ConTEXt:un moderno sistema di packages basato su TEX.

Scarso Luigi
luigi.scarso@logosrl.it

October 5, 2005

Abstract

In questo articolo viene presentato ConTEXt, un insieme coodinato di packages scritti in TEXper la produzione di documenti di alta qualità tipograrafica soprattutto in ambito non-matematico. Vengono illustrate, mediante semplici esempi, l'attività di creazione di documenti e la possibilità di integrazione in un workflow per la composizione automatica.

Contents

1 Introduzione

Per un utente TEX probabilmente la definizione più familiare di ConTEXt è la seguente: un insieme di macro raggruppate in modo coerente in files `*.tex` che, una volta compilate, producono un formato `fmt` utilizzabile per l'attività di typesetting generica, in maniera analoga ad altri sistemi di macro noti come LATEX o $\mathcal{AMS}$TEX.

Il formato usuale si chiama `cont-en.fmt` prodotto da pdfe-TEX; come è noto, quest'ultimo è una compilatore del linguaggio ε-TEX che produce file oggetto sia nel formato standard dvi che nel formato pdf (ε-TEX a sua volta è una estensione del linguaggio TEX: il compilatore ε-TEX può essere utilizzato sia in modalità tradizionale (TEX puro) che estesa). In linea di principio quindi ConTEXt può essere utilizzato per produrre un file dvi dal quale attraverso opportuni converters come `dvips`, `dvipdf` o `dvipdfmx` ottenere un file pdf.

La strada alternativa, probabilmente la più usata, consiste nell'utilizzare ConTEXt per produrre direttamente file in formato pdf utilizzando pdfe-TEX in modalità extended (i.e ε-TEX come estensione del TEX).

In una visione a più alto livello, ConTEXt si presenta con diverse caratteristiche:

▶ ad un *utente generico:*

- rende disponibile un ampio insieme di ambienti per la composizione di documenti *già inclusi nel formato* (ambienti standard), limitando la necessità di caricare runtime altri packages.
  Gli ambienti standard sono fortemente configurabili, in modo da poter confinare le personalizzazioni in files gestibili e riusabili;
- semplifica e facilita il ciclo codice-compilazione-verifica, sia attraverso l'uso di script, sia mediante alcuni ambienti standard dedicati al visual-debug;

▶ ad un *macro writer:*

- presenta l'insieme di macro suddivise in moduli coerenti e documentati;
- fornisce un layer intermedio di macro robuste, importante per esempio per sfruttare le caratteristiche peculiari del pdf (eg. `javascript`) o per la gestione dei fonts.

▶ per la *gestione documentale:*

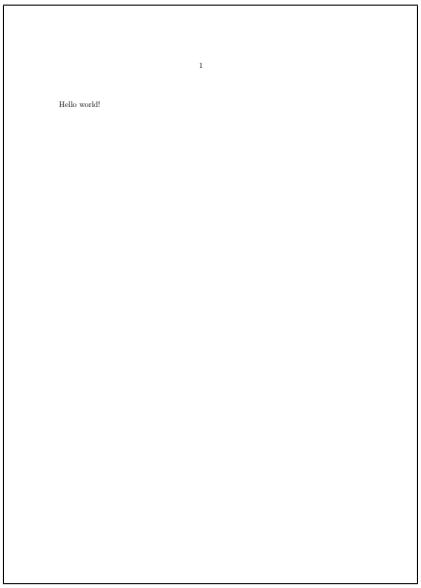
- facilita la gestione delle divese fonti che concorrono a formare il documento finale, nononchè la produzione per i diversi intenti (stampa, prestampa, video)
- si inserisce in un workflow, soprattutto se basato su XML;

▶ per la *compatibilità:*

- può eseree utilizzato con diversi *backend* (`dvips, dvpdfmx` ..., cfr. sopra) o con diversi TEXengine come Omega o Aleph.

In definitiva, ConTEXt si propone come un *moderno sistema di produzione documentale di alta qualità basato sul TEX.*

Ecco due semplici esempi:



a.tex←−

```
\starttext
Hello world!
\stoptext
```

⇐= `$>texexec --pdf a.tex`

Il file `a.tex` contiene l'unica linea

`\starttext Hello world!\stoptext`

ed è processato mediante il comando `texexec` con lo switch `--pdf`, che abilita il backend pdfe-TEX(`texexec` è uno script `perl` che fa parte della distribuzione standard di ConTEXt). Il pdf prodotto `a.pdf` ha una unica pagina in formato A4, con il numero di pagina sulla testatina superiore (questi ed altri paramentri di default sono modificabili in fase di generazione del formato). La figura è in scala 1:4.

a.tex←−

```
\starttext
\startTEXpage
Hello world!
\stopTEXpage
\stoptext
```

Hello world! ⇐= `$>texexec --pdf a.tex`

Il pdf prodotto ha una pagina di dimensioni 61.7774×14.4126 bp; la figura è in scala 1:1 .

