

The L^AT_EX Graphics Companion

Illustrating documents with
T_EX and PostScript



Addison-Wesley Series on Tools and Techniques for Computer Typesetting

This recently inaugurated series focuses on tools and techniques needed for computer typesetting and information processing with traditional and new media. Books in the series address the practical needs both of users and of system developers. Initial titles comprise handy references for L^AT_EX users; forthcoming works will expand that core. Ultimately, the series will cover other typesetting and information processing systems, as well, especially insofar as those systems offer unique value to the scientific and technical community. The series goal is to enhance your ability to produce, maintain, manipulate, or reuse articles, papers, reports, proposals, books, and other documents with professional quality.

Ideas for this series should be directed to the editor: mittelbach@aw.com.
All other comments and questions should be sent to Addison-Wesley: awcse@aw.com.

Series Editor

Frank Mittelbach
Manager L^AT_EX3 Project, Germany

Editorial Board

Jacques André
Irisa/Inria-Rennes, France

Barbara Beeton
Editor, TUGboat, USA

David Brailsford
University of Nottingham, UK

Tim Bray
Textuality Services, Canada

Peter Flynn
University College, Cork, Ireland

Leslie Lamport
Creator of L^AT_EX, USA

Chris Rowley
Open University, UK

Richard Rubinstein
Human Factors International, USA

Series Titles

The L^AT_EX Companion, by Michel Goossens, Frank Mittelbach, and Alexander Samarin

The L^AT_EX Graphics Companion, by Michel Goossens, Sebastian Rahtz, and Frank Mittelbach

Also from Addison-Wesley:

L^AT_EX: A Document Preparation System, Second Edition, by Leslie Lamport

The L^AT_EX Graphics Companion

Illustrating documents with T_EX and PostScript

Michel Goossens
CERN, Geneva, Switzerland

Sebastian Rahtz
Elsevier Science Ltd, Oxford, United Kingdom

Frank Mittelbach
L^AT_EX3 Project, Mainz, Germany



Addison-Wesley

An imprint of Addison Wesley Longman, Inc.
Reading, Massachusetts • Harlow, England • Menlo Park, California
Berkeley, California • Don Mills, Ontario
Sydney • Bonn • Amsterdam • Tokyo • Mexico City

12.5.554

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book and Addison Wesley Longman, Inc. was aware of a trademark claim, they are printed in initial caps or all caps.

The procedures and applications presented in this book have been included for their instructional value. They have been tested with care but are not guaranteed for any particular purpose. The publisher offers no warranties or representations, nor does it accept any liability with respect to the programs or applications.

Library of Congress Cataloging-in-Publication Data

Goossens Michel.

The LaTeX graphics companion : illustrating documents with TeX and PostScript / Michel Goossens, Sebastian Rahtz, Frank Mittelbach
p. cm.

Includes bibliographical references and index.

ISBN 0-201-85469-4

1. LaTeX (Computer file) 2. Computerized typesetting.
3. PostScript (Computer program language) 4. Scientific
illustration--Computer programs. 5. Mathematics printing--Computer
programs. 6. Technical publishing--Computer programs. I. Rahtz,
S. P. Q. II. Mittelbach, Frank. III. Title.
Z253.4.L38G663 1997

686.2'2544536---dc21

98-39081

CIP

*To those hundreds of LaTeX developers
for their enthusiasm and hard work.*

*Special thanks to our families
without whose infinite patience
this book would never have seen the light.*

Reproduced by Addison Wesley Longman, Inc., from camera-ready copy supplied by the authors.

Copyright © 1997 by Addison Wesley Longman, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America.

ISBN 0-201-85469-4

2 3 4 5 6 7 CRS 00 99 98 97

2nd Printing November 1997

Contents

Preface	xxi
1 Graphics with L ^A T _E X	1
1.1 Graphics systems and typesetting	2
1.2 Drawing types	3
1.3 T _E X's interfaces	6
1.3.1 Methods of integration	6
1.3.2 Methods of manipulation	9
1.3.3 T _E X's graphic hooks	9
1.4 Graphics languages	11
1.4.1 T _E X-based graphics languages	11
1.4.2 External graphics languages and drawing programs	20
1.5 Choosing a package	25
2 The L ^A T _E X 2 _ε graphics bundle	27
2.1 Loading the graphics packages	27
2.1.1 Package device driver options	28
2.2 Inclusion of graphics files	29
2.2.1 <code>\includegraphics</code> syntax in the <code>graphics</code> package	30
2.2.2 <code>\includegraphics</code> syntax in the <code>graphicx</code> package	32
2.2.3 Setting default key values for the <code>graphicx</code> package	38
2.2.4 Declarations guiding the inclusion of images	39
2.3 Graphical manipulation of L ^A T _E X objects	41
2.3.1 Scaling a L ^A T _E X box	42
2.3.2 Resizing to a given size	43
2.3.3 Rotating a L ^A T _E X box	44

2.3.4	Combining effects	48
2.3.5	Applications	48
2.4	Other packages in the graphics bundle	50
2.4.1	The epsfig and rotating packages	50
2.4.2	The lscape package	51
2.4.3	The trig package	51
2.4.4	The keyval package	51
3	Working with METAFONT and METAPOST	53
3.1	The META language	54
3.1.1	Examples of META programs	55
3.2	Using the META programs	60
3.2.1	Running METAFONT	60
3.2.2	An alternative to METAFONT — METAPOST	62
3.3	METAPOST macro libraries	68
3.3.1	Boxing macros	68
3.3.2	The METAPOST graph package	72
3.4	L ^A T _E X interfaces to META	83
3.4.1	The mfpic package	84
3.5	mftoops: a direct link between METAFONT and PostScript	93
4	Harnessing PostScript inside L ^A T _E X: the PSTricks package	95
4.1	The components of PSTricks	95
4.2	Basic PSTricks concepts	97
4.2.1	Commands and arguments	97
4.2.2	Setting graphics parameters	98
4.2.3	Coordinates and units	98
4.2.4	Coloring objects	100
4.3	The graphic objects	100
4.4	Examples of basic graphic objects	102
4.5	Mixing text and graphics	107
4.6	Nodes and their connections, and trees	114
4.6.1	Matrices — grid-based nodes	121
4.6.2	Tree diagrams	125
4.6.3	Specifying tree bounding boxes	132
4.7	Data plotting	132
4.8	Working with a third dimension	135
4.9	Iterating commands	139
4.10	Customizing and programming PSTricks	140
4.10.1	New PSTricks objects and styles	140
4.10.2	PSTricks programming examples	145
4.11	Other PSTricks tools	153
4.12	Driver configuration for PSTricks	153
4.13	Summary of PSTricks commands and parameters	154

