook see docbook; a complete pdf reference is at PDFRef and for

JavaScript see Wikipedia at javascript.

For section 4, there are some informations on PDF/A-1 at Wikipedia pdfaWiki, at the techdoc at pdfa, and at the digitalpres. Very useful are also the references of C. Beccari's paper at cbpdfa. An interesting use of JavaScript in PDF is calc.

For section 5, the ConTEXt wiki pdfxctxwiki has some terse informations because the code is the ultimate reference. Tagged PDF is described in the new version of `hybrid.pdf` hybrid that is part of "*Proceedings of the 4th ConTEXt meeting*"brejlov, (to be published). For ICC profiles a good starting point is ICCPRofile; font problems are described by C. Beccari in cbpdfa.

## References

URL `http://cajun.cs.nott.ac.uk/compsci/epo/papers/volume8/issue2/2point5.pdf`.

URL `http://www.tug.org/interviews/mittelbach.pdf`

URL `http://unicode.org/standard/WhatIsUnicode.html`.

URL `http://en.wikipedia.org/wiki/Markup_language`.

URL `http://www.w3.org/XML/`.

URL `http://en.wikipedia.org/wiki/One-way_function`.

URL `http://www.adobe.com/devnet/pdf/pdf_reference.html`.

URL `http://www.w3.org/TR/2003/REC-SVG11-20030114/`.

URL `http://en.wikipedia.org/wiki/XSL_Formatting_Objects`.

URL `http://www.docbook.org/`.

URL `http://www.w3.org/TR/2006/REC-xsl11-20061205/`.

URL `http://en.wikipedia.org/wiki/JavaScript`.

URL `http://en.wikipedia.org/wiki/PDF/A`.

URL `http://www.pdfa.org/doku.php?id=pdfa:en:techdoc`

URL `www.tug.org/applications/pdftex/calculat.pdf`.

URL `http://www.digitalpreservation.gov/formats/fdd/fdd000125.shtml`.

URL `http://en.wikipedia.org/wiki/ICC_profile`.

URL `http://wiki.contextgarden.net/PDFX`.

URL `http://www.pragma-ade.com/general/manuals/hybrid.pdf`

URL `http://meeting.contextgarden.net/2010/talks/`

URL `http://www.guit.sssup.it/downloads/Beccari_Pdf_archiviabile.pdf`.

▷ Luigi Scarso
luigi dot scarso at gmail dot com

# Presentazioni animate in LaTeX

*Gianluca Pignalberi*


## Sommario

Le presentazioni fatte con LaTeX non sono affatto condannate alla staticità totale. Un manipolo di strumenti e le possibilità del formato PDF premettono risultati più che perfetti.

## Abstract

LaTeX presentations are not condemned to stay totally static. A small pool of tools and the PDF format capabilities allow more than perfect results.


## 1 Introduzione

A causa della mia impossibilità a partecipare al GUIT*meeting2009* ho dovuto ingegnarmi con la presentazione. Visto che nel meeting precedente ho causato non pochi problemi logistici affinché qualcuno presentasse in mia vece, mi serviva un modo per le presentazioni di... autopresentarsi.

La maggior parte dell'utenza informatica crede che PowerPoint sia l'unico strumento per fare presentazioni (tanto da usare la locuzione "Presentazione PowerPoint" come se il nome commerciale di un prodotto fosse assurto ad apposizione). L'utenza più smaliziata conosce addirittura Impress di OpenOffice.

Ovviamente in ogni categoria ci sono una o più sottocategorie, e la categoria degli utenti di TeX non fa eccezione. Non credo di sbagliare molto dicendo che esiste ancora qualche "dinosauro" che impagina le proprie presentazioni su pagine A4, stampate su fogli lucidi, riducendosi poi a coprire parte dei lucidi con i fogli opachi, a inseguire la "classicità". Credo anche che molti utenti usino invece una classe o un pacchetto appositamente realizzati per le presentazioni. Questo articolo è per tutti loro, in quanto dimostrerà praticamente che, anche nelle presentazioni, LaTeX:

1. è slegato dai formati di pagina usuali;
2. è in grado di fare quello che fa PowerPoint;
3. lo fa in modi diversi, restituendoci la facoltà di scelta;
4. permette ottimi risultati.

Farò tutto questo descrivendo, come caso d'uso, proprio la creazione della mia presentazione per il GUIT*meeting2009*.

## 2 Una presentazione minimale con beamer

Beamer è forse la più famosa classe LaTeX per realizzare presentazioni. Nel tempo si sono successe varie classi e diversi strumenti, a partire da SLiTeX, passando per Pdfscreen e Prosper fino a PPower4. Beamer sembra riassumere tutte le caratteristiche dei citati programmi, e sembra includerne altre.

La struttura di un documento Beamer minimale è molto semplice: si parte con il solito preambolo, in cui la classe usata sarà proprio beamer e in cui includeremo tutti i pacchetti usati per compilare la presentazione e stabiliremo un tema e uno schema di colore, dopodiché passeremo al corpo del documento, che conterrà quasi certamente comandi specifici di Beamer (anche se non siamo obbligati a farlo, denotando un'idiosincrasia tra intenzioni e azioni).

Un esempio poco più che minimo, con un solo frame e dei blocchi al suo interno, è il seguente:

```
\documentclass{beamer}

\usepackage[T1]{fontenc}
\usepackage[latin1]{inputenc}
\usepackage[italian]{babel}
\usepackage{graphicx}

% This is the file main.tex
\usetheme{JuanLesPins}
%\usecolortheme{orchid}
\title[\LaTeX{} e \emph{comma below}]{combelow: abbasso i segni diacritici di serie B}
\author{Gianluca Pignalberi}
\date{Pisa, 17 ottobre 2009}
\begin{document}
\section{Sommario}
\begin{frame}{Sommario}
\begin{block}{Sommario}
Romeno e lettone devono essere considerate lingue di serie B nel mondo di \TeX?
Se finora lo potevano essere, questo piccolo pacchetto tenta di riportarle al
livello giusto con il semplice uso del segno diacritico corretto. Niente pi\'u
cediglia al posto del comma below.
```

```
22 \end{block}
23 \begin{block}{Abstract}
24 Should Romanian and Latvian be
       considered second choice
       languages in the \TeX{}
25 world? Though up to now they
       could have been, this small
       package tries to put
26 them back at the right level,
       just using the correct
       diacritic mark. No more
27 cedilla instead of comma below.
28 \end{block}
29 \end{frame}
30 \end{document}
```

La linea 1 contiene la dichiarazione della classe usata per comporre il documento; le righe 3–6 specificano i pacchetti usati per comporre la presentazione; nella riga 9 il comando `\usetheme` indica quale tema (schema grafico) avrà il documento (la riga 10, qui commentata, contiene la scelta dello schema di colore da usare). Le righe 11–13 contengono titolo, autore e data del documento. Il titolo breve, quello tra parentesi graffe nella riga 11, sarà il titolo mostrato in alto in tutti i frame della presentazione.

La riga 15 mostra il ben noto comando `\section`: il suo contenuto viene mostrato subito sotto al titolo del documento in ogni slide entro il suo *scope* (cioè finché il compilatore non incontra un nuovo comando `\section`). Ogni comando `\section` viene inoltre usato per comporre il sommario (Table of Contents).

