

METAFONT

Generische Zeichenbeschreibung

Sebastian Wetzel

Inhaltsverzeichnis

1	Einleitung	2
1.1	Was ist METAFONT?	2
1.2	Bitmap-Fonts	2
1.3	Konzept von METAFONT	2
2	Aufbau von METAFONT	3
2.1	Basis & Modus	3
2.2	System von Gleichungen	4
2.3	Kartesisches Koordinatensystem	4
2.4	Linien	5
2.5	Pinsel	7
3	Beispiele	8
3.1	Beispiel 1: Das „D“	8
3.2	Beispiel 2: Das „K“	10
4	Dateizusammenhang	12
4.1	Die Dateien	12
4.2	manuelle Dateiumwandlung	14
5	Einbindung in L^AT_EX	14
6	Zeichensatz Computer Modern	16
7	Quellenangabe	16
8	☺	17

1 Einleitung

1.1 Was ist METAFONT?

METAFONT setzt sich aus 2 Begriffen zusammen: META und FONT.

- Meta ist ein Präfix aus dem Griechischen und bedeutet „nach“. Mit der Zeit hat sich im Lateinischen die Bedeutung „von höherer Ordnung“ durchgesetzt, ebenso in allen modernen Sprachen. Nur im Griechischen blieb die alte Bedeutung erhalten.
- Fonts sind Beschreibungen von Zeichen und Symbolen einer Zeichenfamilie. Größen, Weiten, Stile und weitere Eigenschaften einer Schrift werden dabei definiert.

Aus der Zusammensetzung ergibt sich, dass METAFONT eine Beschreibung von höherer Ordnung ist. Man kann sagen, das METAFONT ein Font ist, welches Fonts beschreibt.

Donald E. Knuth hat METAFONT parallel zu $\text{T}_{\text{E}}\text{X}$ entwickelt. So wie er $\text{T}_{\text{E}}\text{X}$ zum Erstellen von Dokumenten in einem besseren Schriftbild erschuf, so sollte METAFONT eine Programmiersprache auf Kommandozeilenbasis zum Definieren und Erzeugen von Schriften und Symbolen sein. Er brachte auch eine sehr detaillierte Dokumentation heraus: „The METAFONTbook“.

METAFONT kann auch zum Erstellen von Grafiken verwendet werden, aber dazu wurde später METAPOST als Weiterentwicklung geschaffen, welches dann PostScript-Dateien erstellt und nicht Font-Dateien.

1.2 Bitmap-Fonts

Um das Konzept von METAFONT zu verstehen, ist es günstig zu wissen, was ein Bitmap ist. Bei einer Bitmap handelt es sich um eine Beschreibungsart der Zeichen. Auf jedes Zeichen wird dabei ein Raster gelegt. Jedes Rasterelement ist dann gefüllt oder nicht. Entsprechend dieser Informationen wird dann das Zeichen gedruckt.

Probleme treten auf, wenn das Zeichen vergrößert oder auf eine andere Auflösungen portiert werden soll. Da die Rasterung im nachhinein unveränderbar ist, werden die Rasterelemente gestreckt oder gestaucht, was sich dann im Schriftbild negativ auswirkt. (z. B. Treppenbildung, Verschwinden von dünnen Linien)



1.3 Konzept von METAFONT

METAFONT ist ein sehr mathematisches Programm. Durch mathematische Gleichungen werden die Zeichen und Symbole beschrieben. Mit Parametern können die Ergeb-

nisse der Gleichungen beeinflusst werden, d. h. das Aussehen eines Zeichens kann ohne großen Eingriff in den Aufbau (Gleichungen) verändert werden. Sind alle gewünschten Zeichen beschrieben erstellt METAFONT daraus Bitmaps in der benötigten Auflösung und Größe. Dies wird dadurch realisiert, dass die Erstellung erst während der Laufzeit des T_EXen des Dokumentes geschieht. So werden für jede Konstellation von Auflösung und Größe die jeweiligen Bitmaps erstellt, wenn sie nicht schon vorhanden (z. B. explizites Compilieren des Schrift).