4.13.1	PSTricks basic drawing commands	154
4.13.2	PSTricks node-drawing commands	159
4.13.3	PSTricks node connection labeling commands	161
4.13.4	PSTricks drawing commands comparable to node connectors	163
4.13.5	PSTricks tree-drawing commands	163
4.13.6	PSTricks plotting commands	165
4.13.7	PSTricks 3D commands	166
5	The Xy-pic package	167
5.1	Introducing Xy-pic	167
5.2	A first example of Xy-pic code	168
5.3	Basic constructs	169
5.3.1	Initial positions	169
5.3.2	Making connections	170
5.3.3	Dropping objects	171
5.3.4	Entering text in your pictures	173
5.4	Extensions	174
5.4.1	Curves and splines	175
5.4.2	Frames and brackets	176
5.5	Features	178
5.5.1	Arrows	178
5.5.2	Matrix-like diagrams	180
5.5.3	Graphs	184
5.5.4	Two-cell diagrams	188
5.5.5	Polygons	190
5.5.6	Arcs, circles, and ellipses	195
5.5.7	Lattices and web structures	197
5.5.8	Links and knots	198
6	Applications in chemistry, physics, and engineering	205
6.1	Typographical rules for scientific texts	205
6.1.1	Typesetting chemical symbols	207
6.2	The X _Y M _T E _X system	208
6.2.1	General conventions	210
6.2.2	The carom package	214
6.2.3	The lowcycle package	215
6.2.4	The hetarom and hetaromh packages	216
6.2.5	The ccycle and hcycle packages	218
6.2.6	The aliphat package	219
6.2.7	Combining structures	220
6.2.8	Defining your own structures	220
6.3	The ppchtex package	222
6.3.1	Structures	222
6.3.2	Definitions	224

6.3.3	Bonds	225
6.3.4	Combinations	225
6.3.5	Chemical equations	227
6.3.6	Special features	227
6.4	Drawing Feynman diagrams	228
6.4.1	Using FeynMF	229
6.4.2	Writing FeynMF diagrams	231
6.4.3	Extending FeynMF	239
6.5	Typesetting timing diagrams	239
6.5.1	Commands in the timing environment	239
6.5.2	Customization	243
6.6	Electronics and optics diagrams	243
6.6.1	General circuit diagram commands	245
6.6.2	Examples	247
6.7	Using the m4 macro processor for electronics diagrams	250
6.7.1	Basic principles	250
6.7.2	Customizing the diagram	251
7	Preparing music scores	253
7.1	Using TeX for scores — an overview	254
7.2	Using MusiXTeX	255
7.2.1	The structure of a MusiXTeX source	255
7.2.2	Writing notes	256
7.2.3	Note spacing	260
7.2.4	A moderately complete example	261
7.2.5	Running MusiXTeX	263
7.3	abc2mtex — easy writing of tunes	264
7.3.1	Writing abc2mtex source	265
7.3.2	Running abc2mtex	268
7.4	M ^P , a MusiXTeX preprocessor	269
7.4.1	Writing M ^P -source	269
7.4.2	Running M ^P	273
7.5	midi2tex	274
7.5.1	Running midi2tex	274
8	Playing games	277
8.1	Chess	277
8.1.1	A METAFONT chess font	278
8.1.2	chess — a package for typesetting chess	278
8.1.3	Extensions to the chess package	284
8.1.4	Interfacing chess databases	285
8.1.5	Using Adobe's cheq font	288
8.2	Xiangqi — Chinese chess	292
8.3	Go	295

8.3.1	Possible problems	298
8.4	Backgammon	299
8.5	Card games	302
8.6	Bridge	303
8.6.1	Card deals	303
8.6.2	Bidding	305
8.7	Crosswords	306
8.7.1	Recent developments	309
9	The world of color	311
9.1	An introduction to color	312
9.1.1	Color theories	312
9.1.2	Color systems	313
9.1.3	Symbolic values of color	314
9.1.4	Color harmonies	314
9.1.5	Color and readability	316
9.2	Colors and L ^A T _E X — the color package	317
9.2.1	Supported options	317
9.2.2	Using colors	318
9.2.3	Page color	321
9.2.4	Colored box backgrounds	322
9.2.5	Calculating colors	324
9.3	Coloring tables	326
9.3.1	The colortbl package	326
9.3.2	Examples	329
9.4	Color slides with L ^A T _E X: the seminar class	338
9.4.1	Using the seminar class	339
9.4.2	Frame styles	339
9.4.3	Interleaving notes and selecting subsets	341
9.4.4	Controlling slide size, fonts, and magnification	341
9.4.5	Fonts	342
9.4.6	The local control file	342
9.5	Color in the printing industry and separation	346
9.5.1	Color separation	347
9.5.2	Color separation using L ^A T _E X and dvips	348
10	Using PostScript fonts	351
10.1	Using preconfigured PostScript fonts	353
10.1.1	The PSNFSS system	353
10.1.2	Fonts, metric files — the whole lot	359
10.1.3	Installing PostScript fonts	362
10.2	TeX font technology	363
10.2.1	Types of TeX fonts	363
10.2.2	TeX font metric files	364

10.2.3	TeX virtual fonts	367
10.3	PostScript font technology	370
10.3.1	Types of PostScript fonts	370
10.3.2	Adobe Font Metric files	375
10.3.3	Encoding of PostScript fonts	378
10.3.4	Rendering PostScript fonts	379
10.4	Classifying PostScript fonts	381
10.5	Setting up new PostScript fonts	386
10.5.1	TeX and encoding	386
10.5.2	The TeXBase1 encoding	386
10.5.3	Making TeX metrics for PostScript fonts	387
10.5.4	AFM to TFM, and VF, conversion tools	387
10.5.5	The Y&Y font manipulation tools	389
10.5.6	The afm2tfm program	390
10.5.7	The fontinst package	393
10.6	Multiple Master fonts — a case study	406
10.6.1	Generating instance files using Ghostscript	408
10.6.2	The Multiple Master setup of this book	410
11	PostScript drivers and tools	413
11.1	Introduction to dvi drivers	413
11.2	The dvips PostScript driver	415
11.2.1	Command line and configuration file options	415
11.2.2	Paper sizes	422
11.2.3	\special support	424
11.2.4	Font support	428
11.2.5	Special hooks	429
11.2.6	Debugging	432
11.3	PostScript page-manipulation tools	433
11.3.1	The psutils suite	435
11.4	Ghostscript, a PostScript interpreter	442
11.4.1	Ghostscript options and initialization	443
11.4.2	Ghostscript and fonts	446
11.4.3	Using different devices with Ghostscript	447
11.4.4	Interactive Ghostscript versions	450
11.4.5	Ghostscript applications	450
11.5	PostScript font to PK font format conversion	453
11.5.1	The ps2pk program	453
11.5.2	The gsftopk program	455
11.6	Generating images for Web pages using dvips and Ghostscript	455
11.7	PSfrag — adding labels to included pictures	460
A	Technical appendixes	463
A.1	emTeX \special commands	463