La riga 16 contiene l'istruzione che dichiara l'inizio di un frame (una pagina della presentazione) con il relativo titolo. Quest'ultimo è mostrato in grande subito sopra il frame vero e proprio (a seconda del tema, le posizioni qui discusse possono essere diverse). Le righe 17 e 23 dichiarano due blocchi diversi all'interno dello stesso frame. Anche questi hanno il proprio titolo, messo in una fascia colorata diversamente dal resto del blocco. Il blocco conterrà tutto il testo contenuto nel documento tra i comandi `\begin{block}` e `\end{block}`. Allo stesso modo un frame è formato da tutto quanto contenuto tra `\begin{frame}` e `\end{frame}`.

Il risultato del codice precedente è quello mostrato nella figura 1.

Un frame corrisponde all'intera area di una pagina della presentazione (quella che chiameremmo normalmente *slide*), mentre un blocco corrisponde a un pezzo di testo composto all'interno di un rettangolo colorato. Ovviamente nessuno ci vieta di scrivere del testo senza includerlo in un blocco: il testo starà sullo sfondo del frame. Nel listato seguente vediamo lo stesso esempio, ma senza usare i blocchi:

```
1 \documentclass{beamer}
```

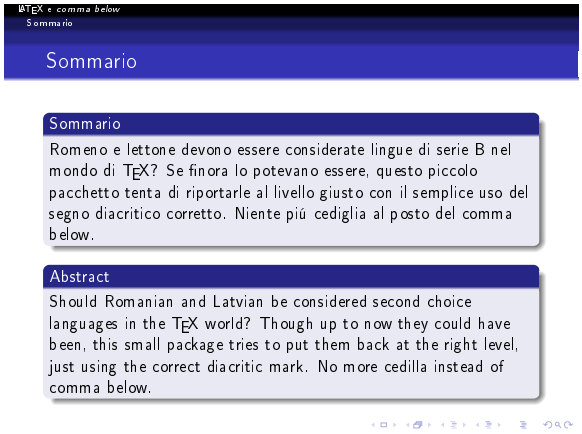


FIGURA 1: Presentazione composta da una sola pagina. La pagina è composta da un frame e relativo titolo e da due blocchi con i relativi titoli

```
2  \usepackage[T1]{fontenc}
3  \usepackage[latin1]{inputenc}
4  \usepackage[italian]{babel}
5  \usepackage{graphicx}
6  \usetheme{JuanLesPins}
7  \title[\LaTeX{} e \emph{comma
       below}]{combelow: abbasso i
       segni diacritici di serie B}
8  \author{Gianluca Pignalberi}
9  \date{Pisa, 17 ottobre 2009}
10 \begin{document}
11 \section{Sommario}
12 \begin{frame}{Sommario}
13 Romeno e lettone devono essere
       considerate lingue di serie B
        nel mondo di \TeX?
14 Se finora lo potevano essere,
       questo piccolo pacchetto
       tenta di riportarle al
15 livello giusto con il semplice
       uso del segno diacritico
       corretto. Niente pi\'u
16 cediglia al posto del comma
       below.
17
18 Should Romanian and Latvian be
       considered second choice
       languages in the \TeX{}
19 world? Though up to now they
       could have been, this small
       package tries to put
20 them back at the right level,
       just using the correct
       diacritic mark. No more
21 cedilla instead of comma below.
22 \end{frame}
23 \end{document}
```

Il documento manca solo delle righe di apertura e chiusura dei blocchi, e il risultato della sua compilazione è molto più lineare, come possiamo vedere nella figura 2.

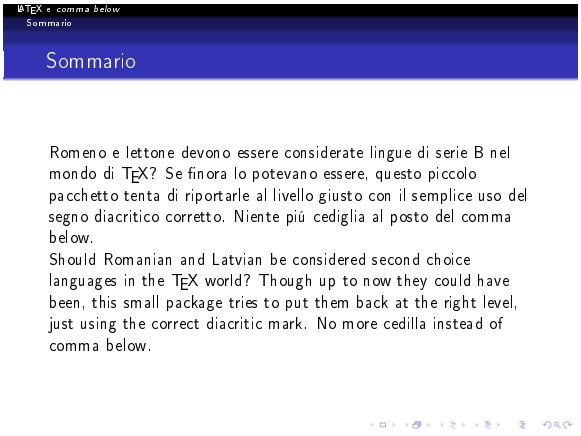


FIGURA 2: La presentazione mostrata nella figura 1 ha ora perso i blocchi, ponendo il testo direttamente nel frame

## 3 Il signore delle slide

Continuare a scrivere le presentazioni secondo i due basilari esempi precedenti fa sì che tutte le pagine della presentazione siano presentate nella loro interezza, una per una, senza che si possa svelare man mano il contenuto di ogni pagina.

Ovviamente Beamer ha un meccanismo per mostrare progressivamente il contenuto di una pagina. In realtà il meccanismo è più generale e serve a mostrare parti di un frame solo in determinati momenti (che non sono legati al tempo, come vedremo andando avanti), che io chiamerei *transizioni*.

Il meccanismo di cui parlo ha una sintassi molto semplice: immediatamente prima della parte di testo da mostrare si dà un comando `\onslide`, seguito dall'intervallo in cui mostrare il testo stesso. Supponiamo che un frame abbia tre frasi, che vogliamo mostrare una dopo l'altra; prima del testo della prima frase metteremo il comando `\onslide<1->`, prima del testo della seconda daremo `\onslide<2->`, e prima del testo della terza frase scriveremo `\onslide<3>`. Un numero $n$ seguito da un trattino indica che il testo seguente verrà mostrato dalla transizione $n$ fino all'ultima. Se dopo il trattino mettiamo un altro numero ($m$), questo indicherà l'ultima transizione in cui il testo sarà visibile (dopo non lo sarà più). Un unico numero $n$ significa invece che il testo sarà visibile solo all'$n$-esima transizione. Infine, un trattino seguito da un numero $m$ indica il testo presente dalla prima slide all'$m$-esima.

Come si evince dal nome del comando, Beamer definisce *slide* quella parte di frame che viene presentata in un dato momento. È una cosa simile al meccanismo di un cartone animato, in cui al lucido di sfondo si sovrapponevano uno o più lucidi con i personaggi e altri elementi, e un concetto del tutto uguale ai livelli delle immagini nei programmi di fotoritocco come Gimp o Photoshop.

Tutto in Beamer può costituire una slide e determinati elementi hanno una sintassi semplificata proprio per sveltire la scrittura. Nei comandi specifici di Beamer possiamo dare i numeri di slide direttamente dopo il comando, senza specificare `\onslide`. Nel listato seguente questa sintassi è usata nella riga 14; le righe 19, 22 e 25 usano `\onslide`.

```
1  \documentclass{beamer}
2  \usepackage[T1]{fontenc}
3  \usepackage[latin1]{inputenc}
4  \usepackage[italian]{babel}
5  \usepackage{graphicx}
6  \usepackage{combelow}
7  \usetheme{JuanLesPins}
8  \title[\LaTeX{} e \emph{comma
       below}]{combelow: abbasso i
       segni diacritici di serie B}
9  \author{Gianluca Pignalberi}
10 \date{Pisa, 17 ottobre 2009}
11 \begin{document}
12 \section{Introduzione}
13 \begin{frame}{Introduzione}
14 \begin{block}{Cosa dice Unicode
       }<1->
15 Unicode ci dice che le lettere
       con il segno a uncino o
       quello a virgola sono
16 sempre ``con cediglia''.
17 \end{block}
18
19 \onslide<2->
20 La forma della cediglia varia in
        base alla lettera: \c{c} vs
       .~\cb{k}.
21
22 \onslide<3->
23 \cb{T} e \cb{S} sono considerate
        varianti tipografiche per il
        romeno di \c{T}
24 e \c{S}.\\
25 \onslide<4>
26 Il simbolo `~\cb{}~' ha comunque
        il suo nome: \emph{comma
       below} (o
27 \emph{comma accent}).
28 \end{frame}
29 \end{document}
```

Il risultato della divisione in slide (in questo caso, quattro) è visibile nella figura 3.

