LaTeX Vorlesung

Grundlagen der LaTeX-Programmierung

Dr. Holger Brünner, Dipl.-Phys.

VisioCraft GmbH, Erlangen

Was sind Makros (Benutzerebene)?

- Makro: Kurzbefehl, der einen längeren Text ausgibt
- Wiederkehrende Texte können vereinfacht werden: statt »Diacetyl-Meta-Phenylendiamin« schreibt man im Text nur noch `\DMP`
- Verhindert Tippfehler und macht das Schreiben angenehmer

Was sind Makros (Systemebene)?

- Makro: Befehl, der bestimmte Aktionen ausführt
- z. B. die Schriftart wechselt (`\textbf{fett}` macht **fett**)
- Stellt Funktionen zur Verfügung, die so noch nicht vorhanden waren
- Verwendet üblicherweise Argumente und verarbeitet diese
- Kann durch Parameter gesteuert werden

LaTeX als Programmiersprache

- LaTeX selbst ist als Makropaket implementiert
- Alle Befehle sind Makros, können durch den Benutzer umdefiniert/erweitert/verändert werden
- Theoretisch können beliebig komplexe Programme in LaTeX geschrieben werden
- . . . das ist aber mühsam!
- Hauptanwendung: Schreiben eigener Formatierungs- und Ausgabemakros
  - Textersetzung
  - logisches Markup (\myemph{})
  - neue Umgebungen (myitemize, mytitlepage)

Ziele der heutigen Vorlesung

- Schreiben einfacher eigener Makros: Vereinfachung und logisches Markup
- Mehrfachausführung von Befehlen und Programmkontrolle
- Packages, Klassen, Besonerheiten
- Plus: Titelseite zum Selberbasteln

Nicht aufgeben, wenn Ihnen das alles klingonisch vorkommt! Nicht unbedingt notwendig, um LaTeX anzuwenden! (Aber es macht das Leben leichter... und Spaß!)

Datentypen

LaTeX kennt die folgenden Datentypen (vereinfacht):

- Befehle: `\cmd[<opt. Argument>]{<Pflichtargument>}`
  - führt Kommandos aus:
    `\cmd ---> <Liste der Anweisungen>`
  - als Zeichenvariable: `\cmd ---> <var>:string`
- Zählervariablen (counter): `<cnt>:int`
- Längenvariablen (length): `<len>:"float"`
- Wahrheitswerte (boolean): `<yn>:bool`

Befehle

- Definition eigener Befehle:
  `\newcommand{\mycmd}[Anzahl Argumente]{Definition}`
- Anstelle des Namens `\mycmd` wird die Definition in das Dokument eingesetzt
- Argumente können übergeben werden als `\mycmd{arg1}{arg2}{arg3}...` (max. 9)
- Keine Argumente: `\newcommand{\mycmd}{Definition}`
- Zugriff auf Argumente innerhalb der Definition mittels `#1, #2, #3...`

\newcommand: Beispiel

- Sie schreiben eine DA über die Wirkung der Substanz: $\mathrm{C_2\,H_5\,OH}$.
- Statt jedesmal im Text zu tippen
  `$\mathrm{C_{2}\,H_{5}\,OH}$`
  verwenden Sie folgende Definition:
  `\newcommand{\alk}{$\mathrm{C_{2}\,H_{5}\,OH}$}`
- Damit schreiben Sie im Text:
  `Die Wirkung von \alk{} auf das Denkvermögen...`
- und erhalten:
  »Die Wirkung von $\mathrm{C_2\,H_5\,OH}$ auf das Denkvermögen...«

\newcommand: Beispiel (2) I

Probleme:

- Eigene Kommandos verschlucken folgenden Leerraum. Ohne die leeren Klammern {} sähe das so aus: »Die Wirkung von $\mathrm{C_2\,H_5\,OH}$auf das Denkvermögen. . . «
- Die Klammern sind lästig. Einfach einen Leerraum in die Definition einzugeben führt aber wieder zu Problemen am Satzende: »Denken und $\mathrm{C_2\,H_5\,OH}$.«
- Zusätzlich bei Verwendung des neuen Befehls im Mathematikmodus: Fehlermeldung (das erste $ schaltet den Mathemodus aus, worauf die folgenden _-Befehle undefiniert sind)!