Naturalmente, è possibile combinare i due esempi in un pdf con 2 pagine di dimensione diversa:

```
\starttext
Hello world!
\page
\startTEXpage Hello world!\stopTEXpage
\stoptext
```

2 Capire ConTEXt

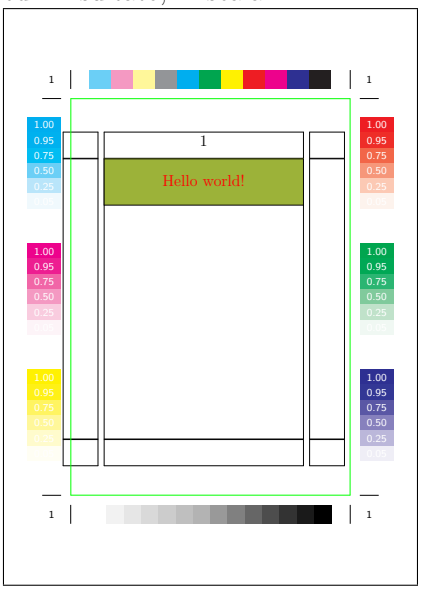
Ci proponiamo di produrre un pdf in formato A7 imposto centralmente su un foglio A6, con i crocini per il taglio ed i tests dei colori; il contenuto è un rettangolo colorato di dimensioni `area_testo` × `3em`. Assumiamo come font `cmr`, un colore di fondo in quadricomia Strange color con valori `ciano=30%, magenta=0%, giallo=90%, nero=20%` ed un colore in primo piano Strange color2 in rgb con valori `rosso=100%, g=0%, b=0%`. Siamo in fase di debug, e vogliamo avere la sensazione degli elementi in gioco.
Ecco un possibile codice:

```
\setupcolors[state=start]
\setuppapersize[A7][A6]
\setuplayout[location=middle,
             marking=color]
\definecolor[Strange color][c=0.3,m=0,y=0.9,k=0.2]
\definecolor[Strange color2][red]

\showframe

\starttext
\framed[width=broad,height=3em,
        background=color,
        backgroundcolor={Strange color},
        foregroundcolor={Strange color2}]
{Hello world!}
\stoptext
```

ed il risultato, in scala 1:2 :



Già ad una prima lettura risulta chiara un sezione di setup compresa tra l'inizio del codice e `\starttext` ed il contenuto del testo , compreso tra `\starttext` e `\stoptext` : questa suddivisione logica (setup,content) è una "buona norma"

seguita dalla maggior parte di utenti ConTEXt, ed è del resto pacificamente accettata anche da utenti di altri macro-packages.
Dal punto di vista di un *utente* i comandi (" macro" in gergo TEX) di ConTEXt hanno in genere o un *prefisso significativo* oppure un nome significativo e sono seguite da una serie di argomenti *variabili*.
Ovvero:

- i prefissi `\setup`, `\define` identificano macro di setup;
- il prefisso `\show` una macro per il debug visuale ;
- i prefissi `\start`, `\stop` racchiudono un ambiente per il contenuto;
- *comando* una macro con un preciso significato, solitamente usata nel contenuto.

Per quanto riguarda gli argomenti di una macro, questi sono in prima istanza del tipo $chiave_j$=$valore_j$ "impacchettati" tra parentesi quadre e separati da virgole `,` i.e.

`[`$chiave_1$=$valore_1$`,` $chiave_2$=$valore_2$`,` $chiave_3$=$valore_3$`,...]` .

In questo caso la varietà è maggiore:

- è possibile avere macro con 2 o più "pacchetti" di argomenti;
- a volte *valore* è assente;
- a volte una chiave ha più valori: $chiave_j$=$\{valore_{j1}$,$valore_{j2}\}$;
- è possibile avere macro senza alcun argomento;
- *valore* può essere a sua volta il nome di una macro che accetta un numero prefissato di argomenti, anche zero. In questo caso sarà indicata con *valore*, come usuale per le macro, ma senza argomenti (nella documentazione, questo caso è indicato con $chiave_j$=*\valore#1* per una macro con un argomento);

Rivediamo ora il codice:

```
\setupcolors[state=start]
```

Macro di setup: attiva la gestione dei colori. È indicata una sola coppia (chiave,valore), le eventuali altre coppie hanno un valore di default.

```
\setuppapersize[A7][A6]
```

Setup: definisce la pagina e il foglio di carta che contiene la pagina.
È una macro con due argomenti del tipo solo chiave: ma può essere chiamata ad esempio anche nel modo seguente:`\setuppapersize[A4,landscape][A2,landscape]`.

```
\setuplayout[location=middle,
             marking=color]
```

Setup: definisce il layout, indicando la posizione mediana dell A7 sull'A6 ed attivando i crocini. Macro con un argomento , con più coppie (chiave, valore).

```
\definecolor[Strange color][c=0.3,m=0,y=0.9,k=0.2]
\definecolor[Strange color2][red]
```

Setup: definiscono due colori. Notare il nome dei primi argomenti: vi sono spazi e numeri. La seconda macro definisce un colore sulla base di un altro colore:

nel formato vi è cioè una macro del tipo `\definecolor[red][r=1,g=0,b=0]` e questo colore viene riutilizzato.

`\showframe`
Setup: macro di debug. Mostra la gabbia di composizione.

`\starttext`
`\stoptext`
Contenuto: la sezione che apre e chiude il contenuto. Macro senza argomenti.

```
\framed[width=broad,height=3em,
        background=color,
        backgroundcolor={Strange color},
        foregroundcolor={Strange color2}]
{Hello world!}
```

Contenuto: disegna un rettangolo colorato. `\framed` ha una argomento di inizializzazione "pacchettizzato" ed una argomento oggetto `Hello world!` racchiuso tra le parentesi { e } .
Anche `\framed{Hello world!}` è una macro valida e produce :

Hello world!