Come già accennato, è molto facile ottenere frame in cui le slide sono visibili solo in determinati momenti, specificando in quali momenti della presentazione una slide deve essere visibile. L'effetto è di ottenere un testo che appaia quando serve e scompaia quando deve sparire, rimanendo solo il tempo necessario.


Figura 3: Un frame formato da quattro slide, mostrate una dopo l'altra

## 4 Tempo

Quanto visto finora implica che un umano sia posizionato al computer per "voltare pagina" ogni volta che serva. Nel mio caso, però, dovendo io essere assente dal convegno, avrei avuto bisogno di qualcuno che "voltasse pagina" per me.

Beamer è in grado di automatizzare esigenze come la mia e fare in modo che la pagina si volti da sola, o nel frame compaia una nuova slide. Il "cronometro" che temporizza i cambiamenti è dato dal comando `\transduration`. Quest'ultimo prende in input il numero di slide tra parentesi angolari e il numero di secondi tra parentesi graffe. Ad esempio, possiamo temporizzare le slide del precedente codice con

```
\transduration<1>{10}
\transduration<2>{10}
\transduration<3>{10}
\transduration<4>{10}
```

e avere una presentazione che mostra ogni slide per dieci secondi, e al termine volta pagina (se la pagina successiva esiste).

Con questo abbiamo posto le basi per creare una presentazione autopresentante con Beamer. Siamo già in grado di farne una, a meno di suoni e animazioni, di cui parleremo nei prossimi paragrafi.

## 5 Suoni e filmati nei frame

Beamer ha la capacità di inserire file sonori e animati nelle presentazioni. A onor del vero, è grazie alle caratteristiche del formato PDF che una cosa del genere è possibile. In caso vogliate approfondire le caratteristiche del formato, potete fare riferimento a PDF. Vediamo nel dettaglio come inserire questi file multimediali nelle nostre presentazioni.

### 5.1 Suoni

Beamer fornisce il comando per inserire un suono in un frame. Questo comando, `\sound`, si trova nel pacchetto multimedia. Ci permette di far eseguire un suono all'apertura di una slide o alla pressione di un tasto. Questa funzionalità, abbinata alla possibilità di temporizzare la durata della persistenza delle pagine, ci permette di calibrare esattamente (o quasi) il tempo oltre il quale far cambiare pagina.

A causa di un bug presente in alcune versioni di Acrobat Reader, dobbiamo essere molto precisi nel passare i dati di codifica del brano da riprodurre.

Inoltre, sebbene sia in teoria possibile riprodurre sia brani in formati non compressi che compressi, il lettore di PDF di Acrobat non riesce a riprodurre questi ultimi, e gli unici due formati funzionanti sono `.aif` e `.au`. Dunque teniamo a portata di mano un programma per la conversione di formati sonori.

La mia esigenza era quella di eseguire un brano vocale (la mia voce che presentava il particolare della pagina) in ogni blocco in un frame. Dunque il codice aveva tanti `\sound` quanti erano i `block` in un frame. All'esecuzione, però, era sempre il primo file vocale del frame a essere riprodotto all'apparizione di un blocco. La soluzione più "comoda" è stata evitare i blocchi in ogni frame e includere un unico file sonoro all'inizio del codice del frame.

Il comando usato nella presentazione è `\sound[autostart,automute,bitspersample=8,inlinesound,channels=2]{}{slide22.au}`. Questo impone che il suono venga riprodotto all'apertura del frame, che la riproduzione termini alla fine del file sonoro, e che i dati del file sonoro siano quelli indicati.

### 5.2 Filmati

Anche l'inclusione di filmati è resa molto semplice da Beamer: il comando `\movie` è quello d'elezione per inserire video in formato AVI o MPEG. Non l'ho usato nella presentazione perché non era quella la mia esigenza. L'unica cosa da ricordare è che `\movie` è incluso nel pacchetto multimedia, distribuito con Beamer, ma come pacchetto separato, e utilizzabile anche indipendentemente da Beamer.

## 6 L'animazione di grafica vettoriale

### 6.1 Creazione dei fotogrammi

Ho avuto bisogno di inserire una piccola animazione in grafica vettoriale nella presentazione. Tale animazione è stata realizzata a partire da una serie di PDF prodotti con LATEX per mostrare enfaticamente come lavorava la prima versione del pacchetto combelow. Il mio scopo era rendere graficamente i passaggi del programma per spiegare più semplicemente l'algoritmo sottostante: ciò non vuol dire che l'animazione mostrasse *esattamente* quanto fatto dal pacchetto.

Non sono stati pochi i problemini da risolvere, sebbene tutti legati ai PDF da animare e non a LATEX e ai suoi pacchetti. Ne parlo qui a beneficio di quei lettori che potrebbero essere interessati a modificare un PDF in maniera diretta.

Tanto per non dimenticare, stiamo parlando del problema del posizionamento di un segno diacritico a forma di virgola (*comma below* o *comma accent*) sotto una lettera. La prima versione del pacchetto calcolava la posizione finale della virgola a partire dalla sua posizione canonica dopo la lettera, in base alla larghezza della lettera stessa. Dunque, le uniche cose che conosciamo per l'animazione sono

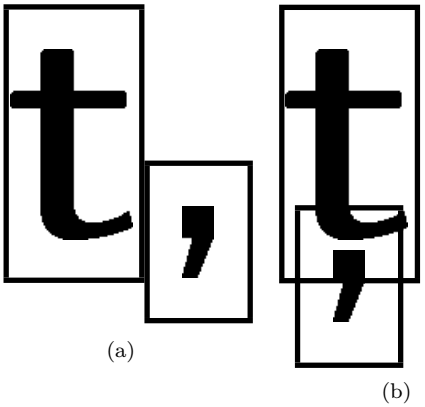


FIGURA 4: Configurazioni iniziale e finale per l'animazione di un'immagine in grafica vettoriale. Notate che la figura b è appositamente imbandierata a sinistra: è la virgola che si sposta verso la t, e non viceversa, o l'una verso l'altra vicendevolmente

le configurazioni iniziale e finale, mostrate nella figura 4.

Mentre la figura 4(a) è data dalla sequenza di testo `t,`, 4(b) è data dal comando `\cb{t}`, così come spiegato in PIGNALBERI (2009). Avrete notato che la figura 4(b) non è centrata; ciò fa pensare che il bordo destro non stia vicino all'estremità destra del disegno. Chiariremo tutto tra poco.

Generare una figura vettoriale come quella nella figura 4(a) è molto semplice: scriviamo un documento LATEX che contenga un codice come

```
\sffamily\textbf{
\Huge
\setbox0\hbox{t}
\setbox1\hbox{,}
\framebox[\wd0]{t}\framebox[\wd
    1]{,}}
```

senza numero di pagina o altro, compiliamolo con PdfLATEX per ottenere un PDF (che sarà in grafica vettoriale) e ritagliamo la figura con `pdfcrop`.

In accordo al pacchetto combelow v0.99a, la figura finale 4(b) può essere generata da un codice come questo:

```
\sffamily\textbf{
\Huge
\setbox0\hbox{t}
\setbox1\hbox{,}
\framebox[\wd0]{t}\raisebox{-.3
    ex}{\hskip-.85ex\framebox[\wd
    1]{,}}}
```

che ha un parametro numerico per `\hskip` in sostituzione dei calcoli del vero codice.