A.2	tpic \special conventions	464
A.3	The bm2font program	467
A.4	The dvips color separation header file	468
A.5	Catalogue of typefaces with <i>Fontname</i> abbreviations	471
A.6	Font encoding tables	474
A.7	fontinst reference information	487
A.7.1	Encoding files	487
A.7.2	Metric files	488
A.7.3	Low-level fontinst commands	491
A.8	Ghostscript drivers	494
B	Getting all the goodies	497
B.1	Connecting to CTAN	498
B.1.1	Finding files on the archive	498
B.2	Getting a package from the archive	498
B.3	List of packages and programs	504
	Bibliography	511
	Index	517

List of Tables

2.1	Overview of color and graphics capabilities of device drivers.	29
2.2	Arguments of \DeclareGraphicsRule.	41
4.1	PSTricks supporting packages.	96
4.2	PSTricks letter abbreviations for common angles.	102
4.3	PSTricks line terminators.	103
4.4	PSTricks dot styles.	106
4.5	PSTricks basic graphical parameters.	156
4.6	PSTricks graphical parameters for node connectors	161
4.7	PSTricks graphical parameters for trees	164
4.8	PSTricks graphical parameters for plots.	166
4.9	PSTricks 3D graphical parameters.	166
6.1	The importance of typographic rules in scientific texts.	207
6.2	Application commands of the hetarom package.	217
6.3	Bond identifiers for ppchtex.	225
6.4	FeynMF vertex and fill styles.	233
6.5	FeynMF linestyles.	233
6.6	FeynMF line-drawing options.	235
6.7	FeynMF vertex-drawing options.	235
6.8	FeynMF polygon options.	236
6.9	Symbol combinations in all font variants of the timing package.	242
6.10	Some basic electronic circuit symbols.	244
6.11	Some integrated circuit symbols.	245
6.12	Some optic diagram symbols.	246

7.1	Overview of MusiX _{TeX} commands.	257
7.2	Variant forms of the <code>\notes</code> command	260
8.1	Coding for chess pieces in the <code>chess</code> fonts.	278
8.2	Coding for xiangqi pieces in the <code>cchess46</code> font.	293
10.1	Effect of PSNFSS package files.	354
10.2	Sample texts for standard PostScript fonts.	355
10.3	The standard PostScript fonts in the <i>Fontname</i> scheme.	381
10.4	Font suppliers defined in the <i>Fontname</i> scheme.	383
10.5	<i>Fontname</i> weight codes.	383
10.6	<i>Fontname</i> variant codes.	384
10.7	<i>Fontname</i> width codes.	385
10.8	Multiple Master instance values matched to <i>Fontname</i> and L ^A T _E X NFSS codes.	411
10.9	Minion Multiple Master instances for <i>Fontname</i> combinations.	411
11.1	Overview of the tools in the <code>psutils</code> set.	437
A.1	Summary of <code>tpic \special</code> requirements.	465
A.2	Command-line options for <code>bm2font</code>	466
A.3	Font encoding table, by name.	474
A.4	Font encoding table, by number.	482
B.1	Location of the packages and programs mentioned in this book.	504

List of Figures

1.1	Pen and ink drawing of a bead.	4
1.2	Bitmap drawing output created with MacPaint.	4
1.3	Digitally transformed image (vertically stretched).	5
1.4	Object-oriented drawing.	5
1.5	Scanned cartoon converted to font via <code>pbm</code> format.	8
1.6	Color <code>gif</code> file converted to T _E X font files by <code>bm2font</code>	8
1.7	Using ASCII symbols to make a histogram.	12
1.8	Bitmap input to generate the symbol in Fig. 1.9.	13
1.9	A “sprite” logo.	13
1.10	An expanded “sprite”.	13
1.11	Output of <code>bit2spr</code>	13
1.12	P _I C _T E _X source code and graph output.	14
1.13	P _I C _T E _X source code and histogram output.	15
1.14	Histogram drawn with P _I C _T E _X , including rotation of text	16
1.15	Piechart produced with <code>aldratex</code>	17
1.16	The <i>lcircuit</i> library of symbols.	18
1.17	Example of flow language.	19
1.18	AutoCAD plotter output converted to METAFONT.	21
1.19	Graph generated by <code>gnuplot</code> using L ^A T _E X picture commands.	21
1.20	Simple example of <code>pic</code> code.	23
1.21	Example of <code>pic</code> code showing use of stored names.	23
1.22	Example of <code>grap</code> output.	24
2.1	The contents of the file <code>w.eps</code>	30
2.2	A L ^A T _E X box and possible origin reference points.	46

3.1	META picture after Naum Gabo, by Kees van der Laan	59
3.2	Finite-state diagram drawn with METAPOST	73
3.3	AutoCAD map converted to METAFONT.	94
3.4	METAFONT drawing enhanced using Corel Draw.	94
4.1	A processor grid, programmed by Denis Girou.	141
4.2	A random walk in PSTricks, programmed by Denis Girou.	144
4.3	A polygon language written in PSTricks by Denis Girou.	146
6.1	Periodic table typeset with L ^A T _E X and chemsym	209
6.2	Some predefined structures in PPCH _T E _X	223
6.3	Simple example of the feynmf package.	229
6.4	Varying the tension parameter of a single line.	232
6.5	Drawing a Feynman diagram in vertex mode.	237
6.6	Vertex mode and improved version using immediate-mode commands.	238
6.7	Timing diagram of a memory read followed by a memory write.	241
7.1	Sequence of score pieces coded in MusiX _T E _X	256
8.1	Chess example — White to move without mating Black.	279
8.2	Example of L ^A T _E X source and output of a chess game.	281
8.3	Chess tournament notation.	283
8.4	Additional chessboard visualization commands.	284
8.5	Chess-Postcard input for bdfchess.	286
8.6	Chess postcards generated by bdfchess.	287
8.7	Font encodings for chess10 and cheq	290
8.8	Solution — White to move without mating Black.	292
8.9	Initial setup of Chinese chess game (xiangqi).	294
8.10	A mate situation after four moves.	295
8.11	The Go symbol.	296
8.12	A Go game at an early stage.	297
8.13	Two-stage ko (favorable to Black).	298
8.14	A backgammon example.	299
8.15	First moves of a backgammon game.	301
8.16	A sample crossword for you to fill in.	307
8.17	Solution to crossword in Fig. 8.16.	309
9.1	Example of simple color separations.	349
9.2	Color separation of a bitmap image using aurora.	349
9.3	Detail of color separation of a bitmap image.	350
10.1	Math sample set in Times with Computer Modern font.	356
10.2	Math sample set in Times with partial Times math (mathptm package).	357
10.3	Files and processes used by L ^A T _E X, a d _v i-to-PostScript driver and PostScript.	361