◄►Può essere interessante per il *macro writer* dare uno sguardo alla implementazione di `\framed`. La macro si trova nel modulo `core-rul.tex`:

```
\unexpanded\def\framed
  {\bgroup
   \presetlocalframed[\??ol]%
   \dodoubleempty\startlocalframed[\??ol]}
```

Tralasciamo per un attimo gli aspetti di località gestiti dallo "strano" argomento `[\??ol]` (la macro ammette come argomento oggetto altre macro `\framedtext`, in modo da creare riquadri contenenti riquadri) e consideriamo `\dodoubleempty`: questa macro (è in `syst-gen.tex`) è responsabile di trattare macro con 2 argomenti opzionali:
`\dodoubleempty \command` permette cioè di gestire

```
\command[#1][#2]
\command[#1][]
\command[][]
```

Nel nostro caso il primo argomento è `[\??ol]`; il secondo può non esserci, ma se esiste è racchiuso tra parentesi quadre, eventualmente vuote. È un errore non specificare l'argomento oggetto.
`\??ol` è una macro: i caratteri solitamente proibiti sono stati sprotetti racchiudendo tutto il modulo tra la coppia `\unprotect` – `\protect`, una procedura di routine per i moduli.

Come si può intuire, le macro come `\dodoubleempty` sono ampiamente usate ed il modulo `syst-gen.tex` ne contiene di diversi tipi, per cui una sua lettura è fortemente consigliata.
Nello spirito literate programming la documentazione è intrecciata nel modulo stesso ed è ottenibile mediante lo script `texexec`:

```
texexec --pdf --modu syst-gen.tex
```

A questo punto, ad un *utente* può sorgere la domanda:
"I comandi (o macro che dir si voglia) sono tutti in inglese ?"

Sorprendentemente la risposta è: "No, è possibile generare un formato in lingua italiana".
La generazione del formato è semplice:
`texexec --make it`
ConTEXt chiama interfaccia l'insieme dei nomi delle macro e dei messaggi di sistema; il comando di cui sopra crea l'interfaccia italiana, per cui ad esempio `\definepapersize` diventa ora `\definiscidimensionicarta`, e coerentemente i nomi delle chiavi e valori. Un documento con l'interfaccia italiana viene processato con lo switch `interface`: `texexec --pdf --interface=it` *documento*.

Attenzione, però: è possibilie definire una interfaccia in italiano per comporre ad esempio documenti in lingua inglese, e viceversa; l'interfaccia *non* determina la lingua (o le lingue) del documento.
Questa pregevole caratteristica si basa sul notevole lavoro svolto da G. Bilotta, ma i volontari sono benvenuti .

2.1 Layout e fonts

In questo esempio, dobbiamo preparare un documento per la stampa in formato non standard, con margini dati e con una famiglia di fonts coerente. Il documento ha una pagina di 20cm×20cm margine superiore 'al vivo' alto 2cm, margine inferiore 2cm con uno spazio dal bordo di 3mm, margini laterali 1.5cm. L'area testo è separata di 5mm dai margini; il numero di pagina è centrato sulla testina superiore.
La famiglia di fonts è la tripla (Times,Helvetica,Courier), la classica (*Serif, Sans, Monospaced*).
La pagina è imposta centralmente su un A4; col termine 'al vivo' si indica il fatto che la testatina superiore supererà di qualche millimetro i crocini (3mm è una misura comune), in modo che al taglio l'eventuale colore non presenti 'bianco' dovuto all'approssimazione della macchina.

La pagina standard di ConTEXt è un A4 imposto su un A4; le aree della pagina sono identificabili come spazio superiore (topspace), testatina superiore (header), spazio per la fascicolazione (backspace), margini laterali (margin), area testo, testatina inferiore (footer) e spazio inferiore (bottomspace).

Ecco un possibile codice (cfr. fig. 1):

```
\definepapersize[20x20][width=20cm,height=20cm]
\setuppapersize[20x20][A4]
\setuplayout[location=middle,
             topspace=-3mm,
     %%topspace e' negativo!
             header={\dimexpr 2cm-\topspace\relax},
             headerdistance=5mm,
             backspace=2cm,
             margin=1.5cm,
             margindistance=5mm,
             height=middle,width=middle,
             footer=2cm,
             footerdistance=5mm,
             bottomspace=5mm,
             marking=on]
```

```
\showlayout
%% hmm, il numero di pagina...
%% per il test
\starttext \ \stoptext
```