2 Aufbau von METAFONT

2.1 Basis & Modus

Die Basis

METAFONT arbeitet viel mit Macros. Diese „Ersetzungen“, welche die Programmierung erleichtern sollen, sind in einer Bibliothek zusammengefasst: der *Basis*. Die Basis kann aber auch schon Zeichendefinitionen enthalten. Die Standardbasis ist `plain` und bedarf keiner extra Einbindung. Um eine andere Basis manuelle einzubinden, muss folgendes eingegeben werden:

```
mf &basis_name ;
```

Der Modus

Ein weiterer wichtiger Bestandteil von METAFONT ist der *Modus*. Er gibt die verwendete Druckerkonfiguration an, beinhaltet also Daten für den Druckertreiber. Vor Allem die Information der zu verwendenden Auflösung ist dabei wichtig. Der Standardmodus ist `proof` und bedarf auch keiner extra Einbindung. Um einen anderen Modus zu verwenden, muss folgendes eingegeben werden:

```
mf \mode:=modus_name ;
```

Zwei wichtige Modi sind der genannte Standardmodus `proof` und `localfont`

- `proof`
Dieser Modus ist zum Entwickeln von Schriften gedacht, weshalb er auch als Standardmodus definiert wurde. Alle Zeichen werden in der Auflösung von 2601.72 dpi (36 pixel pro Punkt) und grau gespeichert. Ebenfalls werden die Markierungspunkte und die Box des Zeichens mit ausgegeben.
- `localfont`
Dieser Modus ist eine Konfiguration vom aktuellen Hauptdrucker. Die Zeichen werden in der entsprechenden Auflösung gespeichert und sind dann fertig zur Einbindung in L^AT_EX.

2.2 System von Gleichungen

METAFONT basiert weiterhin auf 4 Grundideen. Die Erste ist das Lösen von Gleichungen. Dabei setzen wir Variablen durch Gleichungen und Ungleichungen in Beziehungen und METAFONT versucht dann diese Gleichungen/Ungleichungen selbstständig zu lösen. Eine sehr mächtige Variable ist dabei `whatever`. Eigentlich ist sie keine Variable, sondern für METAFONT nur der Hinweis, dass an dieser Stelle eine Zahl eingesetzt werden muss. Wird `whatever` mehrmals verwendet, kann für jedes `whatever` eine andere Zahl eingesetzt werden. Wenn mit den Variablen gearbeitet wird, ist darauf zu achten, dass METAFONT zwischen „:=“ (Zuweisung) und „=“ (Relation) unterscheidet.

einfaches Beispiel:

	Eingabe und	intern in METAFONT
1.	<code>a + b - c = 0;</code>	<code>## c = a + b</code>
2.	<code>c = 2a;</code>	<code>## a = b</code>
3.	<code>a = 5;</code>	<code>## a = 5</code> <code>#### b = 5</code> <code>#### c = 10</code>
4.	<code>c = 0;</code>	<code>! Inconsistent equation</code> <code>(off by -10) ...</code>

Erläuterung:

1. METAFONT versucht, die eingegebene Gleichung günstig umzuformen
 2. Weitere Umformung
 3. `a` wird belegt und METAFONT löst die Belegung von `b` und `c`
 4. METAFONT gibt eine Fehlermeldung aus, da `c` schon belegt ist
- Zusatz: Bei einer weiteren Eingabe `c:=...` gibt es keinen Fehler

2.3 Kartesisches Koordinatensystem

Das Kartesische Koordinatensystem ist die zweite Idee. Es ist ein 2-Dimensionales System mit gleicher Einteilung beider Koordinatenachsen (in METAFONT Pixeleinteilung). Es bildet die Grundlage, auf der die Zeichen und Symbole definiert werden. Jeder beliebige Punkte ist innerhalb dieses Systems durch 2 Koordinaten eindeutig bestimmbar. Genauer betrachtet werden Punkte in METAFONT als Vektoren angesehen. Der Referenzpunkt ist dabei der Koordinatenursprung (= linker Punkt der Basislinie der Zeichenbox). Eine Konvention der `plain`-Basis ist die Darstellung der Vektoren durch Variablen:

$$\mathbf{z}a = (x_a, y_a) \quad a \in \mathbb{N}$$

Wenn eine `x`- oder `y`-Variable definiert wird, steht für METAFONT fest, dass eine Koordinate gemeint ist und der jeweilige Punkt `z` definiert wird. Auch die Einheitsvektoren stehen schon in der Basis als Macro zur Verfügung: `up`, `down`, `left`, `right`, `origin`.