10.4	Multiple Master typeface design space.	373
10.5	Two-dimensional matrix showing Multiple Master Myriad font	373
10.6	The three axis Multiple Master Minion serif font	374
10.7	A font generated for 85 dpi (e.g., computer screen).	380
10.8	A font generated for 300 dpi (e.g., LaserWriter).	380
10.9	An outline font.	380
10.10	The TeXBase1 font layout (Lucida Bright).	388
10.11	Math test with mixture of Minion Multiple Master and Lucida New Math.	405
10.12	Minion instances from opposite ends of the optical size axis set at the same size (exaggerated).	407
11.1	Stamping pages using dvips	430
11.2	Mirror-image printing with dvips.	432
11.3	Multiple logical pages on one physical sheet, using multi.pro.	434
11.4	Nine logical pages on one output page.	436
11.5	Example of the use of ghostview.	451
11.6	Bitmap EPS file, enlarged and distorted.	456
11.7	Outline font EPS file, enlarged and distorted.	456
11.8	A Perl script to transform an EPS file for Ghostscript.	459
11.9	L ^A T _E X to dvi to EPS to GIF	460
A.1	Examples of using bm2font.	467
B.1	Peter Flynn's CTAN Web interface	499
B.2	Graham Williams' CTAN catalogue on the Web.	500
B.3	Viewing CTAN files on the Web.	501
B.4	Transfer of a directory as a zip archive.	502

Color Plates

I	METAPOST examples.	I
II	PSTricks examples 1.	II
III	PSTricks examples 2.	III
IV	PSTricks examples 3.	IV
V	PSTricks examples 4.	V
VI	PSTricks examples 5.	VI
VII	The relation between the RGB and CMYK color models.	VII
VIII	The HSB model.	VII
IX	Symbolic connotation of colors in different countries	VII
X	Color harmony and the primary colors.	VIII
XI	Color harmonies and the chromatic circle (after Itten, 1974).	VIII
XII	Color contrasts for optimizing visibility and readability	IX
XIII	Predefined “Crayola” colors in the <i>named</i> color model for the dvips driver.	X

XIV	Examples of L ^A T _E X color package.	XI
XV	Simple multicolor table.	XII
XVI	Coloring table rules.	XII
XVII	Two-color tables (part I).	XII
XVIII	Two-color tables (part II).	XIII
XIX	Two-color tables (part III).	XIV
XX	Some examples of using color with seminar.	XV
XXI	The separation of colors in the CMYK model.	XVI

Preface

In this book we describe techniques and tricks of extended L^AT_EX typesetting in the area of graphics and fonts. We examine how to draw pictures with L^AT_EX and how to incorporate graphics files in a L^AT_EX document. We explain how to program pictures using METAFONT and METAPOST, as well as how to achieve special effects with small fragments of embedded PostScript. We show how to manipulate whole T_EX pages in PostScript and how to exploit all the features of PostScript Type 1 fonts. And, of course, we also look in detail at a whole range of tools for building graphics in T_EX itself.

T_EX is the world's première markup-based typesetting system, and PostScript is the leading language for describing the printed page. This book describes how they can produce even more beautiful results when they work together. T_EX's mathematical capability, its paragraph building, its hyphenation and its programmable extensibility can cooperate with the graphical flexibility and the font-handling capabilities of PostScript, to provide a rich partnership for both author and typesetter. We describe dvi-to-PostScript drivers, look at tools for manipulating PostScript files, and discuss in detail the features of the free program Ghostscript which lets you view or print PostScript files even if you do not have a PostScript printer.

This volume is not a complete consumer guide to packages. In trying to teach by example we present hundreds of examples of the most useful types of solutions based on proven and well-known implementations. But, in the space available, we cannot provide a full manual for every package. Our main aim is to show how easy it is to use a given package and whether or not it seems to do what is required, not to dwell on precise details of syntax or run options. Nevertheless, we have described in more detail a few selected tools that we consider especially important.

We assume you know some L^AT_EX; you cannot read this book by itself if you have never used T_EX before. We recommend that you start with *L^AT_EX: A Document Preparation System, Second Edition* (Lamport, 1994), and continue with *The L^AT_EX Companion* (Goossens et al., 1994) to explore some of the many (non-graphical) packages available.

Why L^AT_EX, and why PostScript?

This book is about L^AT_EX, graphics, and PostScript. We believe that the structured approach of a system like L^AT_EX is the best way to use T_EX, and L^AT_EX is by far the most widely used T_EX format. This means that it attracts those developing new packages (especially since the major revision of L^AT_EX in 1993), and thus some of what we describe works only in L^AT_EX (this applies especially to material to do with font selection schemes). We apologize in advance for our L^AT_EX bias to those who appreciate the elegance of the original plain T_EX format and its derivatives and promise them that most of the packages work well with any T_EX dialect: the delights of systems like METAPOST, PSTricks, Xy-pic, and MusiX_TE are open to all.

We also want to explain why we talk about PostScript so much. This language has been established for over a decade as an extremely flexible page-description language and it remains the tool of choice for professional typesetters. Among the features that make it so attractive are:

- the quantity, quality, and flexibility of Type 1 fonts;
- the device-independence and portability of files;
- the quality of graphics and the quantity of drawing packages that generate it;
- the facilities for manipulating text;
- the mature color-printing technology;
- the encapsulation conventions that make it easy to embed PostScript graphics;
- the availability of screen-based implementations (Display PostScript and Ghostscript/GhostView).

In the last few years PostScript has spawned an enterprising child, the PDF (Portable Document Format) language used by Adobe Acrobat. Designed for screen display with hypertext features, PDF offers a new degree of portability and efficiency. While we do not cover PDF here, L^AT_EX works well in producing “rich” PDF documents, and special versions of T_EX that produce PDF directly will probably be available by the time this book is published.