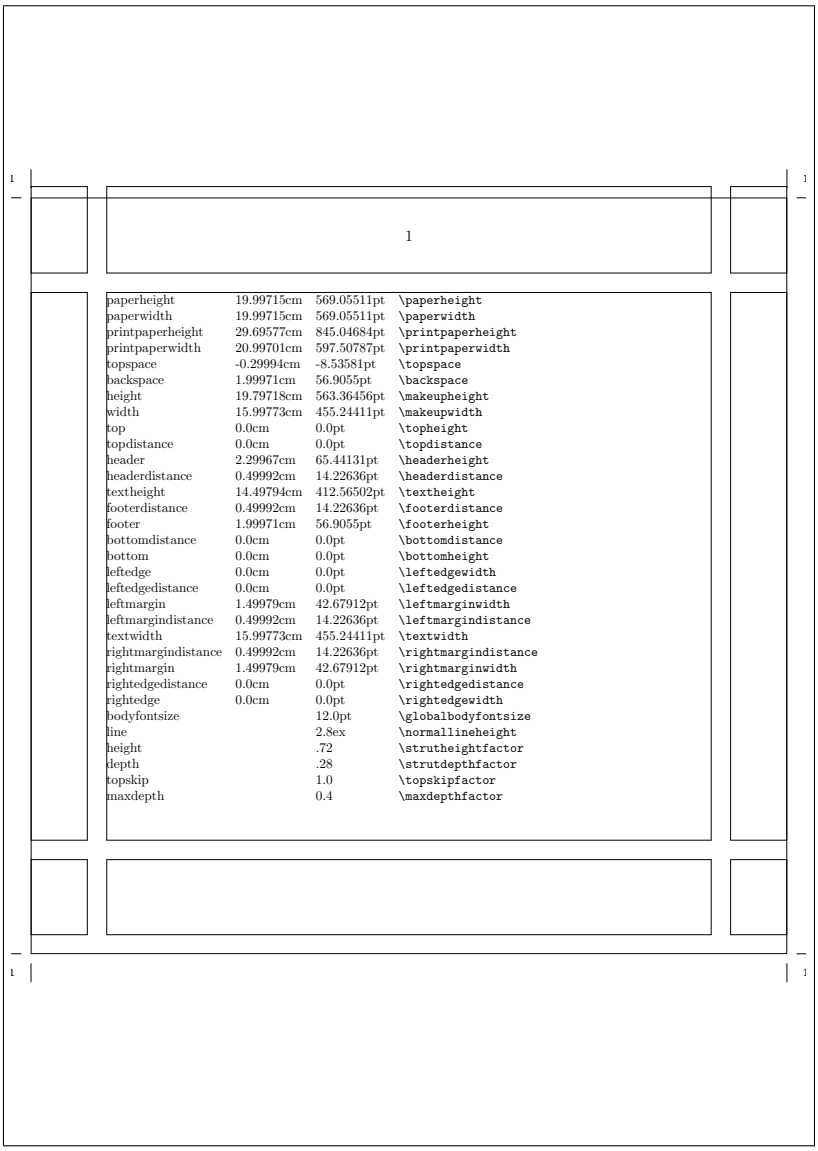


Figure 1: il nostro layout, in scala 1:2 .

A questo punto codice dovrebbe essere chiaro: l'unico trucchetto consiste nel "tirare in alto" lo spazio superiore di 3mm e di aumentare la testatina superiore di 3mm : `header={\dimexpr 2cm-\topspace\relax}` ha proprio questo significato.

◄►`\dimexpr 2cm-\topspace\relax` è una estensione ε-TEX per calcolare espressioni con dimensioni.

Se il lettore prova questo codice, non gli sfuggirà il fatto che la numerazione di pagina viene tagliata, un effetto certamente poco gradevole, e penserà ad un comando tipo `setuppagenumbering[...]`: ed effettivamente

```
\setuppagenumbering[location=header,
                    command=\MyPageNumber]
\def\MyPageNumber#1{%
\vbox to \dimexpr \headerheight \relax{%
\vfil
\vbox to \dimexpr \headerheight +\topspace\relax{%
\vfil
\centerline{#1}
\vfil}}}
```

`command=\MyPageNumber` indica che la chiave `command` attiva la macro `\MyPageNumber` definita altrove (sotto in questo caso) ; nella documentazione troviamo

```
\setuppagenumbering[..,..=..,..]
%% molte altre chiavi/valori prima di questi...
location    header footer left right middle margin marginedge inleft inright
command    \command#1
```

e sembra ragionevole supporre che l'argomento `#1` sia il numero di pagina corrente; la macro `\MyPageNumber` si prende cura di sistemare il numero nella posizione giusta.
◄►per evitare di ridefinire macro di sistema è buona norma che le proprie macro inizino con lettere maiuscole.

◄►Con i fonts entriamo in uno degli aspetti più intriganti di ConTEXt, in cui sono necessarie competenze del *macro writer* .
Se letto attentamente, la frase *"La famiglia di fonts è la tripla (*Times, Helvetica, Courier*), la classica (Serif, Sans, Monospaced)"* dice poco: quale font indica "Times"? Con quale encoding (mappa dei nomi simbolici verso i glifi) ? Quale famiglia usare per il documento, e con quale corpo ? Come impiegare le 3 famiglie ?
Il termine "classico" suggerisce che Times sia il font del documento, probabilmente con corpo 10pt, mentre Helvetica venga utilizzato per i titoli e Courier per parti di contenuto con significato speciale come listati; è una scelta ragionevole, e quindi seguiremo questi suggerimenti.

ConTEXt attua per i fonts una struttura a strati:

1. uno strato a diretto contatto con il file del font che si preoccupa di inserire il file nel sistema, generare i files ausiliari corretti e permettere di utilizzare il font all'interno del documento. In questo livello si utilizza principalmente il comando `texfont`, uno script perl altamente specializzato. In sostanza il compito di `texfont` è di rendere disponibile un nome simbolico associato al file fisico del font in modo del tutto analogo a quanto avviene col TEX;
2. uno strato in cui il nome simbolico viene mappato in un nome più significativo ed ad alto livello. È possibile avere più mappe dello stesso nome simbolico verso altri nomi significativi;
3. uno strato in cui i nomi significativi vengono raggruppati per fornire una famiglia coerente e le famiglie raggruppate per creare una collezione.