Generare le figure intermedie implica modificare i valori per `\hskip` e per `\raisebox`. Valori di `hskip` tra $-0.1$ e $-0.85$ avvicinano la virgola alla lettera fino a centrarla su di essa; valori di `\raisebox` tra $-0.1$ e $-0.3$ abbassano la virgola fin sotto la lettera. La tabella 1 riassume i valori

di posizionamento orizzontale e verticale per ogni fotogramma.

Come evidente dalla tabella 1, i fotogrammi non hanno le stesse dimensioni, né orizzontali, né verticali. Se provassimo ad animare una sequenza composta da quei fotogrammi otterremmo un pessimo risultato: ogni fotogramma verrebbe ridimensionato alle dimensioni del primo fotogramma. In questo caso vedremmo tutti i fotogrammi successivi al primo (che è il più largo e il più corto) deformarsi per allargamento e per accorciamento, e un'animazione che non renderebbe giustizia al vero processo.

La prima cosa da fare, prima di realizzare l'animazione vera e propria, è rendere tutti i fotogrammi delle stesse dimensioni. Per fare questo editeremo i file PDF come se fossero file di testo.

Per semplificare il lavoro, riporto qui le prime righe del primo fotogramma, omettendo quelle con caratteri binari incomprensibili all'uomo:

```
1  %PDF-1.4
2  %...
3  4 0 obj <<
4  /Length 38
5  /Filter /FlateDecode
6  >>
7  stream
8  ...
9  endstream
10 endobj
11 3 0 obj <<
12 /Type /Page
13 /Contents 4 0 R
14 /Resources 2 0 R
15 /MediaBox [0 0 20 25]
16 /Parent 5 0 R
17 >> endobj
```

La parte da modificare è quella relativa al bounding box dell'immagine, cioè quella della riga 15. Dovremo fare in modo che il bordo sinistro di tutti i fotogrammi rimanga fisso, cosicché non sembri la lettera a spostarsi, ma la virgola. Inoltre occorre fare in modo che anche il bordo superiore rimanga fisso, sempre affinché sia la virgola a muoversi anche verticalmente. Il bounding box (`/MediaBox` nel file PDF) è specificato da quattro numeri, che rappresentano la coordinata iniziale (primi due numeri, ascissa e ordinata dell'angolo in basso a sinistra) e finale (ultimi due numeri, ascissa e ordinata dell'angolo in alto a destra), e quindi la dimensione del box stesso. Nella tabella 2 sono elencati i valori dei *media box* calcolati da `pdfcrop`

Tabella 1: Valori di posizionamento orizzontale e verticale per i fotogrammi dell'animazione in grafica vettoriale

| **Numero fotogramma** | **Shift orizzontale** | **Shift verticale** | **Fotogramma** |
|---|---|---|---|
| 0 | -0.0 ex | -.0 ex |  |
| 1 | -0.1 ex | -.0 ex |  |
| 2 | -0.2 ex | -.0 ex |  |
| 3 | -0.3 ex | -.0 ex |  |
| 4 | -0.4 ex | -.0 ex |  |
| 5 | -0.5 ex | -.0 ex |  |
| 6 | -0.6 ex | -.0 ex |  |
| 7 | -0.7 ex | -.0 ex |  |
| 8 | -0.8 ex | -.0 ex |  |
| 9 | -0.85 ex | -.0 ex |  |
| 10 | -0.85 ex | -.1 ex |  |
| 11 | -0.85 ex | -.2 ex |  |
| 12 | -0.85 ex | -.3 ex |  |

Tabella 2: Dimensioni del bounding box di ogni fotogramma dell'animazione di grafica vettoriale

| Numero fotogramma | MediaBox originale | MediaBox finale |
|---|---|---|
| 0 | 0 0 20 25 | 0 -4 20 25 |
| 1 | 0 0 19 25 | 0 -4 20 25 |
| 2 | 0 0 18 25 | 0 -4 20 25 |
| 3 | 0 0 17 25 | 0 -4 20 25 |
| 4 | 0 0 16 25 | 0 -4 20 25 |
| 5 | 0 0 15 25 | 0 -4 20 25 |
| 6 | 0 0 13 25 | 0 -4 20 25 |
| 7 | 0 0 12 25 | 0 -4 20 25 |
| 8 | 0 0 12 25 | 0 -4 20 25 |
| 9 | 0 0 12 25 | 0 -4 20 25 |
| 10 | 0 0 12 26 | 0 -3 20 26 |
| 11 | 0 0 12 28 | 0 -1 20 28 |
| 12 | 0 0 12 29 | 0 0 20 29 |

e quelli finali modificati a mano, in base alle dimensioni di tutti i fotogrammi. Questi numeri, che sembrano essere usciti come conigli dal cilindro di Mandrake, hanno invece un significato ben preciso. Il fotogramma più largo è il primo (quello numero 0), mentre il più alto è l'ultimo (numero 12). Dobbiamo fare in modo che tutti i fotogrammi siano larghi quanto il primo e alti quanto l'ultimo. Dobbiamo inoltre rispettare il vincolo di non far spostare a destra la lettera (quindi fissiamo il bordo sinistro del box, variando il destro, cioè lo spazio dopo la lettera, da dove la virgola parte) e di non farla spostare in basso (quindi fissiamo il bordo superiore e accettiamo di variare quello inferiore, cioè lo spazio sotto la lettera, dove la virgola arriva). Dopo il cambiamento delle dimensioni tutti i fotogrammi avranno i lati orizzontali di 20 pt, e quelli verticali di 29 pt.

Ora i fotogrammi sono pronti da animare e includere nel documento (naturalmente l'animazione non si vedrà nel documento cartaceo).

### 6.2 L'animazione

Ora che abbiamo lungamente visto come creare dei fotogrammi di un'animazione in grafica vettoriale, passiamo a vedere in pratica come ottenere l'animazione in un file PDF.

Beamer fornisce un comando utile a ottenere le animazioni di cui parliamo. Naturalmente non è necessario che i fotogrammi siano in grafica vettoriale: possono anche essere immagini raster (foto, illustrazioni o altro).

Beamer è in grado di creare due tipi di animazioni: il primo è relativo ai frame e ai relativi effetti di avvicendamento. I comandi a esso dedicati sono `\animate` e `\animatevalue` (trovate tutte le informazioni relative su beamer). Il secondo tipo di animazione riguarda invece l'avvicendamento di immagini esterne. Il comando dedicato è `\multiinclude` e si trova nel pacchetto xmpmulti. Io ho provato a usarlo, ma l'ho trovato criptico e scomodo da usare. Inoltre non sono riuscito a ottenere il risultato che mi aspettavo. Lo cito qui solo per completezza di informazione.

Un altro pacchetto che possiamo usare, quello che ho usato io per ottenere l'animazione di cui parlo, è animate. Questo pacchetto serve a ottenere animazioni a partire da una serie di fotogrammi di grafica vettoriale o immagini raster, o da grafica *inline*. Basato su JavaScript, consente animazioni molto versatili.

Appena incluso il pacchetto nel documento, abbiamo a disposizione i due comandi `\animategraphics` e `\animateinline`. Mentre il secondo serve a generare animazioni a partire da materiale composto al momento, il primo usa immagini preparate in precedenza. Visto che i fotogrammi erano già pronti, `\animategraphics` è proprio il comando che ho usato. La sua sintassi è `\animategraphics[<opzioni>]{<fotogrammi per secondo>}{<nome comune dei file>}{<primo>}{<ultimo>}`.