Again, we apologize to those of you who are disappointed not to read about L^AT_EX’s association with QuickDraw, TrueType fonts, the Windows GDI, HPGL, PCL, etc., but with so many packages available, we had to make a choice.

Please note that the absence of a given package or tool in this book in no way implies that we consider it as less useful or of inferior quality. We do think, though, that we have included a representative set of tools and packages, and we sincerely hope that you will find here one or more subjects to entertain you.

How this book is arranged

This book is subdivided in two basic ways: by application area, and by technique. We suggest that all readers look at Chapter 1 before going any further, since this introduces how we

think about graphics and summarizes some techniques developed in later chapters. We also suggest you read Chapter 2, which covers the L^AT_EX standard graphics package, since the tools for including graphics files will be needed often.

We have tried to make it possible to read each of the other chapters separately; you may prefer to go straight to the chapters that cover your subject area or look at those that describe a particular tool. Three chapters are dedicated to particular programs and packages:

- 3 **Working with METAFONT and METAPOST** shows how to exploit the power of Knuth’s METAFONT language; this can work well with L^AT_EX and provides unparalleled expressive power for describing many types of diagrams.
- 4 **Harnessing PostScript inside L^AT_EX: the PSTricks Package** describes what we think is the most powerful system to integrate L^AT_EX closely with the PostScript language.
- 5 **The Xy-pic Package** introduces a package that goes to great lengths to define a notation for many kinds of mathematics diagrams and implements it in a generic and portable way.

Then follow three chapters that discuss problems in special application areas and survey some packages in each:

- 6 **Applications in chemistry, physics, and engineering** looks at chemical bonds, Feynman diagrams, and circuit diagrams.
- 7 **Preparing music scores with T_EX** considers the powerful MusiX_TE package and some of the support programs that you can use to print beautiful musical scores.
- 8 **Playing games** is for those who use L^AT_EX for play as well as work; we show how to typeset chess boards, annotate diagrams for bridge or other card games, or even handle backgammon and Go.

Our last three chapters address areas of general interest, mostly not related to particular L^AT_EX packages or subject areas:

- 9 **The world of color** provides full details about the standard L^AT_EX color package and also describes other packages based on it. We also look at some issues of color design and problems related to color printing.
- 10 **Using PostScript fonts** describes the ins and outs of using PostScript fonts with L^AT_EX, including AFM files, virtual fonts, font naming, and Multiple Master fonts.
- 11 **PostScript drivers and tools** provides a complete reference to some of the most commonly used free software packages—dvips, Ghostscript, the psutils suite, and programs to convert PostScript fonts to PK format.

Finally the appendixes contain technical information that complements the above chapters. They also give the Internet location of all software described in this book and show sample sessions to help you download it in different ways.

Using, and finding, all those packages and programs

We describe many packages and options that extend or modify $\text{\LaTeX}$'s basic possibilities. In order to show their action, we must in principle load them all at the same time. For various reasons, however, that is impractical, if not impossible. Indeed, many packages use a lot of internal variables for which most implementations of $\text{\TeX}$ have fixed limits. In producing this book we used a different strategy (pioneered with *The $\text{\LaTeX}$ Companion*): we ran nearly all the examples as separate jobs and included them as EPS files. In almost all cases this was entirely automated—the code printed here is exactly that used to prepare the output. To help you run the examples yourself, each piece of $\text{\LaTeX}$ code also shows the `\usepackage` commands needed to run it.

We have put the source of the examples on CTAN (Comprehensive $\text{\TeX}$ Archive Network—see Appendix B) in the directory `info/lgc`. Most examples are identified by a number enclosed in a box. This means that the exact code used to create the picture is available, with the identifying number as filename. Most files are in $\text{\LaTeX}$ source format (with an extension of `ltx`) but there are also plain $\text{\TeX}$ files (extension `ptx`), `METAPOST` source files (extension `mp`), `Mu $\text{\TeX}$` preprocessor source files (extensions `abc` and `mpp`), `pic` files (extension `pic`), and `m4` sources (extension `m4`).

All the packages and programs described in this book are freely available in public software archives; some are in the public domain, while others are protected by copyright. Some programs are only available in source form or work only on certain computer platforms, and you should be prepared for a certain amount of “getting your hands dirty” in some cases. We also cannot guarantee that later versions of packages or programs will give results identical to those in our book. Many of them are under active development, and new or changed versions appear several times a year; we completed this book at the end of 1996, and tested the examples with the versions current at that time.

In Appendix B are given full details on how to access CTAN sites and how to download files using the Internet. You can also purchase a CD-ROM from the $\text{\TeX}$ Users Group containing implementations of $\text{\TeX}$ for various systems and many packages and fonts. This CD-ROM contains all the packages described in this book and *The $\text{\LaTeX}$ Companion*.

Acknowledgments

This book has two sources. First, it descends from a survey by Sebastian Rahtz of $\text{\TeX}$ and graphics originally prepared for the $\text{\TeX}$ eter conference in 1988, revised many times and distributed as a technical report in 1989 by Southampton University Computer Science Department,¹ and from Rahtz' ongoing support and documentation for using PostScript fonts in $\text{\LaTeX}$. Second, it descends from material that Michel Goossens, Frank Mittelbach, and Alexander Samarin could not include in *The $\text{\LaTeX}$ Companion*. It became clear, long before that book was finished, that it could not do full justice to graphics and PostScript and that another book (at least) would be needed. This book was conceived, born, and brought up at

¹Sebastian Rahtz would like to thank Professor David Barron and Les Carr at Southampton University for their shared learning experience, expertise, and encouragement when we all first learnt $\text{\TeX}$ ten years ago.

CERN, arising from text-processing support for this great laboratory by Michel Goossens, assisted in 1993–1994 by Sebastian Rahtz.

We have many people to thank. Our primary debt, of course, is to the authors of the programs and packages we describe. Every author whom we contacted to discuss problems provided us with practical help in the spirit of the $\text{\TeX}$ community, and often gave us permission to reuse examples from their documentation.

Christel Mittelbach substantially improved the music chapter and Leonor Barroca spent a great deal of time on the `METAPOST` and `METAPOST` chapter—without their work we would never have finished this book.

We gratefully recognize all our many colleagues in the $\text{\TeX}$ world who developed the packages, not only those described here, but also the hundreds of others that help users typeset their documents faster and better. Without the continuous effort of all these enthusiasts, $\text{\TeX}$ would not be the magnificent and flexible tool it is today. We tried to do them justice by mentioning, when first describing a given package, the original author and/or other major contributors.