\newcommand: Beispiel (3)

Lösung:

- `\usepackage{xspace}` in die Präambel:
  Leerraum abhängig vom nachfolgenden Zeichen
- Mathemodus »dynamisch« einschalten: `\ensuremath`
- Erweiterte Definition des Kommandos:

```
\newcommand{\alk}{%
  \ensuremath{\mathrm{C_{2}\,H_{5}\,OH}}\xspace},
```

und aus `Viel \alk. \alk und Wasser. $\alk$` wird:

Viel $\mathrm{C_{2}\,H_{5}\,OH}$. $\mathrm{C_{2}\,H_{5}\,OH}$ und Wasser. $\mathrm{C_{2}\,H_{5}\,OH}$

\newcommand: Beispiel (4) I

- Andere Alkohole: Übergabe der Anzahlen als Parameter an das neue Kommando
- Angabe der Anzahl der Argumente nötig

```
\newcommand{\malk}[2]{%
  \ensuremath{\mathrm{C_{#1}\,H_{#2}\,OH}}\xspace}
```

- z. B. Propanol: `\malk{3}{7}` $\Rightarrow \mathrm{C_3\,H_7\,OH}$

\newcommand: Beispiel (4) II

- Umdefinition bestehender Makros ist mit `\renewcommand` möglich: `\renewcommand{\<name>}{<neue Definition>}`
- Das nutzen wir hier zu einer Behandlung von Ethanol als Sonderfall des Alkohols:

  ```
  \renewcommand{\alk}{\malk{2}{5}}
  ```

- Achtung: Anzahl der Parameter bei der Neudefinition muß mit der ursprünglichen Anzahl übereinstimmen!

\newcommand: Zusammenfassung

- Erste Definition eines neuen Makros:
  `\newcommand{\mymacro}{...`
- Änderung des Makros: `\renewcommand{\mymacro}{...`
- Erlaubte Zeichen für den Namen:
  - Groß- und Kleinbuchstaben (werden unterschieden!)
  - Keine Ziffern!
  - Keine Sonderzeichen!
  - Auf Umlaute besser verzichten - kann gehen, muß aber nicht
- »Unbegrenzte« Anzahl an eigenen Makros definierbar
- Optionale Argumente, `\newcommand*{...` - Doku lesen!

Zähler

- In Zählern werden integer-Werte gespeichert
- Bereits bekannter Zähler: Seitenzahl `page`
- Dienen zur Automatisierung von Zählungen
- können automatisch / in Abhängigkeit von anderen Zählern manipuliert werden.
  Z. B. wird der *subsection*-Zähler automatisch zurückgesetzt, wenn *section* inkrementiert wird.
- Mit Zählern können arithmetische Operationen durchgeführt werden

Eigene Zähler

- Deklaration eigener Zähler:
  `\newcounter{mycnt}`
- *mycnt* wird mit 0 initialisiert
- Zugriff auf die Werte von mycnt:
  - `\arabic{mycnt}` gibt den Wert von *mycnt* als Textstring in arabischen Ziffern für die Ausgabe zurück
  - `\roman{mycnt}` gibt den Wert von *mycnt* als Textstring in kleinen römischen Ziffern zurück
  - `\value{mycnt}` gibt den Wert von *mycnt* als Zahl zurück (Nur für Berechnungen! Nicht als Text darstellbar!)
  - `\themycnt` Kurzform, Standard $\equiv$ `\arabic{mycnt}`

Eigene Zähler: Beispiele

Alle Überschriftenebenen haben einen verbundenen Zähler des gleichen Namens:

- Nummer der aktuellen Subsection für die Kopfzeile: `\thesubsection`
- Nummer der aktuellen Section in großen römischen Ziffern: `\Roman{section}`

Die Zähler der Ebenen sind untereinander verbunden: bei einer Erhöhung des counters *section* (=neue `\section`) wird der Zähler *subsection* zurückgesetzt.