Nella fattispecie, nella presentazione ho usato `\animategraphics[autoplay,loop]{6}{tcomma-}{0}{12}`.

## 7 Conclusioni

Ho curato la realizzazione della presentazione qui studiata con TEXLive 2007. È possibile che le versioni successive abbiano superato, o lo faranno a breve, i pochi problemi esposti. Gli amanti di LATEX non ne saranno certo scoraggiati, e indirizzati verso programmi di presentazione alternativi.

## Riferimenti bibliografici

beamer (2009). *The beamer class.*

PDF (2008). *Document management — Portable document format — Part 1: PDF 1.7.* Adobe Systems Incorporated, 1ª edizione.

PIGNALBERI, G. (2009). «combelow: abbasso i segni diacritici di serie b». In *ArsTEXnica*. GuIT, Pisa, 8, pp. 70–75.

▷ Gianluca Pignalberi
g dot pignalberi at alice dot it

# Managing Printed and Online Versions of Large Educational Documents

*Jean-Michel Hufflen*

## Abstract

We have developed a LaTeX 2ε package — pfa-macros — usable for both presentational education, concerning 'classical' students, and distance education, where most of a curriculum is performed by means of online documents. First, we explain why requirements for educational documents are not the same for these two ways of teaching. Then we show why our package allows us to manage two versions—printed and online—of the same textbook.

**Keywords** Presentational education, distance education, course text, online course, case study, LaTeX, PDF, pdfLaTeX, pfa-macros package.

## Sommario

Abbiamo sviluppato un pacchetto LaTeX 2ε—pfa-macros—utilizzabile sia per l'istruzione frontale, riguardante gli studenti 'classici', che l'istruzione a distanza, dove la maggior parte del percorso è svolta per mezzo di documenti online. Per prima cosa spieghiamo perché i requisiti dei documenti di istruzione non sono gli stessi per questi due stili di insegnamento. Poi mostriamo perch'é il nostro pacchetto ci permette di gestire due versioni —stampata e online—dello stesso libro di testo.

**Parole chiave** Istruzione frontale, istruzione a distanza, libro di testo, corso online, caso di studio, LaTeX, PDF, pdfLaTeX, pacchetto pfa-macros.

## 0 Introduction

The Internet has revived *correspondence education*: now many network tools are widely used within this field: electronic mail, mailing lists, forums, online documents available *via* the Web, etc. Moreover, the term 'correspondence education' seems to be quite old, since it appears to be related to 'classical' letters sent and delivered by post, so nowadays the term '*distance education*' is preferred. As result of greater and greater interest in distance education, most universities in the world have increased such offerings. An example of a French academic institution delivering distance education is the CTU[1], part of the University of Franche-Comté, located at Besançon. The CTU allows students to get all the units required for a master in Computer Science. Of course, the University of Franche-Comté still provides curricula in presentational education —for students who physically attend 'classical' lectures, exercises and lab classes—which remains the 'traditional' way of teaching. Obviously some teaching units are common to the two curricula of presentational and distance education.

In this article, we show how some new LaTeX commands allow us to manage the different parts of a single document's body, for presentational students as well as distance ones. In fact, these parts have been initially written as chapters and appendices of a textbook for presentational students. Later, they have been reused and maintained as we explain in Section 1. Then Section 2 goes thoroughly into requirements about educational documents and shows that requirements for textbooks for presentational students and online documents for distance students are not the same. Our commands have been grouped into a package pfa-macros[2]: Section 3 describes the broad outlines of it. Finally, Section 4 discusses some alternative solutions.

A report about this work has already been published as (Hufflen, 2010), but within a general conference about computer-aided education, so there we reduced technical details about LaTeX's features as far as possible. The present article gives more descriptions about our package's functionalities. However, reading it only requires knowledge of LaTeX as an end user.

## 1 History

As an Assistant Professor, we are in charge of some teaching units. In particular, one is devoted to *functional programming*[3]. In fact, it is entitled *Advanced Functional Programming*, PFA for short[4], since it is attended by graduate students —4th-year university degree in computer science— that is, students who already have experience in programming. The 'philosophy' and contents of this unit are described in (Hufflen, 2009). Let us

1. ***C**entre de **T**élé-enseignement **U**niversitaire*, that is, *University Centre for Teleteaching.*

2. Available online: `http://lifc.univ-fcomte.fr/home/~jmhufflen/latex-etc/pfa-macros.sty`.

3. *Functional programming* emphasises application of functions, whereas *imperative programming*—the paradigm implemented within more 'traditional' languages such as Pascal (Wirth, 1971) or C (Kernighan and Ritchie, 1988)— emphasises changes in state.

4. ***P**rogrammation **F**onctionnelle **A**vancée*, in French. Our package's name—pfa-macros—originates from this acronym.

just recall briefly that students actually practise only one programming language within this unit—Scheme (Springer and Friedman, 1989)—but alternative implementations of functional programming concepts are exemplified using other functional programming languages, such as Common Lisp[5] (Steele et al., 1990), Standard ML[6] (Paulson, 1996), caml[7] (Leroy et al., 2010), and Haskell[8] (Peyton Jones, 2003). Other comparisons with modern object-oriented languages such as Java (2008), C++ (Stroustrup, 1991), and C# (Microsoft Corporation, 2001) are also given. In addition, we show in (Hufflen, 2009) that some examples are demonstrated using TEX's language. As a consequence, a textbook based on what is taught within this unit should include many excerpts of programs using various languages.

Setting up this teaching unit pfa began in spring 1997 and the first version of our printed document (Hufflen, 1997) came out in August 1997, with a pre-version of a short additional document (Hufflen, 1998) devoted to an introduction to the λ-calculus (Church, 1941), the common root of functional programming languages[9].

When the master's for distance students was launched, for the academic year 2004–2005, its curriculum obviously resembled the master's in presentational education. But a unit common to these two curricula was not necessarily handled by the same teacher. Concerning us, we have been in charge of the pfa unit within both presentational and distance education, but this arrangement does not hold true for all the units. So we were interested in a method that would allow us to derive the two versions—printed and online—from the same source files. Such a *modus operandi* would ease the maintenance of our documents. For example, some slight mistakes—especially typing ones—should be fixed once, and we wished to add more examples. More ambitiously, the version of standard Scheme changed, from (Clinger et al., 1991) to (Kelsey et al., 1998), so we ought to adapt some existing texts and examples[10].

## 2 Different requirements

The document (Hufflen, 1997) consists of six chapters. Each chapter includes exercises, given with model solutions. These chapters are followed by several appendices—making precise some extra information or devoted to lab class exercises done by students—and a rich 'Bibliography' section. The whole document is approximately 400 pages long. It can be viewed as a textbook, even if its dissemination is limited to this unit's students[11]. The students are progressively given the successive parts of this document, but it is organised as a whole, with precise architecture: cross-references are widely used throughout it. Of course, it contains not only text—in the sense of successive paragraphs—but also many examples of programs and some mathematical formulas, even if it is not really a textbook in mathematics.

### 2.1 Requirements about typography

When the first teaching units were launched in distance education, teachers were obviously asked to install online documents on the Web. Some teachers wrote documents using html[12]. However, such a choice seemed to us unsuitable for scientific documents: the look of resulting Web pages depends on the browser used; in addition, formatting mathematical formulas and program fragments often results in poor-quality output. We could have used some converters from LATEX source texts to html pages,[13] which may use *images* to insert fragments whose conversion to html is difficult, e.g., mathematical formulas. However, even if these converters allow the output's quality to be improved, in comparison with direct writing in html, authors have to adapt source texts in order for the conversion to work properly. In other words, it may be difficult to do such a task for a large document already written and formatted.