We are greatly indebted to Dwight Aplevitch, Barbara Beeton, Karl Berry, David Carlisle, Malcolm Clark, L. Peter Deutsch, Denis Girou, John Hobby, Alan Hoenig, Ross Moore, Thorsten Ohl, Kris Rose, Tom Rokicki, Chris Rowley, Virgil Stokes, Geoffrey Tobin, Ulrik Vieth, and Timothy van Zandt for their careful reading of various chapters of the manuscript. Their numerous comments, suggestions, corrections, and hints have substantially improved the quality of the text.

On the publishing side, we wish to thank Peter Gordon, our editor at Addison Wesley Longman, Inc., who gave us much-needed support and encouragement over the three years duration of this project. When it came to production, Avanda Peters and Rosa Gonzalez were unfailingly patient with our idiosyncrasies and steered us safely to completion. Katrina Avery edited our dubious prose into real English, and Art Ogawa took great care in overseeing the typesetting and the color pages; we greatly appreciate their work.

Feedback

We would like to ask you, dear reader, for your collaboration. We kindly invite you to send your comments, suggestions, or remarks to any of the authors. We shall be glad to correct any mistakes or oversights in a future edition, and are open to suggestions for improvements or the inclusion of important developments we may have overlooked. We will maintain a list of errata in a file called `grphcomp.err` in the $\text{\LaTeX}$ distribution, and this will contain current addresses for the authors.

Graphics with L^AT_EX

Every picture tells a story, he sang, but while L^AT_EX is good at helping you typeset your words in a beautiful manner, L^AT_EX manuals usually tell you little or nothing about producing and including graphics with your text. This book attempts to fill that gap by showing you the tools and T_EXniques developed over the last decade that let you generate, manipulate, and integrate graphics with your text.

In these days of the multimedia PC, graphics appear in various places. With many products we get collections ready-to-use of clipart graphics, and in shops we can buy CD-ROMs with “the best photos” of important places, and so forth. All such graphics can be included in a L^AT_EX document, as we will see in this and the following chapters. To do this they must be available in a suitable format, but fortunately many popular graphic formats either are directly supported or a conversion program exists, that allows transformation into a supported representation.

If you want to become your own graphic artist, you can often use stand-alone dedicated drawing tools (e.g., xfig or idraw on Unix, Adobe Illustrator, Corel Draw or FreeHand on Mac or PC, and many others). Or if you use one of the modern calculation tools like Mathematica, Maple, or MATLAB, you can produce graphics by using one of their many graphical output representations. If you are the lucky owner of a scanner or even of a digital camera, you can produce digital images of photos, hand-drawn pictures, or other graphics using these tools and their accompanying software. In all these cases it is easy to generate files that can be directly included in the L^AT_EX source through the commands of the graphics package described in Chapter 2.

But there are also many occasions when closer integration with the typesetting system than that offered by such programs is desirable, and this too can be done with L^AT_EX. Such integration is necessary if you want to use the same fonts in text and graphics, or more generally if the “style” of the graphics should depend on the overall style of the document. Close

integration of graphics with the surrounding text clearly requires generation of the graphic by the typesetting system itself, since otherwise any change in the document layout style requires extensive manual labor and the whole process becomes very error-prone.

* * *

In this chapter we consider graphic objects from various angles. We start with a look at the various requirements that are posed on graphic objects depending on the application used. This is followed by an analysis of the types of drawings that appear in documents and the strategies typically employed to generate, integrate, and manipulate such graphics. We then change focus and discuss the interfaces offered by T_EX for dealing with graphic objects and how these interfaces can be used successfully.

Armed with this knowledge, we finally plunge into a short survey of graphics languages built within and around T_EX. All this will help you select the right set of tools for the job at hand from those described in the book. In fact, this chapter offers more than that, as it also gives examples of languages and approaches not covered in detail elsewhere in the book. This survey will provide you with enough information to decide whether or not to follow the pointers and obtain such a package for a particular application.

1.1 Graphics systems and typesetting

When speaking about graphic objects, we first should ask ourselves what we understand by the term. One extreme position is to view everything put on paper as a graphic object, including the characters of the fonts used. This quite revolutionary view was in fact adopted in the design of the page description language PostScript, in which characters can be composed and manipulated by exactly the same functions as other graphic objects (we see some examples of this in Chapter 4).

However, most typesetting systems, including T_EX, do not try to deploy such a general model but instead restrict their functional domain to a subset of general graphic objects, e.g., by providing very sophisticated functions to place characters, resolve ligatures, etc., but omitting operators to produce arbitrary lines, construct and fill regions, and so forth. As a result the term “graphics” for most L^AT_EX users is a synonym for “artwork” and the fact that L^AT_EX already has a graphics language—the picture mode—is often forgotten.

When discussing graphical capabilities of an ideal typesetting system, we must remember that different applications have different, sometimes conflicting requirements:

- One extreme is the need for complete portability between platforms; another is to take into account even differences in the way printers put ink onto paper.
- A graphic might need to be correctly scaled to a certain size depending on factors of the visual environment created by the typesetting system, e.g., on the measure of the text.
- However, it is also possible that parts of the graphic should not scale linearly: e.g., it might be important for readability to ensure that textual parts of a graphic do not become smaller or larger than some limit. It might also be required that, when a graphic

is scaled by, say, 10 percent to fit the line, any included text must stay the same, so as to avoid making it larger than the characters in the main document body.

- It might be required that the graphical object be closely integrated with the surrounding text, e.g., by using the same fonts as in other parts of the document or more generally by containing objects that should change their appearance if the overall style of the document is changed. (The latter is especially important if the document is described by its logical content rather than by its visual appearance, with the intention of reusing it in various contexts and forms.)

As L^AT_EX is a general-purpose typesetting system used for all types of applications, the above requirements and more might arise in various situations. As we see throughout, a large number of them can be handled with grace if not to perfection. In some cases an appropriate solution was anything else than obvious and developing the mature macro packages and programs we have now took a decade or more.

1.2 Drawing types

The typology of graphics at the beginning of this chapter focused on the question of the integration with the L^AT_EX system, and divided the graphics into externally and internally generated ones. A different perspective would be to start from the types of graphics we might encounter in documents and discuss possible ways to generate and incorporate them. So let us list briefly the different types of pictures we might find in a document.