Operationen mit Zählern:

- Wert `val` dem Zähler *cnta* zuweisen:
  `\setcounter{cnta}{val}`
- Zum Wert in *cnta* den Wert `val` addieren:
  `\addtocounter{cnta}{val}`
- Zum Wert in *cnta* den Wert im zweiten Zähler *cntb* addieren: `\addtocounter{cnta}{\value{cntb}}`
- Zähler *cnta* um 1 erhöhen (inkrementieren):
  `\stepcounter{cnta}`

Längen

- Dimensionen werden in Längenregistern gespeichert
- Benötigen Maßeinheit (mm, cm, pt...)
- Können »variabel« sein: Länge zwischen 1 und 2 cm
- Deklaration: `\newlength{\mylen}`
- Operationen:

  `\setlength{\mylen}{<NeueLänge>}`,

  `\addtolength{\mylen}{<ZusatzLänge>}`
- Variable Maße:

  `\setlength{\mylen}{<NeueLänge>plus <P> minus <M>}`

Längen: Beispiele

- `\newlength{\mylen} \setlength{\mylen}{10cm} \begin{minipage}{\mylen}...` produziert Minipages gleicher Breite
- `\addtolength{\oddsidemargin}{1cm}` verschiebt den Satzspiegel um 1 cm (Bitte nicht verwenden!)
- `\settowidth{\mylen}{Text}` mißt die Breite eines Textblocks und speichert diese in `\mylen`

```
\newlength{\mylen}\settowidth{\mylen}{Dies ist Text}
Dies ist Text\hspace{-\mylen}%
  \raisebox{0.5ex}{\color{red}\rule{\mylen}{0.4pt}}
```

~~Dies ist Text~~

Vergleichsoperationen

- Häufiges Problem: Verzweigung abhängig vom Wert einer Variablen
- klassisch: if. . . then - Konstrukte
- Zur Vereinfachung der Notation: `\usepackage{ifthen}`
- Verzweigung:

  `\ifthenelse{test}{true-Code}{false-Code}`
- Test: value1 = value 2 (oder auch <, >)
- Vordefinierte Abfragen:

  `\iflanguage{},\ifnum{},\lengthtest...`

Vergleichsoperationen 2

Exemplarische Beispiele:

- Test auf Gleichheit zweier Zeichenketten:

```
\ifthenelse{\equal{<strA>}{<strB>}}%
{Code für ">wahr"<}{Code für ">falsch"<}
```

- Test auf A > B (beide in countern valA und valB gespeichert):

```
\ifthenelse{\value{valA} >
   \value{valB}}{Code für A>B}{Code für
   A!>B}
```

- Welche Sprache ist gerade aktiv?

```
\iflanguage{ngerman}{Deutsch}{english}
```

- Beispiel im Übungs-Zipfile!

Schleifen

- Wiederholung von gleichen Aufgaben: `For...next` - Schleifen
- LaTeX: `whiledo`-Befehl:

  `\whiledo{Bedingung}{<code>+<Inkrement>}`

Notenliste zum Ankreuzen:

```
\newcounter{ctra}\setcounter{ctra}{1}
Note: \whiledo{\value{ctra} < 7}%
      {\thectra {\large\ding{111}}
      \stepcounter{ctra}}
```

Note: 1❑ 2❑ 3❑ 4❑ 5❑ 6❑

Übungsaufgaben zum Termin 9 - Teil 1(1)

- Schreiben Sie einen neuen Befehl `\GSO`, der Ihnen die Tipparbeit beim (vollständigen) Namen Ihrer Hochschule abnimmt, d. h. als Abkürzung für »Georg-Simon-Ohm-Hochschule für angewandte Wissenschaften Nürnberg« dient. Wie ist das mit dem Leerraum nach dem Befehl?
- Schreiben Sie einen neuen Befehl `\spiel` mit 4 Argumenten, der Fußballergebnisse in der Form <Mannschaft1> gegen <Mannschaft2>: <n>:<m> ausgibt, also z. B.

  ```
  \spiel{1. FC Nürnberg}{HSV}{0}{2}
  ```

  ⇒ 1. FC Nürnberg gegen HSV: 0:2

Übungsaufgaben zum Termin 9 - Teil 1(2)

- Formatieren Sie den Gewinner der Partie im letzten Beispiel **fett**. Das geht mit einer konditionalen Abfrage auf den Argumenten. Bei Unentschieden soll keine Mannschaft fett gedruckt werden. Der Befehlsname soll gleich bleiben.
  `\spiel{1. FC Nürnberg}{HSV}{0}{2}`
  $\Rightarrow$ 1. FC Nürnberg gegen **HSV** 0 : 2
- Schreiben Sie einen Befehl, der zwei Ganzzahlen addiert und das Ergebnis im Mathematikmodus ausgibt, also aus
  `\addiere{5}{3}` $\Rightarrow$ $5 + 3 = 8$ macht

Übungsaufgaben zum Termin 9 - Teil 1(3)

- Für TEXperten (und solche mit Programmiererfahrung): Schreiben Sie einen Befehl `\countdown`, der ein ganzzahliges Argument übernimmt und von diesem Wert an rückwärts bis Null zählt:
  `\countdown{12}` $\Rightarrow$ 12 – 11 – 10 – 9 – 8 – 7 – 6 – 5 – 4 – 3 – 2 – 1 – 0

  (Hinweis: Überflüssige Leerzeichen entstehen durch die Zeilenenden. Diese können durch % »auskommentiert« werden. Verwendung siehe Musterlösung im zip-File.