Concerning the insertion of program fragments, let us recall that this point was essential, especially about the fragments given in languages other than Scheme. We could perform some demonstrations during the lab classes of presentational students, so they could observe these other programs' behaviour. The same *modus operandi* was impossible for distant students, and it was difficult to ask them to install many compilers or interpreters. So the solution was to ask them for exercises only in Scheme—as done for presentational students—but the examples given throughout our text must be explicit, in order for these students to understand without running them. In addition, we paid much attention to the indentation of these programs and inserted some comments throughout them using special effects—e.g., slanted fonts—so they do not use `verbatim`-like environments, but are built by

5. 'Lisp' stands for **LIS**t **P**rocessor.
6. 'ml' stands for **M**eta**L**anguage.
7. **C**ategorical **A**bstract **M**achine **L**anguage.
8. Named after Haskell Brooks Curry (1900–1982).
9. Within the distance education's curriculum, this part—more related to theoretical Computer Science—has become additional in the sense that it does not belong to the topics that distance students have to assimilate. In other words, distance students are given this document as a 'cultural' additional part.
10. Later, in 2007, another change occurs, deeper, from (Kelsey et al., 1998) to (Sperber et al., 2007).
11. As abovementioned, this document is changing each academic year. If you are interested in it, you can get the most recent version, just write to the author.
12. **H**yper**T**ext **M**arkup **L**anguage. A good introduction to it is (Musciano and Kennedy, 2002).
13. Some—e.g., LATEX2`HTML`, TEX4ht—are described in (Goossens et al., 1999, Ch. 3–4).

means of `tabbing` environments[14].

From our point of view, only PDF[15], Adobe's format, (see Goossens et al., 1999, Ch. 2) offers some sufficient warranty about the quality of texts displayed on the Web. This point is also related to *communication*: when a teacher writes some formulas onto a blackboard, students see the result exactly as the teacher formats it. The same warranty is given by PDF files, not by HTML pages. So we decided to systematically use PDF files, generated by the pdfLaTeX program (Goossens et al., 1999, § 2.4). In addition, if the hyperref package (Goossens et al., 1999, § 2.3) is used, PDF files produced by pdfLaTeX can support hyperlinks, as in HTML. Let us now come to the organisational differences between texts for presentational and distance students.

### 2.2 Presentational vs distance education

Of course, distance students could not be given a single document as a huge PDF file. It is preferable for distance students to get separate medium-sized files, according to the steps of their planning. Besides, let us not forget that these files are downloaded: students cannot be asked to download a huge file again if only some typing mistakes have just been fixed. Splitting this big document into separate files induces a precise organisation of cross-reference links throughout the original version. Information redundancy should be avoided: as an example, all the parts should point to the same 'Bibliography' section, as a separate file.

In fact, a document for distance students should have two purposes. It should be printed, as any book you can read, seated in a rocking chair. It should be also be studied on a screen, in which case, the notion of 'physical' page disappears and is replaced by a *view*. Within this context, putting a big figure containing only some text in the top of a page may be confusing if you do not see the whole of the current page. We solved this problem by defining light-coloured background for online versions when figures are displayed.

Another importance difference is related to exercises. Presentational students get the successive texts at the end of each series of lectures devoted to a chapter, so model solutions may be given after each exercise, especially if this exercise has already been proposed at classes. That cannot be the same for a document devoted to distance education, because students are supposed to do exercises by means of this document, so model solutions should be grouped at the end of each chapter, or provided in separate files.

14. More precisely, we developed some functions usable within emacs, in order to build these environments from original source files.

15. **P**ortable **D**ocument **F**ormat.

## 3 The pfa-macros package

Let us assume that the chapters, sections, etc. of the two versions—printed and online—are numbered identically. Besides, LaTeX allows each chapter of a document to be associated with its own auxiliary (.aux) file, containing information solving cross-references[16]. So we can compile a chapter for the online version by using the auxiliary files of the document's other chapters of the 'presentational' version. A cross-reference written by LaTeX's `\ref` command is implemented in pdfLaTeX as an internal hyperlink, which is fine for cross-references within the same chapter. For external references, that is, cross-references to another chapter's part,we define a new command[17]:

`\pfaexternalref[`*`chapter-file`*`]{`$label_0$`}`

If the big document for presentational education is generated, this works like `\ref{`$label_0$`}`. If the chapter is generated as part of the online text, a link to the file *`chapter-file`*.pdf is put. In both cases—printed and online version—the same text is displayed or enlighted by a hyperlink. That means that $label_0$ is a label identifying a resource belonging to a file used to build *`chapter-file`*.pdf. If the complete version of the file *`chapter-file`*.pdf has already been put on the site, it can be searched. Otherwise, this PDF file is a kind of *stub* whose contents reads something like:

> This chapter will be put later.

—in French—and when the complete version is put, the hyperlink will remain the same. Of course, when we started this task, such a choice led us to look for all the occurrences of the `\ref` command and change some into `\pfaexternalref` ones[18]. We use similar technique for cross-references to footnotes belonging to another chapter :

`\pfaexternalfootnoteref[`*`chapter-file`*`]{%`
$label_0$`}`

Within the presentational version, the footnote number is displayed—within this version, all the footnotes are numbered globally, regardless of chapters, unlike LaTeX $2_\varepsilon$ standard classes such as book or report—followed by the corresponding page number. In the online version, a *short title* of this footnote is enlighted by a hyperlink pointing to the corresponding chapter. This short title is followed by the footnote's page number—chapter

16. That is the case if you use the commands `\include` and `\includeonly`, as explained in (Mittelbach et al., 2004, § 2.1.2).

17. To avoid clashes among LaTeX names, the new commands introduced by our packages are prefixed by '`\pfa...`' or '`\ifpfa...`'.

18. In practice, that was not difficult, because a good technique is to prefix labels' name by an identifier for the corresponding chapter. So the file name to be put was not difficult to supply.

pages are numbered separately and prefixed by chapters' numbers for online versions—and is created as follows:

```
\pfalabelledfootnote[short-title]{%
 label0}{body}
```

For example:

```
\pfalabelledfootnote[\emph{The Name of
the Rose}]{f-eco}{\emph{The Name of the
Rose} is a novel written by
\foreignlanguage{italian}{Umberto Eco}
(1932--) in 1980.}
```

The short title is ignored when the presentational version is typeset, and displayed as a marginal note within the online version. As shown by the previous example, *label*$_0$ is the footnote's label, *body* its contents. Let us come back to our previous example, the command:

```
\pfaexternalfootnoteref[f]{f-eco}
```

—where *f* is the file's name containing the 'eco' label—will be displayed as something like:

> Footnote ___, at the bottom of p. ___

within the printed version, the two occurrences of '___' being numbers. Within the online version, it will result in something like:

> Footnote *The Name of the Rose*, within the file *f*

the underlined part being displayed using blue-coloured characters in the 'actual' online document.

The LaTeX command `\cite` has been redefined into a new command `\pfacite`:

```
\pfacite[pre-text]{citation-key-list}
```

which works like:

```
\cite[pre-text]{citation-key-list}
```

(see Mittelbach et al., 2004, § 12.2.1) for the presentational version, and creates a hyperlink pointing to the PDF file containing the complete bibliography for the online version, the same text being displayed in both cases.