Punkt-/Vektoroperationen

Die definierten Punkte bzw. Vektoren können nicht nur absolut gesetzt werden, sondern auch in Beziehung zueinander. Dazu dienen verschiedene Operationen:

<code>shifted</code>	ein Vektor wird um einen anderen Vektor verschoben – Addition/Subtraktion von Vektoren
<code>angle</code>	der Winkel eines Vektors bezüglich der Horizontalen wird berechnet
<code>dir</code>	der Einheitsvektor eines Winkels wird berechnet
<code>length</code>	die Länge eines Vektors wird berechnet
<code>scaled</code>	ein Vektor wird verlängert oder verkürzt – Multiplikation von Vektoren – $\text{scaled } d = d \cdot z$
<code>xscaled/yscaled</code>	Skalierung für die jeweilige Komponente eines Vektors
<code>rotated</code>	ein Vektors wird um den Referenzpunkt mit einem Winkel gedreht
<code>rotatedaround</code>	ein Vektors wird um einen beliebigen Punkt mit einem Winkel gedreht
<code>dotprod</code>	Skalarprodukt zweier Vektoren
$d[za, zb] = za + d(zb - za)$	die sogenannte Geradengleichung – ein beliebiger Punkt ist auf einer Geraden, die durch 2 Vektoren beschrieben wird, erreichbar

METAFONT versucht dann die Beziehungen aufzulösen.

2.4 Linien

Die dritte Grundidee ist die Art, wie METAFONT die Punkte verbindet. Dazu wird mit einem „Pinsel“ auf eine „schöne“ Weise eine Linie gezogen, die durch alle Punkte verläuft. „Schön“ bedeutet, dass METAFONT versucht, die Linie als Kurve darzustellen und die Krümmung soll sich zwischen zwei Punkten nicht ändern. Wenn es nicht anders geht, wird auch dies erlaubt. Der Standardpinsel ist ein halber Punkt (im `proof`-Modus 18 Pixel). Um das Zeichnen einzuleiten wird `draw` eingegeben, gefolgt von den zu verbindenden Punkten:

```
draw za..zb..zc.. ...;
```

Damit erstellt METAFONT automatisch eine „schöne“ Linie. Diesen Prozess kann aber beeinflusst werden. Zum Einen kann der Eintritts- und/oder Austrittsvektor im Punkt verändert werden. Dazu wird einfach ein Vektor vor bzw. hinter dem Punkt angegeben, z. B.

```
draw z1{up}..{left}z2..z3;
```

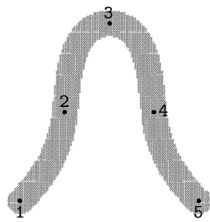
Zum Anderen kann die Linie vervollständigt werden. Dies geschieht durch das anhängen von

```
... ..cycle
```

an den Zeichenbefehl.

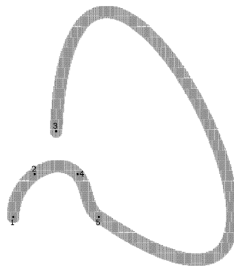
Beispiele: Linien

METAFONT output 2006.06.20:1041 Page 1



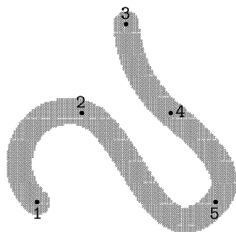
```
draw z1..z2..z3..z4..z5;
```

METAFONT output 2006.06.20:1041 Page 1



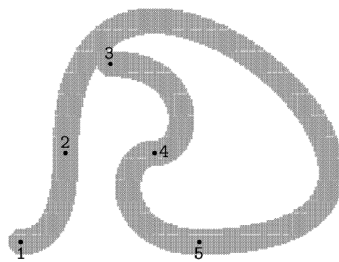
```
draw z1..z2..z4..z5..z3;
```

METAFONT output 2006.06.20:1042 Page 1

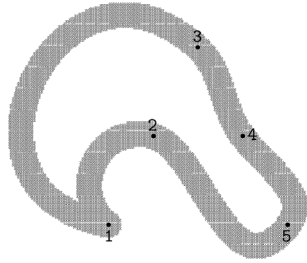


```
draw z1..z2..z5..z4..z3;
```

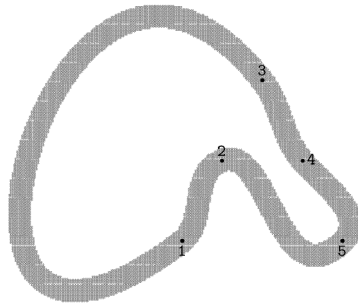
METAFONT output 2006.06.21:0021 Page 1



```
draw z1{right}..z2{up}..{left}z5..
{right}z4{right}..{left}z3;
```