Eigene Umgebungen

- Für eigene Listen, Tabellen, Aufzählungen...
  $\Rightarrow$ eigene Umgebungen definieren.
- Analog zu `\newcommand`:

  `\newenvironment{myenv}{<startenv>}{<endenv>}`

`myenv`	Name der neuen Umgebung
`<startenv>`	Code, der zu Beginn der Umgebung ausgeführt wird
`<endenv>`	Code, der beim Beenden der Umgebung (also beim `\end{myenv}`) ausgeführt wird

  stellt `\begin{myenv} ... \end{myenv}` zur Verfügung.

Besonderheiten bei eigenen Umgebungen

- Im <startenv>-Code können lokale Kommandos per `\newcommand{}` definiert werden
- (das entspricht einer Überladung der Kommandos)
- Achtung: Referenzierung der Argumente dort per Doppelung des Hashes, d.h. `##1` meint das erste Argument eines innerhalb des `myenv` definierten `newcommands` `\mylocalcmd`
- Einbindung anderer Umgebungen (`\begin{tabular}...`) in TeX-Syntax `\tabular ... \endtabular`

Klassen und Pakete

`.cls` und `.sty`

- Dokumentenklassen: legen Grundeinstellungen des Dokuments fest
- Pakete: stellen Funktionen zur Verfügung
- $\Rightarrow$ Beides sind »normale« LaTeX-Files (mit einigen Besonderheiten), werden per `\usepackage{<package>}` bzw. `\documentclass{<classfile>}` geladen
- Was sind die Besonderheiten?

Klassen

- Dokumentenklassen: legen Grundeinstellungen des Dokuments fest
- Bisher bekannt: article, scrartcl, book ...
- Klassen können (genau) eine andere Klasse laden & auf dieser aufbauen
- Klassen können Pakete laden
- Eigene Klasse: Vereinheitlichung und Vereinfachung – Integration des Headerfiles

.cls-Files: Klassen

Präambel eines normalen Dokuments:

```
\documentclass[a4paper,ngerman,twoside]{article}
\usepackage[ansinew]{inputenc}
\usepackage[T1]{fontenc}
\usepackage[ngerman]{babel}
\usepackage{xcolor,graphicx}
```

$\Rightarrow$ in eigene Klasse packen!

.cls-Files: myarticle.cls

Eine Demoklasse:

```
\ProvidesClass{myclass}[2007/05/24 v0.1
   Demoklasse]
\LoadClassWithOptions[a4paper,ngerman,twoside]
                     {article}
\RequirePackage[ansinew]{inputenc}
\RequirePackage[T1]{fontenc}
\RequirePackage[ngerman]{babel}
\RequirePackage{xcolor,graphicx}
```

$\Rightarrow$ als `myarticle.cls` speichern, und Dokumente dieser Klasse können mit `\documentclass{myarticle}` deklariert werden.

.cls-Files: Besondere Befehle

- `\ProvidesClass` Identifiziert die Klasse (wird im Logfile angezeigt)
- `\LoadClassWithOptions`: Läd eine andere Klasse, übergibt Optionen
- `\RequirePackage`: Läd ein Paket. Wird später versucht, das Paket nochmals zu laden (z. B. mit abweichenden Optionen), wird es nicht nochmals geladen.

$\Rightarrow$ Dokumentenvorlage

Das Geheimnis des \usepackage{xyz}...