Our pfa-macros package also provide a construct for conditional texts:

```
\ifpfaclassical...
 ...% (For presentational education students.)
 \else%
 ...% (For distance education ones.)
\fi
```

In particular, this command is useful to insert an exercise's model solution immedialtely after it:

```
\newcommand{\pfaincludems}[1]{%
 \ifpfaclassical%
  \input{model-solutions/#1}\else\relax%
 \fi}
```

or at a chapter's end:

```
\newcommand{\pfafordeincludems}[1]{%
 \ifpfaclassical\relax\else%
  \input{../model-solutions/#1}%
 \fi}
```

the PDF files for online versions being generated within a subdirectory of the directory containing chapter's source files, that is why we get files for model solutions by means of the path '`../model-solutions/...`' in the last command.

## 4 Discussion—Other methods

There are other methods to let a LaTeX document to refer to a label defined within another source file, that is, an *external document*. A first example is given by some commands of the html package (Goossens et al., 1999, § 3.5.3), unsuitable for us, since this package is only interesting if you want to derive HTML pages. A second implementation of external references using hyperlinks is given by the xr package (Mittelbach et al., 2004, § 2.4.6), or better, by xr-hyper, its reimplementation tailored to work with the hyperref package. Nevertheless, this package has two drawbacks for us. First, it does not deal with bibliographical citations (`\cite` commands). Second, it cannot refer to an external label that will be defined later. To explain that, let us consider that the first chapter refers to a section of the second chapter. As long as the second chapter is replaced by a stub, the hyperlink will fail; it will work only as soon as this chapter's complete text is made public[19]. Within our system, the hyperlink always points to the second chapter's PDF file, a stub or the complete text[20].

If we had started from scratch, that is, if both the presentational and distance unit were launched at the same time, an interesting method could have been to specify our input files using XML[21], and XSLT[22] (W3C, 2007) could have been used to derive texts for LaTeX, or in XSL-FO[23] (W3C, 2006), an XML language that aims to describe high-quality print outputs[24]. Let us notice that in order for the

19. From a pedagogical point of view, such a *forward reference* is often viewed as bad. But it can occur within a footnote, or a fragment that can be skipped at first reading.

20. That could be improved in a future version: if the external label exists, the hyperlink directly points to the corresponding resource, if not, it points to a stub.

21. e**X**tensible **M**arkup **L**anguage. (Ray, 2001) is a good introduction to this meta-language.

22. e**X**tensible **S**tylesheet **L**anguage **T**ransformations.

23. e**X**tensible **S**tylesheet **L**anguage—**F**ormatting **O**bjects.

24. However, concerning this second choice, current XSL-FO processors—generating PDF files—are not complete yet,

complexity of this *modus operandi* to be mastered, we would have had to define a percise taxonomy for our source files using XML-like syntax, by means of a DTD[25] or a *schema*[26].

## 5 Students' feedback

As far as we know, students' feedback is globally positive. In fact, they quickly perceive that PDF files allow them to watch exactly what teachers want to express. Our document giving many 'cultural' complements, they told us that they were overflowed with the whole details of our text. The solution was to define typographical signs to mark up what is important ('➽➽') and what may be skipped in a first reading ('✻✻'). Likewise, we defined a command to label difficult exercises ('♠♠'). Concerning hyperlinks, those pointing to a resource belonging to the current chapter seems to be most useful, so pointing to the beginning of another chapter in the case of an external reference does not cause much trouble.

## 6 Conclusion

In our introduction, we mentioned that the Internet had revived correspondence education'. Originally, TEX was designed to typeset Donald E. Knuth's mathematical textbooks. We could think that (LA)TEX was closely related to only 'classical' textbooks, in the sense of books printed on sheets of paper, and bound.

From our point of view, the present work shows that LATEX is still unrivalled to 'intelligently' process texts for several purposes. As mentioned above, the first version of our course text came out in 1997. Then it has evolved deeply—chapters and appendices have been wholly revised—and continuously, since we have applied some changes each year. We did it successfully—in particular when we had to be conformant with new revisions of standard Scheme—so we can think that our system is reliable. Anyway, we have reached our goal: we spent a lot of time to format our document for presentational students, we successfully reused it for distance students, and we are able to maintain it for both versions.

even if they implement most of this recommendation, so using XSL-FO is interesting for experiment, but not for intensive use by students.

25. **D**ocument **T**ype **D**efinition. A DTD defines a document markup model (Ray, 2001, Ch.5).

26. Schemas have more expressive power than DTDs, in the sense that they are more modular, they allow users to define types precisely, which makes more precise the validation of a XML text with respect to a schema. In addition, this approach is more homogeneous since schemas are XML texts, whereas DTDs are not (Ray, 2001, pp. 189–193).

## Acknowledgements

I am grateful to the distance education students who addressed me very constructive criticisms; year after year, they indirectly helped me improve my tools. Karl Berry and Barbara Beeton kindly proofread an abridged version of this article, thanks to them. Thanks also to Gianluca Pignalberi, who proofread the complete version and translated the abstract and keywords in Italian.

## Remark

This is the complete version of a paper intended for EuroTEX 2010. An abridged version will be published in a TUGboat issue.

## References

Alonzo Church. *The Calculi of Lambda-Conversion.* Princeton University Press, 1941.

William D. Clinger and Jonathan A. Rees, with Harold Abelson, Norman I. Adams iv, David H. Bartley, Gary Brooks, R. Kent Dybvig, Daniel P. Friedman, Robert Halstead, Chris Hanson, Christopher T. Haynes, Eugene Edmund Kohbecker, Jr., Donald Oxley, Kent M. Pitman, Guillermo Juan Rozas, Guy Lewis Steele, Jr., Gerald Jay Sussman and Mitchell Wand. Revised report[4] on the algorithmic language scheme. ACM *Lisp Pointers*, 4(3), July 1991.

Michel Goossens and Sebastian Rahtz, with Eitan M. Gurari, Ross Moore and and Robert S. Sutor. *The LATEX Web Companion.* Addison-Wesley Longman, Inc., Reading, Massachusetts, May 1999.

Jean-Michel Hufflen. Programmation fonctionnelle avancée. notes de cours et exercices. Polycopié. Besançon, July 1997.

Jean-Michel Hufflen. Introduction au λ-calcul (version révisée et étendue). Polycopié. Besançon, February 1998.

Jean-Michel Hufflen. Using TEX's language within a course about functional programming. MAPS, 39:92–98, August 2009. In EuroTEX 2009 conference.

Jean-Michel Hufflen. Recycling previous documents for distance education. In *Proc.* CSEDU *2010*, volume 1, pages 469–472, Valencia, Spain, April 2010.

*Java Technology.* `http://java.sun.com`, March 2008.

Richard Kelsey, William D. Clinger and Jonathan A. Rees, with Harold Abelson, Norman I. Adams iv, David H. Bartley, Gary Brooks, R. Kent Dybvig, Daniel P. Friedman,

Robert Halstead, Chris Hanson, Christopher T. Haynes, Eugene Edmund Kohlbecker, Jr, Donald Oxley, Kent M. Pitman, Guillermo J. Rozas, Guy Lewis Steele, Jr, Gerald Jay Sussman and Mitchell Wand. Revised[5] report on the algorithmic language Scheme. HOSC, 11 (1):7–105, August 1998.

Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, 2nd edition, 1988.

Xavier Leroy, Damien Doligez, Jacques Garrigue, Didier Rémy and Jéróme Vouillon. *The Objective Caml System. Release 3.12 Documentation and User's Manual*. `http://caml.inria.fr/pub/docs/manual-ocaml/index.html`, 2010.