- Pictures can be drawn free-hand without a computer, such as the drawing of a glass bead in Fig. 1.1. For use in L^AT_EX, such a graphic must be transformed into a digital image, using, for example, a scanner. It can then be included and to some extent manipulated, e.g., rotated with the commands provided by the standard L^AT_EX graphics package described in Chapter 2. The caption and legend is normally produced using L^AT_EX tools to allow proper referencing from within the document, but other than that the graphic is treated as a single object with no accessible inner structure—a “black box”.
- “Art” graphics drawn with bitmap tools on a computer, such as the example in Fig. 1.2, are to some extent the computer equivalents of pen and ink drawings. This drawing was created with MacPaint; the low screen resolution of the original is very obvious, and the diagonal lines are poorly defined. But the distinctive characteristic of this type of drawing is that the resolution chosen in the generation process cannot easily be changed without loss of quality (or alternatively without a lot of manual labor). In other respects such a picture is like a free-hand drawing: there is generally no desire to integrate the drawing with the text or to worry about conformity of typefaces—the drawing is treated as a “black box”.
- Photographs can be scanned like hand-drawn pictures, which converts the continuous tone of the photograph to a distinct range of colors or gray levels (black and white

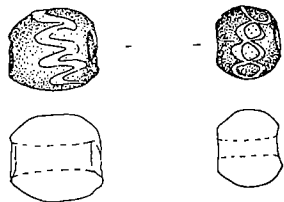


Figure 1.1: Pen and ink drawing of a bead.

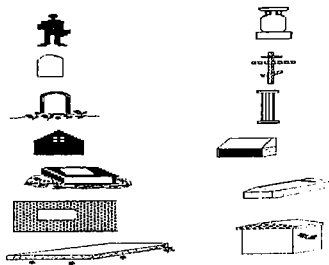


Figure 1.2: Bitmap drawing output created with MacPaint.

photographs treated in this way are known as half-tones). Full color reproduction requires sophisticated printing techniques, but this issue arises at the printing stage and does not normally affect the typesetting. Although the image may be enhanced or distorted (see Fig. 1.3), it remains purely a black box to T_EX.

- “Object-oriented” device-independent art graphics are in a different class. The important aspect here is that the graphic is stored in form of abstract objects that incorporate no device-dependent information (unlike bitmap graphics, where the storage format just contains information about whether a certain spot is black or white, making them resolution-dependent). This device independence makes it easy to reuse the graphic with different output devices and also allows us to manipulate individual aspects of the graphic during the design process.

In Fig. 1.4, created with Adobe Illustrator, the ducks were created by drawing one object in terms of curves and then copying and rotating it many times. This type of drawing often has quite a close relationship with the text, and may have textual annotations comparable to typeset text. Although it is usually possible to add text to the graphic with external tools like Illustrator, it is not in general possible to use L^AT_EX to typeset this text (see, however, Section 11.7 for a solution in some circumstances).

While external graphics languages do not allow such a close relationship, other languages built on top of L^AT_EX do provide this important functionality. L^AT_EX itself is already equipped with a simple and fairly low-level object-oriented graphic language—the picture mode. Although not always easy to use, it has allowed generations of L^AT_EX users to produce diagrams of surprisingly complexity. Several high-level special-purpose languages have been developed on top of this interface ranging from chemical formulae to chess typesetting.

Other general-purpose languages, not built on L^AT_EX’s picture mode, have been designed and implemented in the past decade; two very important ones, PSTricks and Xy-pic, are presented in detail in Chapters 4 and 5. Based on different paradigms, they differ greatly in approach, focus, and user interface and so have both found their place



Figure 1.3: Digitally transformed image (vertically stretched).

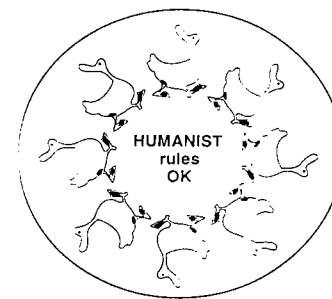


Figure 1.4: Object-oriented drawing.

in the L^AT_EX world. Later chapters focus on smaller specialized languages tailored for specific application domains such as physics and chemistry diagrams (Chapter 6), music (Chapter 7), and games (Chapter 8).

It is particularly important to use L^AT_EX to typeset the text within the graphic when the text includes formulae, as is often the case with the next two types of graphics:

- Algorithmic display graphics (histograms, graphs, etc.) are drawings created without human interaction but often contain text that it would be desirable to match with the document text. The scaling and distance between elements is an integral part of the drawing.

Extensive plotting and diagram facilities are provided by many L^AT_EX packages building on the picture mode, by generic T_EX packages like P_IC_TE_X (Wichura, 1988) and dratex (Gurari, 1994), and by PSTricks, described in Chapter 4. All these solutions let us deploy the full power of L^AT_EX’s typesetting functions within textual parts of the graphic and thus integrate it perfectly with surrounding document elements.

- Finally we have algorithmic structural graphics, which can be derived from a textual representation; unlike the previous category, however, often merely the spatial relationship between elements is important, not their exact position or size. Category diagrams, trees, and flowcharts are examples of this type of picture. Such graphics are natural candidates for generation by graphics languages internal to L^AT_EX that provide high-level interfaces focusing on objects and relationships while deciding final placement and layout automatically.

Of the general-purpose languages, the Xy-pic system (described in Chapter 5) is perhaps the most flexible one for this type of graphics, though P_IC_TE_X, PSTricks, and dratex are also suitable. For special applications such as tree drawing, many other L^AT_EX languages are available as well (see Brüggemann-Klein and Wood (1989), for instance).

As we see, there are many different types of graphics, each having different requirements. The first three types essentially present themselves as black boxes to L^AT_EX and thus their use within a L^AT_EX document involves no more than their inclusion and in some cases their manipulation as a whole. The necessary functionality is discussed in detail in Chapter 2 but, as we see later in this chapter, T_EX also offers facilities for including graphics that do not make use of the interface described there and are of interest in certain situations.

The latter three types are by far the common type of graphics in scientific texts, with uses ranging from maps (Taupin, 1993b) to chemical structures and commutative diagrams. They have in common the object-oriented data format, that is, they are described at least in part in an abstract way by specifying objects and their relations using a suitable language. In most cases, close integration with the surrounding text is desirable and can be achieved by choosing one of the graphics languages presented throughout this book.

In some cases interactive drawing programs can be instructed to output their results in one of the graphics languages built directly on top of L^AT_EX's picture mode. Georg Horn's TeXcad, distributed with the popular emT_EX implementation for MSDOS and OS/2, is probably the best known; the widely used Unix xfig can also translate to an internal graphic language. In this way such pictures, although externally produced, can also be influenced by layout decisions within the document. While such mechanically produced L^AT_EX code is normally not suitable for further manual editing and modification and thus manipulation is limited to layout facilities implemented by the chosen graphics language, the approach nevertheless can offer the best of two worlds in certain situations.