`draw z1..z2..z5..z4..z3..z1;`



`draw z1..z2..z5..z4..z3..cycle;`

2.5 Pinsel

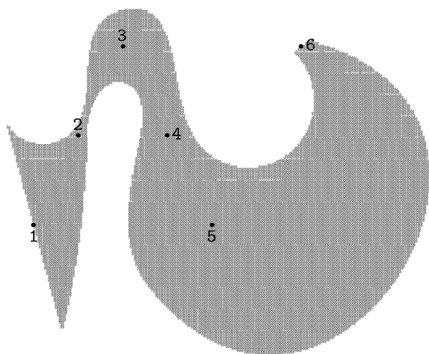
Die letzte Grundidee ist die Veränderbarkeit des „Pinsels“, denn er ist der gestalterische Aspekt der zu zeichnenden Linien. Es gibt drei Ausgangsformen (`penrazor`, `pencircle`, `pensquare`), die mit `pickup` geladen werden. Diese können dann beliebig verändert werden. Dazu stehen die gleichen Operationen wie für Vektoren zur Verfügung, aber nur `scaled` (bzw. `xscaled` und `yscaled`) und `rotated` sind wirklich sinnvoll.

Eine 2. Möglichkeit, der Linie eine Form zu geben, ist die implizite Nutzung von `penrazor`. Dabei wird zu jedem Punkt die Breite und der Winkel des Pinsels definiert:

`penposa (breite, winkel);`

Um dann damit zu zeichnen muss statt `draw penstroke` benutzt werden. Zusätzlich muss an jedem Punkt ein „e“ angehängt werden.

Beispiel: Pinsel (2. Möglichkeit)



```
z1=(0,0); z5=(100,0);
z3=(50,100);
z2=.5[z1,z3];
z4=.5[z3,z5];
z6=(150,100);
penpos1(125,-75);
penpos2(10,0);
penpos3(40,-90);
penpos4(30,-180);
penpos5(100,-135);
penpos6(10,40);
penstroke z1e..z2e..z3e..z4e..z5e..z6e;
labels(1,2,3,4,5,6);
```

```
showit;
shipit;
end
```

3 Beispiele

3.1 Beispiel 1: Das „D“

METAFONT output 2006.06.20:1003 Page 1

2.

3.

1.

```
Phi=(1+sqrt5)/2;
w=100;
h=Phi*w;
z1=z2-h*up=origin;
x3=w;
y2-y3=Phi*(y3-y1);
drawdot z1;
drawdot z2;
drawdot z3;
labels(1,2,3);
showit;
shipit;
end
```

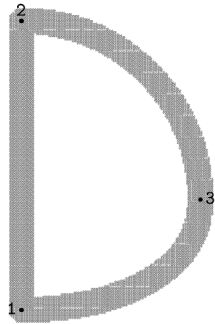
METAFONT output 2006.06.20:1001 Page 1

2.

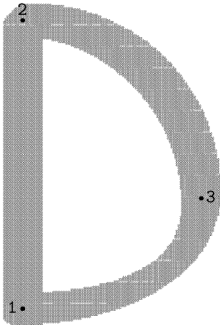
3.

1.

```
Phi=(1+sqrt5)/2;
w=100;
h=Phi*w;
z1=z2-h*up=origin;
x3=w;
y2-y3=Phi*(y3-y1);
draw z1..z2;
draw z1..z3..z2;
labels(1,2,3);
showit;
shipit;
end
```



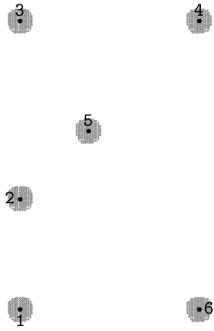
```
Phi=(1+sqrt5)/2;
w=100;
h=Phi*w;
z1=z2-h*up=origin;
x3=w;
y2-y3=Phi*(y3-y1);
draw z1..z2;
draw z1{right}..z3{up}..{left}z2;
labels(1,2,3);
showit;
shipit;
end
```



```
Phi=(1+sqrt5)/2;
w=100;
h=Phi*w;
z1=z2-h*up=origin;
x3=w;
y2-y3=Phi*(y3-y1);
pickup pencircle scaled 15 xscaled Phi rotated
angle (Phi,1);
draw z1..z2;
draw z1{right}..z3{up}..{left}z2;
labels(1,2,3);
showit;
shipit;
end
```