- Wir wissen: Fehlen Funktionen, kann man diese per `\usepackage{<package>}` einbinden
- Packages:= Sammlung von `\newcommand`-Befehlen, die ein anderer geschrieben hat
- Speicherung in .sty-Files, aber »gleiches« Format wie .tex
- Jeder LaTeX-User kann & darf eigene packages schreiben und diese der community zur Verfügung stellen
- $\exists$ unzählige Pakete $\Rightarrow$ `www.ctan.org`, Suchen in Dokumentationen!
- ...oder selber machen ;-)

.sty-Files: Packages

- `\ProvidesPackage{mypackage}`: identifiziert das Paket. Dateiname = Paketname!
- Wahlweise `[2007/05/31 v0.1 Meine Makros]` als opt. Arg
- Benötigte Pakete mit `\RequirePackage` laden
- Bestehende Befehle mit `\renewcommand` umdefinieren oder mit `\newcommand` neue Befehle anlegen
- `\ProvidesCommand` erzeugt neuen Befehl nur, wenn es ihn nicht schon gibt

Eigenes Paket für die heutige Übung:

```
\ProvidesPackage{DFB}[2007/05/26 v0.1
   Darstellung von Fussballergebnissen (HB)]
\newcommand{\spiel}[4]{% Stellt ein Spiel dar
#1 &gegen& #2 & #3 &:& #4\\}

\newenvironment{spieltag}%
% Stellt eine Tabelle für die Spiele eines
   Spieltages zur Verfügung
{Ergebnisse des Spieltages:\\ % begin-code
\tabular{rclccc}
}
{\endtabular} % end-code
```

und zur Einbindung: `\usepackage{DFB}`

Das ominöse @...

- Viele Pakete verbergen »interne« Befehle vor dem Benutzer
- Dazu dienen Makronamen mit einem »@«:
  `\@startsection,\@ne`
- »@« ist für Kommandonamen ein besonderes Zeichen (catcode 11)
- führt im normalen Modus `\mein@befehl` zu einer Fehlermeldung

Das ominöse @ (cont.)

- Manchmal werden aus newsgroup oder Anleitungen Befehle mit @ verwendet, um besondere Aufgaben zu lösen
- Nützlich z. B.:
  `\g@addto@macro{\<macroname>}{<Zus. Befehle>}` hängt an ein bestehendes Makro weitere Befehle an
- Kann in .sty-Files direkt verwendet werden
- im normalen Dokument mit
  `\makeatletter`
  <@-Befehle>...
  `\makeatother` kapseln!

»Echte« Pakete

```
\ProvidesPackage{sectsty}[2002/02/25 v2.0.2
Commands to change all sectional heading styles]
\NeedsTeXFormat{LaTeX2e}[1998/06/01]
\long\def\SS@ocl#1#2#3{\ifnum #1>\SS@chatlevel
   #2\else #3\fi}
\def\SS@oclto#1#2{\SS@ocl{#1}{\typeout{#2}}{}}
\DeclareOption{chatty}{\def\SS@chatlevel{1}}
\ProcessOptions
\let\@svsec\relax
\newcommand{\nohang}{\let\@hangfrom\@empty}
\let\SS@origunderline\underline
\newcommand*{\SS@ulemsectuline}[2]{%
  \ifhmode% run-in head
    \begingroup%
    \def\hskip##1\relax##2\@@par{
    \endgroup\@@hskip##1\relax#1{##2}}%
  \else
```

Dec@ding style files...

- ...uff!
- Packages verwenden eine Mischung aus low-level-TeX und »gewöhnlichen« Befehlen
- und jede Menge unverständliche Konstrukte
- Aussichtslos?
- Erster Einblick »Decoding Style Files«, in Literaturliste