Microsoft Corporation. *Microsoft C# Specifications*. Microsoft Press, 2001.

Frank Mittelbach and Michel Goossens, with Johannes Braams, David Carlisle, Chris A. Rowley, Christine Detig and Joachim Schrod. *The LaTeX Companion*. Addison-Wesley Publishing Company, Reading, Massachusetts, 2 edition, August 2004.

Chuck Musciano and Bill Kennedy. HTML *&* XHTML*: The Definitive Guide*. O'Reilly & Associates, Inc., 5 edition, August 2002.

Lawrence C. Paulson. ML *for the Working Programmer*. Cambridge University Press, 2 edition, 1996.

Simon Peyton Jones, editor. *Haskell 98 Language and Libraries. The Revised Report*. Cambridge University Press, April 2003.

Erik T. Ray. *Learning* XML. O'Reilly & Associates, Inc., January 2001.

Michael Sperber, William Clinger, R. Kent Dybvig, Matthew Flatt and Anton van Straaten, with Richard Kelsey, Jonathan Rees, Robert Bruce Findler and Jacob Matthews. *Revised$^{5.97}$ Report on the Algorithmic Language Scheme*. `hhttp://www.r6rs.org`, June 2007.

George Springer and Daniel P. Friedman. *Scheme and the Art of Programming*. The MIT Press, McGraw-Hill Book Company, 1989.

Guy Lewis Steele, Jr., with Scott E. Fahlman, Richard P. Gabriel, David A. Moon, Daniel L. Weinreb, Daniel Gureasko Bobrow, Linda G. DeMichiel, Sonya E. Keene, Gregor Kiczales, Crispin Perdue, Kent M. Pitman, Richard Waters and Jon L White. COMMON LISP. *The Language. Second Edition*. Digital Press, `http://www.cs.cmu.edu/Groups/AI/html/cltl/cltl2.html`, 1990.

Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 2 edition, 1991.

W3C. *Extensible Stylesheet Language (*XSL*). Version 1.1*. `http://www.w3.org/TR/2006/REC-xsl11-20061205/`, December 2006. W3C Recommendation. Edited by Anders Berglund.

W3C. XSL *Transformations (*XSLT*). Version 2.0*. `http://www.w3.org/TR/2007/WD-xslt20-20070123`, January 2007. W3C Recommendation. Edited by Michael H. Kay.

Niklaus Wirth. The programming language pascal. *Acta Informatica*, 1(1):35–63, 1971.

▷ Jean-Michel Hufflen
LIFC (EA CNRS 4157)
University of Franche-Comté
16, route de Gray
25030 BESANÇON CEDEX
FRANCE
`jmhufflen at lifc dot univ-fcomte dot fr`

# Eventi e novità

### TUG 2011

TUG 2011 will be held in Cairo, Egypt, tentatively from November 14–17, 2011, including an excursion. Hossam Fahmy is the chief organizer. More information will follow.

`http://www.tug.org/tug2011/`

### 5th International ConTEXt Meeting

The fifth ConTEXt meeting will take place in Porquerolles, France on September 19–24, 2011

You are cordially invited to join the 5th ConTEXt meeting. The meeting will host ConTEXt and LuaTEX developers and users and gives you the opportunity to present your results, experiences and ideas on future development. The talks will be followed-up by tutorials on different ConTEXt and LuaTEX techniques.

*Call for submissions*

As in previous years, anything at all related to ConTEXt that you would like to share is an acceptable subject for a presentation, tutorial, discussion, Q&A session, demonstration, workshop, recital, sketch, or sermon.

The programme committee is Hans Hagen and Taco Hoekwater, feel free to email them with hints and ideas. Because we want something interesting on the website, abstracts should be sent in before May 31st or even now, before opening the registration.

*Dates and deadlines*

- November 1, 2010, beginning of registration.
- Registration is open until we reach the full capacity.
- September 19-24, 2011, 5th ConTEXt Meeting.

`http://meeting.contextgarden.net/2011/`

### TEX Live 2010

In 2010, the default version for PDF output is now 1.5, enabling more compression. This applies to all the TEX engines when used to produce PDF and to `dvipdfmx`. Loading the `pdf14` LATEX package changes back to PDF 1.4, or set `\pdfminorversion=4`.

pdf(LA)TEX now automatically converts a requested Encapsulated PostScript (EPS) file to PDF, via the `epstopdf` package, when and if the LATEX `graphics.cfg` configuration file is loaded, and PDF is being output. The default options are intended to eliminate any chance of hand-created PDF files being overwritten, but you can also prevent `epstopdf` from being loaded at all by putting `\newcommand{\DoNotLoadEpstopdf}{}` (or `\def...`) before the `\documentclass` declaration. It is also not loaded if the `pst-pdf` package is used. For more details, see the `epstopdf` package documentation.

A related change is that execution of a very few external commands from TEX, via the `\write18` feature, is now enabled by default. These are commands are `repstopdf`, `makeindex`, `kpsewhich`, `bibtex`, and `bibtex8`; the list is defined in `texmf.cnf`. Environments which must disallow all such external commands can deselect this option in the installer, or override the value after installation by running `tlmgr conf texmf shell_escape 0`.

Yet another related change is that BibTeX and Makeindex now refuse to write their output files to an arbitrary directory (like TEX itself), by default. This is so they can now be enabled for use by the restricted `\write18`. To change this, the `TEXMFOUTPUT` environment variable can be set, or the `openout_any` setting changed.

XETEX now supports margin kerning along the same lines as pdfTEX. (Font expansion is not presently supported.)

By default, `tlmgr` now saves one backup of each package updated (`tlmgr option autobackup 1`), so broken packages updates can be easily reverted with `tlmgr restore`. If you do post-install updates, and don't have the disk space for the backups, run `tlmgr option autobackup 0`.

New programs included: the pTEX engine and related utilities for typesetting Japanese; the BibTeXU program for Unicode-enabled BibTeX; the `chktex` utility for checking (LA)TEX documents; the `dvisvgm` DVI-to-SVG translator.

Executables for these new platforms are now included: `amd64-freebsd`, `amd64-kfreebsd`, `i386-freebsd`, `i386-kfreebsd`, `x86_64-darwin`, `x86_64-solaris`.

A change in TEX Live 2009 that we failed to note: numerous TEX4ht-related executables were removed from the binary directories. The generic `mk4ht` program can be used to run any of the various `tex4ht` combinations.

Finally, the TEX Live release on the TEX Collection DVD can no longer be run live (oddly enough). A single DVD no longer has enough room. One benefit is that installation from the physical DVD should now be much faster.

Questa rivista è stata stampata
presso Logo S.r.l. Servizi di Stampa Digitale, Borgoricco (PD)
su carta ecosostenibile Vision Trend White
prodotta da Steinbeis Temming Papier GmbH & Co.


## ArsTEXnica – Call for Paper

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 entro e non oltre il 14 Marzo 2011. 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 d'esempio (.tex).

Gli articoli potranno trattare di qualsiasi argomento inerente al mondo di LATEX e non dovranno necessariamente essere indirizzati ad un pubblico esperto. In particolare tutorial, 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 `http://www.guit.sssup.it/arstexnica`, 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.

Chiunque, poi, volesse collaborare con la rivista a qualsiasi titolo (recensore, revisore di bozze, grafico, etc.) può contattare la redazione all'indirizzo `arstexnica@sssup.it`.

# ArsTEXnica

Rivista italiana di TEX e LATEX

*Numero 10, Ottobre 2010*



00010
9 771828 236001