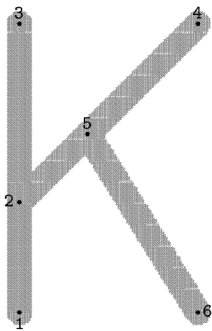
3.2 Beispiel 2: Das „K“

METAFONT output 2006.06.20:1025 Page 1

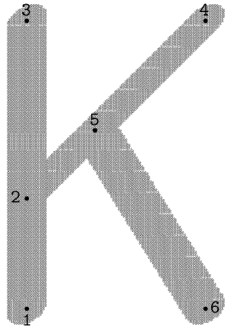


```
Phi=(1+sqrt5)/2;
w=100;
h=Phi*w;
z1=z3-h*up=z6-w*right=origin;
z4=z1 rotatedaround (.5[z3,z6],180);
z2=whatever*up;
y3-y2=Phi*(y2-y1);
z5=whatever[z2,z4];
x6-x5=Phi*(x5-x1);
drawdot z1;
drawdot z2;
drawdot z3;
drawdot z4;
drawdot z5;
drawdot z6;
labels(range 1 thru 6);
showit;
shipit;
end
```

METAFONT output 2006.06.20:1027 Page 1



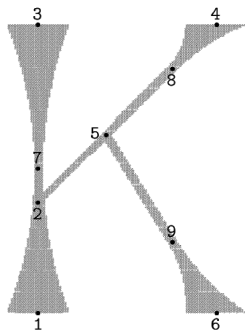
```
Phi=(1+sqrt5)/2;
w=100;
h=Phi*w;
z1=z3-h*up=z6-w*right=origin;
z4=z1 rotatedaround (.5[z3,z6],180);
z2=whatever*up;
y3-y2=Phi*(y2-y1);
z5=whatever[z2,z4];
x6-x5=Phi*(x5-x1);
draw z1..z3;
draw z2..z4;
draw z5..z6;
labels(range 1 thru 6);
showit;
shipit;
end
```



```

Phi=(1+sqrt5)/2;
w=100;
h=Phi*w;
z1=z3-h*up=z6-w*right=origin;
z4=z1 rotatedaround (.5[z3,z6],180);
z2=whatever*up;
y3-y2=Phi*(y2-y1);
z5=whatever[z2,z4];
x6-x5=Phi*(x5-x1);
pickup pencircle scaled 15 xscaled Phi rotated
angle (Phi,1);
draw z1..z3;
draw z2..z4;
draw z5..z6;
labels(range 1 thru 6);
showit; shipit;
end

```

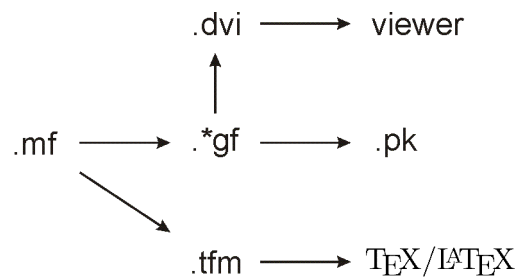


```

Phi=(1+sqrt5)/2;
w=100;
h=Phi*w;
z1=z3-h*up=z6-w*right=origin;
z4=z1 rotatedaround (.5[z3,z6],180);
z2=whatever*up;
y3-y2=Phi*(y2-y1);
z5=whatever[z2,z4];
x6-x5=Phi*(x5-x1);
z9=.6[z5,z6];
z7=.5[z1,z3];
z8=whatever[z2,z4]=(x9,whatever);
penpos1(35,0);
penpos2(5,-90);
penpos3(35,0);
penpos4(35,0);
penpos5(5,angle(z4-z5));
penpos6(35,0);
penpos7(5,0);
penpos8(5,angle(z4-z2)-90);
penpos9(5,angle(z6-z5)+90);
penstroke z1e..upz7e..z3e;
penstroke z2e--z8e{z4-z2}..z4e;
penstroke z5e--z9e{z6-z5}..z6e;
labels(range 1 thru 6);
showit; shipit;
end