Torniamo al nostro esempio, e supponiamo di possedere l' insieme dei fonts richiesti (per semplicità di `type1`, con le relative metriche `afm`) e di doverli installare nel sistema.

Assumiamo che i font provengano dal vendor `xyz` e che la codifica sia texnansi.
Il primo passo consiste nel creare la directory `myfont` (il nome non è rilevante) ed al suo interno altre 3 directory `Times`, `Helvetica`, `Courier`; in ciascuna directory si copiano i files relativi (ie., nella la directory `Times` i files relativi alla variante neretto, romana, inclinata etc. della famiglia Times e similmente per le altre famiglie) .
Posizioniamoci all'interno di `myfont` ed installiamo i fonts nel sistema con lo script `texfont`:

- `texfont --vendor=xyz --encoding=texnansi --sourcepath=Times --collection=times --makepath --install`
- `texfont --vendor=xyz --encoding=texnansi --sourcepath=Helvetica --collection=helvetica --makepath --install`
- `texfont --vendor=xyz --encoding=texnansi --sourcepath=Courier --collection=courier --makepath --install`

Se tutto è andato a buon fine nella directory corrente dovremmo avere 3 files `*.map` del tipo seguente (eg. per i fonts nella directory `Times`):

```
% This file is generated by the TeXFont Perl script.
%
% You need to add the following line to pdftex.cfg:
%
%   map +texnansi-xyz-times.map
%
% Alternatively in your TeX source you can say:
%
%   \pdf    {+texnansi-xyz-times.map}
%
% In ConTeXt you can best use:
%
%   \loadmapfile[texnansi-xyz-times.map]

texnansi-raw-utmb8a NimbusRomNo9L-Medi 4 < utmb8a.pfb texnansi.enc
texnansi-raw-utmbi8a NimbusRomNo9L-MediItal 4 < utmbi8a.pfb texnansi.enc
texnansi-raw-utmr8a NimbusRomNo9L-Regu 4 < utmr8a.pfb texnansi.enc
texnansi-raw-utmri8a NimbusRomNo9L-ReguItal 4 < utmri8a.pfb texnansi.enc
```

Abbiamo completato il punto 1) di cui sopra: possiamo testare il font con il nome simbolico codifica-nomefile con questo semplice codice:

```
\loadmapfile[texnansi-xyz-times.map]
\font\MyFont=texnansi-utmr8a at 12 pt
\starttext
\MyFont Hello
\stoptext
```

Il nome `texnansi-utmr8a` non dice molto; meglio mappare i nomi simbolici utilizzando `\definefontsynonym`:

```
\definefontsynonym[NimbusRomNo9L-Medi][texnansi-utmb8a][encoding=texnansi]
\definefontsynonym[NimbusRomNo9L-MediItal][texnansi-utmbi8a][encoding=texnansi]
\definefontsynonym[NimbusRomNo9L-Regu][texnansi-utmr8a][encoding=texnansi]
\definefontsynonym[NimbusRomNo9L-ReguItal][texnansi-utmri8a][encoding=texnansi]
```

Certamente NimbusRomNo9L-Regu è più descrittivo di texnansi-utmr8a ma è possibile fare meglio: definiamo una seconda mappa, sempre per i files in `Times`, sfruttando la mappa precedente:

```
\definefontsynonym[Serif][NimbusRomNo9L-Regu]
\definefontsynonym[SerifItalic][NimbusRomNo9L-ReguItal]
\definefontsynonym[SerifBold][NimbusRomNo9L-Medi]
\definefontsynonym[SerifBoldItalic][NimbusRomNo9L-MediItal]
```

Decisamente meglio: abbiamo dei nomi significativi. Possiamo dire di aver completato il punto 2); ci manca l'ultimo passo.
Creiamo il file `my-type-collections.tex` (per ora solo per `times`) e raggruppiamo le mappe:

```
\starttypescript [map] [texnasni]
\loadmapfile[texnansi-xyz-times.map]
\stoptypescript
\starttypescript [serif] [times] [name]
\definefontsynonym[Serif][NimbusRomNo9L-Regu]
\definefontsynonym[SerifItalic][NimbusRomNo9L-ReguItal]
\definefontsynonym[SerifBold][NimbusRomNo9L-Medi]
\definefontsynonym[SerifBoldItalic][NimbusRomNo9L-MediItal]
\stoptypescript
\starttypescript [serif] [times] [texnansi]
\definefontsynonym[NimbusRomNo9L-Medi][texnansi-utmb8a][encoding=texnansi]
\definefontsynonym[NimbusRomNo9L-MediItal][texnansi-utmbi8a][encoding=texnansi]
\definefontsynonym[NimbusRomNo9L-Regu][texnansi-utmr8a][encoding=texnansi]
\definefontsynonym[NimbusRomNo9L-ReguItal][texnansi-utmri8a][encoding=texnansi]
\stoptypescript
```

Vediamo un piccolo test (tralasciamo per ora `\setupbodyfont`):

```
\usetypescriptfile[my-type-collections]
\definetypeface[MyFont][rm][serif][times][default][encoding=texnansi]
\setupbodyfont[MyFont,rm,10pt]
\starttext
Hello world!
\stoptext
```