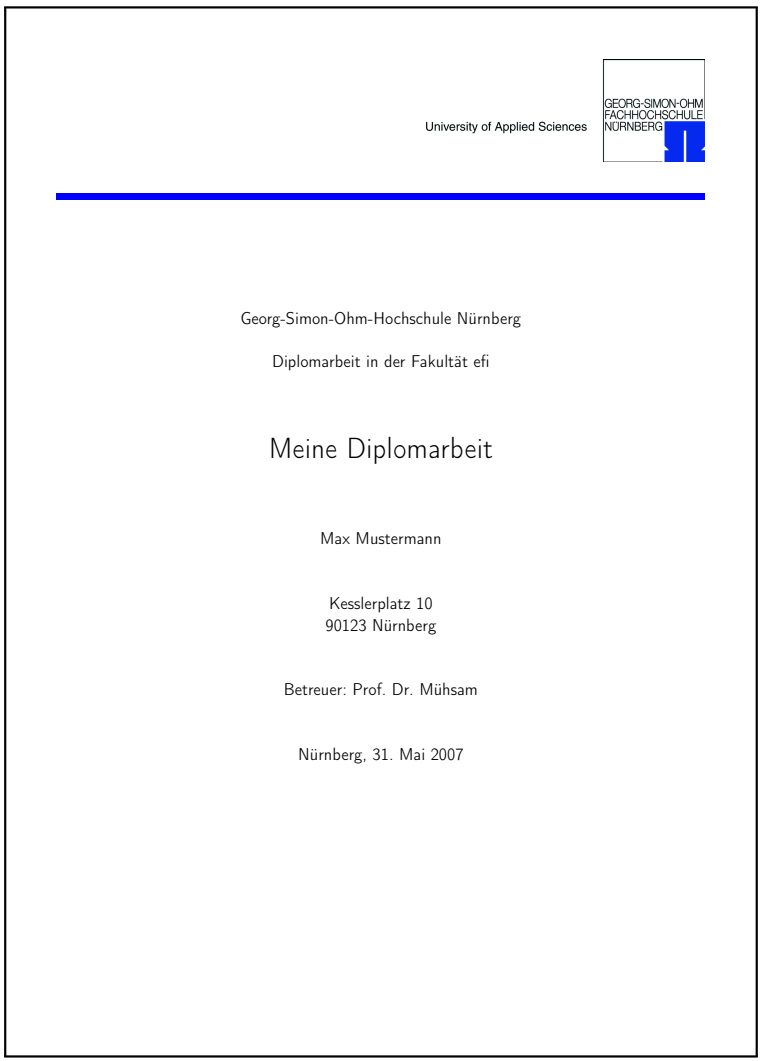
Beispiel-Stylefile: FH-Titelseite

Was wir wollen:

- Einbindung einer eigenen Titelseite
- mit dem selben Komfort wie Standardklassen
- also mit Belegung der Titelvariablen vor dem Aufruf von `\maketitle`:

  `\Autor{Max Mustermann}`

⇒ Eigenes Style-File (im zipfile zum Download)!



FH-Titelseite - Benutzung I

Einbinden der Titelseite als `\usepackage{LTXKursTitel}`
Sie haben Zugriff auf die neuen Variablen:

`\DAAutor`	Ihr Name
`\DATyp`	Art der Arbeit (vorbelegt mit »Diplomarbeit«)
`\DAAutorAdresse`	Ihre Adresse
`\DAFachbereich`	Fachbereich, z. B. efi
`\DATitel`	. . . na, was wohl?
`\DABetreuer`	Ihr Betreuer
`\DAOrt`	Nürnberg?
`\DAAbgabedatum`	(gestern *g*)

FH-Titelseite - Benutzung II

- Erinnern Sie sich an die erste Vorlesung?
  `\author{Ich}`,`\title{Mein Titel}`?
- Analog hier `\DA...`Befehle:
  `\DATitel{Meine revolutionäre Diplomarbeit}`.
- Layoutdetails (Ränder) per
  `\setlength{\RandLinks}{1cm}`...einstellbar
- `\maketitle` im Dokument generiert die Titelseite.

Übungsaufgaben zum Termin 9 - Teil 2

Öffnen Sie das Zipfile Uebung9.zip und extrahieren Sie die Dateien in Ihr Arbeitsverzeichnis.

- Binden Sie die Titelseite »LTXKursTitel« in ihr Projekt ein. Öffnen Sie das .sty-File (Winshell: »Öffnen« - alle Dateien) und lesen Sie die Dokumentation. Fragen?
- Vervollständigen Sie Ihre Angaben nach dem soeben erläuterten Muster

Dokumentation

Allgemeines Nachschlagewerk: LaTeX-Begleiter
Frank Mittelbach, Michel Goossens, Johannes Braams *et al.*: Der LaTeX-Begleiter. Pearson Studium 2. Auflage 2005.

LaTeX 2_ε for class and package writers:
`ftp://tug.ctan.org/pub/tex-archive/macros/latex/doc/clsguide.pdf`
oder durch `texdoc clsguide` auf der Eingabeaufforderung

Dec@ding Style Files: Was steckt im Detail hinter dem kryptischen Code (für TeXperten) - **Achtung, neue URL!**
`http://people.cs.uu.nl/andres/DecodingStyles.pdf`

The UK List of TeX Frequently Asked Questions:
`http://www.tex.ac.uk/cgi-bin/texfaq2html`

Fortsetzung . . .

. . . nächste Woche!