```

4 Dateizusammenhang



4.1 Die Dateien

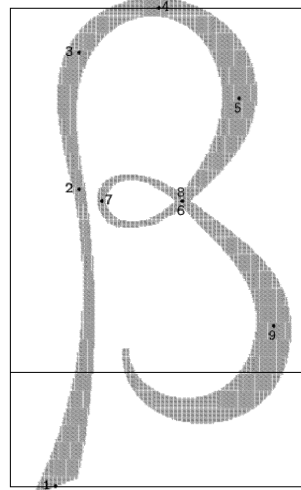
.mf

Diese Dateien sind die Quelldateien für METAFONT und können mit jedem beliebigen Editor erstellt werden. Eine komplette Schriftfamilie hat unter normalen Umständen folgenden Aufbau:

- Definition von Parametern
- Umrechnung pixelunabhängige ($a\#$) in pixelabhängige (a) Parameter: `define_pixel`
- Zeichen *char* mit Box der Breite *b*, Höhe *h* und Tiefe *t* einleiten:
`beginchar(char, b, h, t);`
- Beschreibung von Punkten/Vektoren und deren Verbindungen
- Zeichen beendet: `endchar;`
- weitere Zeichenbeschreibungen
- besondere Parameter, wie Ligaturen oder Unterschneidungen
- Ende der Datei: `end`

Beispiel einer .mf-Datei

```
u#:=4/9pt#;
define_pixels(u);
beginchar(66,13u#,16u#,5u#); "Letter beta";
x1=2u; x2=x3=3u;
bot y1=-5u; y2=8u; y3=14u;
x4=6.5u; top y4=h;
z5=(10u,12u);
z6=(7.5u,7.5u); z8=z6;
z7=(4u,7.5u);
z9=(11.5u,2u);
z0=(5u,u);
penpos1(2u,20);
penpos2(.5u,0);
penpos3(u,-45);
penpos4(.8u,-90);
penpos5(1.5u,-180);
penpos6(.4u,150);
penpos7(.4u,0);
penpos8(.4u,210);
penpos9(1.5u,-180);
penpos0(.3u,20);
pickup pencircle;
penstroke z1e..z2e..z3e..z4e..z5e..z6e..{up}z7e..z8e..z9e..{up}z0e;
labels(range 1 thru 9);
endchar;
end
```



.*gf

GF steht für „generic font“ und diese Dateien enthalten die Bitmap der Zeichen. * steht für eine Zahl, die die Auflösung angibt, in der METAFONT die Schrift erstellt hat. Diese Zahl hängt vom verwendeten Modus ab. In der frühen Zeit von METAFONT waren diese Dateien für den Drucker bestimmt. Heute werden nur noch die .pk-Dateien vom Druckertreiber verwendet.

.pk

PK steht für „packed“ und ist einfach nur die gepackte Version von .*gf-Dateien. Sie sind die relevanten Dateien für den Druckertreiber.

.tfm

TFM steht für „T_EX font metric“. Diese Dateien enthalten Größen der Zeichenboxen, Ligaturen, Unterschneidungen, etc. Deshalb sind dies die Dateien, die von T_EX/L^AT_EX verwendet werden. Die anderen genannten Dateien sind für T_EX/L^AT_EX uninteressant.

4.2 manuelle Dateiumwandlung

Aktuelle Distributionen von L^AT_EX benötigen nur die .mf-Dateien und erstellen alles weitere automatisch. Zum Entwickeln oder expliziten Umwandeln der Dateien können die Befehle in die Kommandozeile eingegeben werden. Der normale Aufruf von META-FONT mit seinen Standardeinstellungen (Basis und Modus) erfolgt mit:

```
mf datei_name.mf;
```

Dabei wird nur eine .2602gf-Datei erstellt. Mit dem Tool `gftidvi` wird sie in eine .dvi-Datei umgewandelt, welche man sich dann anschauen kann. Werden aber eigens definierte Einstellungen benutzt, kann man diese durch:

```
mf [&basis_name ;] [\mode:=modus_name ;] input datei_name.mf;
```

mit einbinden. Abhängig vom Modus werden dann *.gf- & .tfm-Dateien erstellt. Zusätzlich muss noch das Tool `gftopk` ausgeführt werden und man hat alle benötigten Dateien.

5 Einbindung in L^AT_EX

Es gibt 2 Möglichkeiten, um eine erstellte Schriftart einzubinden: eine recht einfache und eine komplizierte Variante.