In modo simile si opera per `Helvetica` e `Courier`: nel file `my-type-collections.tex`, all'interno della sezione [map] vi saranno le mappe dei nomi simbolici

```
\loadmapfile[texnansi-xyz-helvetica.map]
\loadmapfile[texnansi-xyz-courier.map]
```

seguiti dalle mappe per Helvetica

```
\definefontsynonym[NimbusSanL-Regu][texnansi-uhvr8a][encoding=texnansi]
\definefontsynonym[Sans][NimbusSanL-Regu]
%% altre mappe per helvetica...
```

e dalle mappe per Courier

```
\definefontsynonym[NimbusMonL-Regu][texnansi-ucrr8a][encoding=texnansi]
\definefontsynonym[Mono][NimbusMonL-Regu]
%% altre mappe per courier...
```

Il vantaggio del modello consiste in questo: con il meccanismo di mapping si possono utilizzare le definizioni per il cambiamento di stili e corpo già presenti nel formato, e quindi accessibili run-time. In `font-unk.tex`, tra le altre definizioni, si trova infatti:

```
\definebodyfont [default] [rm]
  [tf=Serif sa 1,
   bf=SerifBold sa 1,
   it=SerifItalic sa 1,
   sl=SerifSlanted sa 1,
   bi=SerifBoldItalic sa 1,
   bs=SerifBoldSlanted sa 1,
   sc=SerifCaps sa 1]
```

Così con `\setupbodyfont[MyFont,rm,10pt]` (cfr. sopra) si definisce il font times come font di documento, ed *automaticamente*, grazie al `\definebodyfont`, saranno disponibili le varianti di stile con le macro `\tf` (stile normale), `\bf` (neretto), `\it` (italico), `\bi`, (neretto inclinato), e così via.

L'aspetto finale del nostro codice sarà simile a questo (`DocFont` è il nome scelto per la collezione delle 3 famiglie):

```
\usetypescriptfile[my-type-collections]
\definetypeface[DocFont][rm][serif] [times]    [default] [encoding=texnansi]
\definetypeface[DocFont][ss][sans]  [helvetica][default] [encoding=texnansi]
\definetypeface[DocFont][tt][mono]  [courier]  [default] [encoding=texnansi]
\setupbodyfont[DocFont,rm,10pt]
\starttext
{\ss Hello world! \tt Hello world! \rm Hello world!}
\stoptext
```

Non solo: in questo modo sono disponibili anche le macro per variazioni del corpo come `\tfx`, `\tfa`, ed anche macro per le variazioni simultanee famiglia-stile, come `\ssbf` : in fig. 2 è possibile vedere un sommario per quanto riguarda la collezione cmr (il font standard).

[12.0pt]													\mr : Ag
	\tf	\sc	\sl	\it	\bf	\bs	\bi	\tfx	\tfxx	\tfa	\tfb	\tfc	\tfd
\rm	Ag	Ag	*Ag*	*Ag*	**Ag**	***Ag***	***Ag***	Ag	Ag	Ag	Ag	Ag	Ag
\ss	Ag	Ag	*Ag*	*Ag*	**Ag**	**Ag**	**Ag**	Ag	Ag	Ag	Ag	Ag	Ag
\tt	Ag	Ag	*Ag*	*Ag*	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag

Figure 2: La collezione cmr .

Non è qui possibile esaurire completamente la gestione dei fonts in ConTEXt: basti pensare ai fonts come `cmr` che hanno un proprio *design size*, o alle collezioni per la matematica, alla gestione di stili come calligrafico o *handwriting*, alla possibilità di definire una propria scala di variazione del corpo, di utilizzare mappe per sopperire alla mancanza di varianti. . . . Sono aspetti che richiedono competenza sia tipografica, sia del sistema TEX: ConTEXt semplifica, ma non può eliminare l'inerente complessità.

2.2 Sezionamento, sommario,indice

La suddivisione del contenuto in unità gerarchiche è affrontato in ConTEXt in modo generale: solo due strutture, la prima con i livelli numerati, la seconda no. Più precisamente:

- la suddivisione numerata:
  `\part`,
  `\chapter`
  `\section`,
  `\subsection`,
  `\subsubsection`,
  `\subsubsubsection`,
  `\subsubsubsubsection`;
- quella non numerata:
  `\part`,
  `\title`,
  `\subject`,
  `\subsubject`,
  `\subsubsubject`,
  `\subsubsubsubject`,
  `\subsubsubsubsubject`.

In stile ConTEXt, ogni livello di sezionamento è configurabile: `\setuphead` è il comando "incriminato" di cui sotto riportiamo per esteso la definizione (è un tipico esempio di come in ConTEXt viene illustrata una macro):

```
setuphead[...][..,..=..,..]
...             section
style           normal bold slanted boldslanted type cap small... command
textstyle       normal bold slanted boldslanted type cap small... command
numberstyle     normal bold slanted boldslanted type cap small... command
number          yes no
ownnumber       yes no
page            left right yes
continue        yes no
header          none empty high nomarking
text            none empty high nomarking
footer          none empty high nomarking
before          command
inbetween       command
after           command
alternative     normal inmargin middle text
command         \command#1#2
numbercommand   \command#1
textcommand     \command#1
prefix          + - text
placehead       yes no
incrementnumber yes no file
align           left right normal broad
tolerance       verystrict strict tolerant verytolerant stretch
indentnext      yes no
file            name
expansion       yes command no
```

◀▶setuphead si trova in `core-sec.tex`:
`\def\setuphead{\dodoubleargumentwithset\dosetuphead}`
dove `\dodoubleargumentwithset` è un'altra macro di sistema, definita in `syst-gen.tex`

Il significato delle chiavi–valori è abbastanza chiaro; tra le più interessanti
`command  \command#1#2`
permette di modificare con una macro il numero ed il testo della sezione, come ad esempio in

```
\setuphead[chapter]
          [command=\MyChapterCommand]
\def\MyChapterCommand#1#2{%
\centerline{#1--#2}
}
```

A prima vista strana, `placehead` permette di far sparire completamente il titolo della sezione (`placehead=no`); risulta utile ad esempio nei casi in cui non si voglia far apparire il titolo della sezione nell'area testo ma solo nelle testatine superiore/inferiore ed eventualmente nel sommario.
◀▶Per una scelta voluta, nel formato `\setuphead[part]` pone `placehead=no`.

I nomi `part, chapter, section,...` sono utilizzabili come riferimento in diversi contesti. Ad esempio ConTEXt permette di definire un proprio sistema di sezionamento equivalente a quello esistente, con la macro `\definehead`:

```
\definehead[Categoria][chapter]
\definehead[SottoCategoria][section]
```

`\Categoria` e `\SottoCategoria` sono ora macro equivalenti a `\chapter` e `\section`.
Ancora, se vogliamo che il contenuto della testatina superiore riporti il capitolo e sezione corrente, il comando
`\setupheadertexts[chapter][section]`
svolge proprio questa funzione (esiste anche `\setupfootertexts`)

Strettamente legato al sezionamento è la generazione del sommario: ogni sezione numerata è associata ad una *lista semplice*, generata mediante il comando `\placelist[`*sezione*`]` e personalizzabile con `\setuplist[`*sezione*`][...]`: ad esempio

```
\setuplist[chapter]
  [before=\blank,after=\blank,style=bold]
\starttext
\placelist[chapter]
\chapter{Foo}
%% altre sezioni
%%
\chapter{Baa}
%% altre sezioni
%%
\stoptext
```

genera la lista dei soli capitoli e la pone sulla prima pagina.

Le liste predefinite (corrispondenti cioè a `part`, `chapter`,..., `subsubsubsubsection`) sono a loro volta raggruppate in una lista `content` (ie. il nostro *sommario*) mediante `\definecombinedlist[content]`:

```
\definecombinedlist [content]
 [part,chapter,section,subsection,subsubsection,
  subsubsubsection,subsubsubsubsection]
 [level=subsubsubsubsection,
  criterium=local]
```

Il sommario è generato con `\placecontent` ed è, analogamente alle liste parziali, personalizzabile con `\setupcombinedlist[content][...]`, come ad esempio

```
\setupcombinedlist
  [content]
  [alternative=c,
   aligntitle=no,
   width=2.5em]
```

(`alternative=c` è una presentazione predefinita: ogni sezione compare con numero, titolo, e pagina separata da "filling dots").
◀▶`\placecontent` è sensibile al contesto: prima del primo capitolo genera il sommario, *dopo* un capitolo genera il *sommario parziale*.

La possibilità di agire sia sul setup della singola lista che di quella combinata, nonchè quella di generare la singola lista e la possibilità di ridefinire la composizione della lista `content` rende praticamente adattabile a qualsiasi esigenza il meccanismo predefinito di sezionamento. In alternativa ConTEXt permette di definire, parallelamente ad un proprio sezionamento, un corrispondente sistema di liste parziali e combinate, mediante `\definelist` e `\definicombinedlist`: un esempio di come adattare un sistema al problema, piuttosto che il contrario.

Concettualmente simile è la creazione di una *lista–indice* o registro, tipicamente usata per gli indici analitici.
In questo caso la possibiltà di definire diversi indici risulta vantaggiosa: pensiamo ad esempio al caso di un indice di codici numerici seguito da un indice in ordine alfabetico:
`\defineregister[CodeIndex][CodesIndex]`
`\defineregister[DescrIndex][DescrsIndex]`
da utilizzare nel contenuto ad esempio come `\CodeIndex{123}123` e `\DescrIndex{foo}foo`; gli indici effettivi si possono generare con `\placeCodeIndex` e `\placeDescrIndex`.

Esiste un indice predefinito, `index`, ovviamente configurabile e generato con `\placeindex` o `\completeindex` (cfr. fig. 3):

```
\setupcolors[state=start]
\setupregister[index]
          [textcommand=\Tframe,command=\Cframe,pagecommand=\Pframe]
\def\Tframe#1{\vrule width0pt depth1em \ss #1}
\def\Cframe#1{%
\framed[width=2em,align=middle,
```