Die einfache Variante sieht vor, das zuerst die neue Schriftart einen Namen bekommt:

```
\font\name=datei_name [scaled x] (alte Konvention) oder
```

```
\newfont{name}{datei_name [scaled x]} (neue Konvention).
```

Bei der Skalierung der Schrift ist darauf zu achten, dass *x* noch durch 1000 dividiert wird. Möchte man die Schrift nun anwenden, geht man folgendermaßen vor:

```
{\name Text}
```

In dieser Variante werden alle Zeichen genau in der Art benutzt, wie sie in der .mf-Datei definiert wurden.

In der komplizierteren Variante muss alles genaustens definiert werden, für welche Schriftserie, für welche Schriftform und für welche Schriftgröße wir welche neue Schriftart verwenden wollen. Zusätzlich gibt man auch noch die Schriftkodierung an:

```
\DeclareFontFamily{schriftkodierung}{datei_name}{code}
```

```
\DeclareFontShape{schriftkodierung}{datei_name}{schriftserie}
{schriftform}{schriftgröße datei_name}{code}
```

Mit `code` kann noch \LaTeX code definiert werden, der zusätzlich ausgeführt werden soll. Auch die Anwendung der definierten Schriftart sieht sehr kompliziert aus:

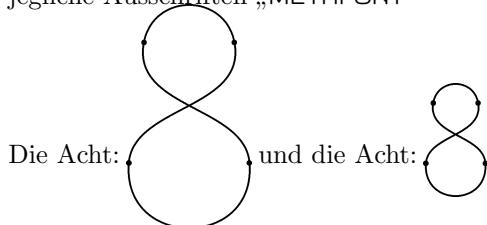
```
{\usefont{schriftkodierung}{datei_name}{schriftserie}{schriftform}
Text}
```

Hier versucht \LaTeX sogar, die neue Schrift automatisch auf die gerade verwendete Schriftgröße zu skalieren (siehe folgendes Beispiel). Die genauen Auswirkungen jeder einzelnen Einstellung bitte in der Dokumentation „The METAFONTbook“ nachlesen.

Beispiel

Variante 1:

jegliche Ausschriften „METAFONT“



Variante 2:

Die Acht: **8**

\LaTeX code des Beispiels

```
\newfont{\mf}{logo10}
\newcommand{\MF}{\mf META}\-\mf FONT\}
\DeclareFontFamily{OT1}{eight}{}
\DeclareFontShape{OT1}{eight}{m}{n}{<-> eight}{}
\newfont{\acht}{eight}
\font\achte=eight scaled 500
Variante 1: \
\indent jegliche Ausschriften "\MF " \
\indent Die Acht: {\acht 8} und die Acht: {\achte 8} \[\baselineskip]
Variante 2: \
\indent Die Acht: {\usefont{OT1}{eight}{m}{n}8}
```

Quelldatei eight.mf

```
mode_setup;
u# := 2mm#;
define_pixels(u);

beginchar("8", 8u#, 10u#, 5u#);
  z1 = ( 0u, 0u);
  z2 = ( 8u, 0u);
  z3 = ( 1u, 8u);
  z4 = ( 7u, 8u);
  pickup pencircle scaled 1u#;
  draw z4 .. z1 .. z2 .. z3 .. cycle;
  pickup pencircle scaled 3u#;
  drawdot z1;
  drawdot z2;
  drawdot z3;
  drawdot z4;
endchar;

end
```

6 Zeichensatz Computer Modern

Der Zeichensatz `COMPUTER MODERN` beinhaltet alle Standardschriften von \LaTeX . Er wurde von Donald E. Knuth natürlich mit `METAFONT` erstellt. Alle Zeichendefinitionen sind bereits in der Basis (`cm.base`) enthalten. Die Quelldateien enthalten „nur“ noch 62 Parameter. Damit lassen sich alle Schriften (ob dick oder dünn, ob mit oder ohne Serifen, etc.) erstellen. In den aktuellen Distributionen von \LaTeX liegen schon verschiedene vorcompilierte Schriften vor (`cmxx.mf & .tfm`). `xx` steht dabei für die Schriftart, z. B. `cmr10.mf` für die Quelldatei der *Roman*-Achrift in der Größe 10pt.

7 Quellenangabe

1. Müllejans, Gordon: `METAFONT: Eine Referenz`, Addison-Wesley, 1992
2. <http://metafont.tutorial.free.fr/>
3. <http://www.dante.de/faq/de-tex-faq/>
4. <ftp://ftp.dante.de/pub/tex/documentation/>

8 ☺