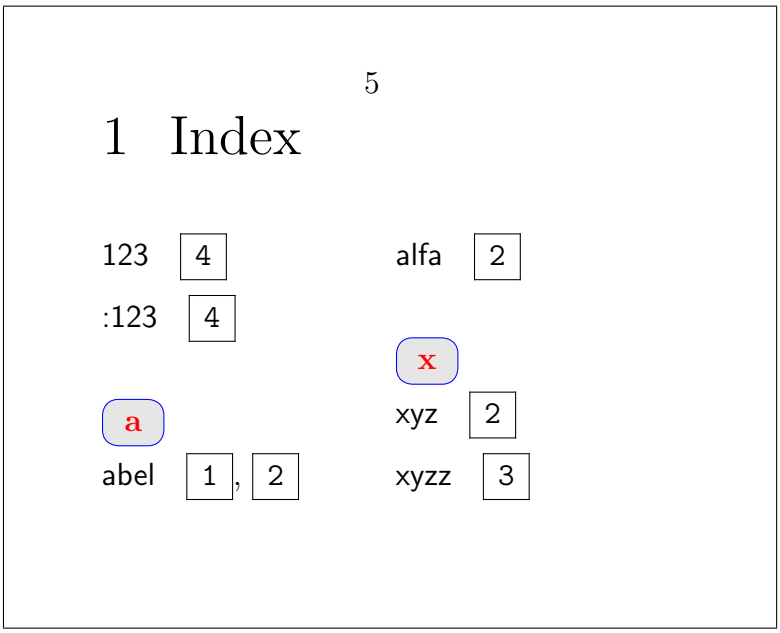


Figure 3: esempio di (brutta) formattazione dell'indice

```
      background=color,backgroundcolor=gray,
      foregroundcolor=red,framecolor=blue,corner=round]{#1}}
\def\Pframe#1{\tf\strut\inframed[width=1.5em,height=1.5em,]{\tt#1}}
\starttext
\index{abel}abel \page
\index{alfa}alfa \index{abel}abel
\index{xyz}xyz \page
\index{xyzz}xyzz \page
\index{123}123
\index{:123}:123 \page
\completeindex
\stoptext
```

◀▶Probabilmente gli indici necessitano di una ulteriore documentazione; ad es. le chiavi `textcommand` e `pagecommand` non sono illustrate nella macro `\setupregister`.

ConTEXt gestisce anche il sezionamento *funzionale* di un documento, ovvero la sua strutturazione nelle parti frontmatter · bodypart · appendices · backmatter Ogni parte è racchiusa dalla coppia `\start`*nome* `\stop`*nome* , ed al suo interno il comportamento del sezionamento varia concordemente: per esempio, in

```
\starttext
\startfrontmatter
\completecontent
\chapter{Presentazione}
\stopfrontmatter
\startbodymatter
\chapter{Foo}
```

```
\stopbodymatter
\startappendices
\chapter{Indice}
\stopappendices
\stoptext
```

il capitolo Presentazione non è numerato , Foo è numerato con cifre arabe, Indice è numerato con lettere alfabetiche `A,B...`; tutte e tre appaiono concordemente nel sommario.
Il sezionamento funzionale non è obbligatorio, come dimostrano gli esempi fino ad ora visti.

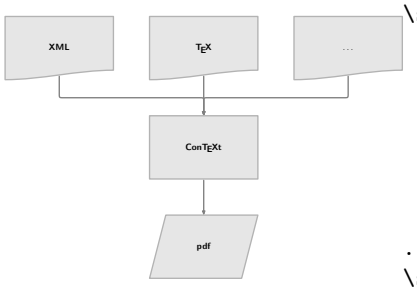
2.3 Contenuti

Le caratteristiche del TEX fanno parte di ConTEXt: l'*utente* ritrova cioè in ConTEXt gli stessi meccanismi del TEX per la composizione del testo, come il cambio del font, l'inserzione delle note, i riferimenti incrociati (che nella versione "interattiva" di un documento diventano "hyperlink"), testo in margine, ... (in caso contrario non sarebbe il caso di parlare di un sistema basato sul TEX).

La linea guida in ConTEXt per la gestione del contenuto è quella di includere nel formato pochi ambienti *altamente configurabili* per l'attività di typesetting ordinaria, ambienti che essenzialmente sono

1. elenchi;
2. tabelle;
3. riquadri (ie. "scatole" per raggruppare contenuto);
4. figure.

e usare moduli caricabili run-time (una sorta quindi di *plug-ins*) per esigenze speciali, come ad esempio `m-chart.tex` per i diagrammi flow-chart:




```
\usemodule[chart]
\starttext
\startTEXpage
\startFLOWchart[process]
  \startFLOWcell
    \name       {XML}
    \location   {1,1}
    \shape      {singledocument}
    \text       {{\ss\bf XML}}
    \connection [bt] {CONTEXT}
  \stopFLOWcell
...
\stopFLOWchart
\FLOWchart[process]
\stopTEXpage
\stoptext
```

◄►I moduli sono file di macro TEX identificati dal prefisso `m-`*modulo*`.tex`, caricati runtime con `\usemodule[`*modulo*`]`

Bisogna però considerare che nella gestione del contenuto intervengono diversi fattori: se è vero che pochi ambienti favoriscono un rapido inserimento del contenuto, ciò può condurre ad una strutturazione confusa del contenuto stesso. Ad esempio, se tutte le descrizioni, liste, elenchi fossero gestite con un unico ambiente elenco altamente configurabile, nel contenuto le distinzione logiche sarebbero meno chiare, mentre nel documento finale queste risalterebbero, e questo scollamento di struttura non giova all'autore/compositore.
Un altro fattore importante è di tipo tecnico (anzi "TEXnico"): alcuni problemi di typesetting sono difficili, sia da formulare che da risolvere. Un tipico esempio sono le tabelle: è evidente che un unico ambiente tabella è logico e desiderabile, ma praticamente è più efficiente suddividere il problema in "piccole tabelle" e "grandi tabelle" ed utilizzare due ambienti diversi.