1.3 T_EX's interfaces

To understand the merits of the different approaches to graphics as implemented by various packages, it is helpful to consider yet another point of view: the interfaces provided by T_EX for dealing with them. Describing the methods by which graphics can be generated, included, or manipulated will give you some feeling for such important issues as portability, quality, and resource requirements of individual solutions. We assume that the reader has a reasonable understanding of how T_EX works, i.e., the progression from source file to a dvi file that is processed by a driver to produce printed pages.

We follow the classification of graphics as used so far: that is, we start by looking at ways of including externally generated graphics, i.e., those that appear as black boxes to T_EX, and then touch on the facilities offered for manipulating them. Finally we consider the interfaces provided to build graphics languages within T_EX, i.e., those that can form a basis for the last three types in the classification above.

1.3.1 Methods of integration

T_EX offers two major facilities for integrating graphics as a whole, one involving the `\special` command, the other using the font interface. The latter can be subdivided into two methods: first, representing the whole graphic as a single character of a special font

(or a number of such characters if the graphic is large); second, producing the graphic by decomposing it with a preprocessor and using fonts containing small pixel patterns to represent gray values. All three methods have been used in practice, the first two being the most common solutions.

Using `\special` commands

In reading *The T_EXbook*, the reference source for the T_EX language (Knuth, 1986a), we find no directly usable ways to include externally generated graphics. The only command available is the `\special` command, which by itself does nothing, but does enable us to access capabilities that might be present in the dvi driver program, or, to quote Knuth:

The `\special` command enables you to make use of special equipment that might be available to you, e.g. for printing books in glorious T_EXnicolor.

Saying it differently, Knuth saw that there might be a need to enrich the T_EX language but was reluctant to provide primitives for further graphical operators and data structures or, in case of the inclusion of external graphics, a well-defined interface.

Thus `\special` allows us to access *special* features of a driver program that translates the dvi output of T_EX into a language understood by the output device, and if this driver has mechanisms to include external graphics then we can import such graphics. The price we pay is non-portability, since our source will contain calls to a non-standard interface—and in fact authors of different drivers have implemented different conventions so that documents that contain direct calls to `\special` only work with a very restricted setup.

Therefore in 1993 Leslie Lamport, Frank Mittelbach, and Chris Rowley designed a high-level interface for L^AT_EX that abstracts from the underlying low-level syntax understood by the individual drivers. This interface was implemented by David Carlisle and Sebastian Rahtz in the graphics package (Carlisle, 1995) for the new L^AT_EX 2_ε standard, which became officially available in June, 1994 and is discussed in detail in Chapter 2. In a similar manner the use of `\special` commands to address the color capabilities of some drivers was made transparent with the high-level interface provided by the color package described in Chapter 9.

By offering a set of high-level commands, the dependency on the idiosyncrasies of the driver used is effectively eliminated from the document. The only place where one has to change a document using such commands is a single line in the preamble that controls how the commands are implemented (by loading a driver-specific control file).

While the development of these interfaces has lessened the need for a fully standardized syntax for using the `\special` command, such standardization is still desirable because it would simplify both the implementation of existing interfaces and the development of further high-level interfaces for special applications.

Using fonts

Another way to include external graphics results from the observation that there is an implicit standardized interface for external graphic objects in T_EX—its system of accessing

external fonts. A font is described to T_EX by its external name and by a `.tfm` file that contains information about the glyphs in the font necessary to determine their placement. And that's it—the actual shape of the character is irrelevant as far as T_EX is concerned. Everything else, i.e., the actual printing of the glyph, is again the task of the driver program, but this time a format (the `.pk` or `.ppl` files) exists that is implemented by all modern drivers and can serve as an interchange format for graphic objects to a certain extent. If the graphic can be transformed into such a font format, we can include it by loading this “font” with standard L^AT_EX commands, selecting it and asking for the character representing our graphic. This technique is effective and portable (such fonts can be used with different implementations of T_EX and, almost always, with different output devices). The major drawbacks are the fact that scaling cannot easily be performed without going back to the original artwork and reproducing a font at a different resolution, and the fact that T_EX has a limit on the number of fonts it can load.

Programs exist to translate Microsoft Paint files to `.ppl` format (`bitppl` by Gustav Neumann) and `.pbm` format¹ to `.pk` format (`pbmtoptk` by Angus Duggan). Both programs work by taking one or more existing pictures in bitmap form and converting them to font format, assigning successive pictures to letters of the alphabet. For example, Fig. 1.5, a cartoon by Duane Bibby, was included into this book by scanning it and producing a graphic file in GEM picture format, which was then converted to `.pbm` format. From that a font called `lion` was generated using `pbmtoptk` containing a single character “A” representing the graphic. From that point on the graphic can be included in T_EX with a declaration like `\font\lion=lion` to load the font followed by `{\lion A}` at the point where it should appear.



Figure 1.6: Color gif file converted to T_EX font files by `bm2font`.

The most sophisticated of these programs is Friedhelm Sowa's `bm2font`, which accepts graphics in `.pcx` or `.gif` format and produces one or more `.pk` and their corresponding `.tfm` files. Many drivers are unable to handle large characters, so the program automatically splits larger graphics and represents the fragments as individual characters. To keep this process transparent to the user, a `.tex` file is written that places the fragments together again.

Half-toning can be undertaken and even color separations can be made (Sowa, 1994). The drawback of this approach is the fact that many new fonts are created quickly, using up T_EX's capacity. For this reason it can be used only with documents containing a small number of graphics. Appendix A.3 gives an overview of the command line options available with the `bm2font` program.

¹The `.pbm` (portable bitmap) format is an interesting intermediate graphic format for transforming one format into another, since programs exist that convert many graphic formats to and from `.pbm` format. The `.pbm` suite, mostly written by Jef Poskanzer, is in the public domain. This format, with many others, is explained in Murray and van Ryper (1996).



Figure 1.5: Scanned cartoon converted to font via `.pbm` format.

Using half-tones

We can also include a half-tone drawing by making up a font that consists of grey-level blocks and combining them together in the normal T_EX way. Both Donald Knuth (1987) and Adrian Clark (1987) have demonstrated this technique. The drawing can be scaled, but it is not easy to put text within the picture boundary, and one is dependent on the original resolution. Clark's fonts and preprocessor programs can be found in the T_EX archives.

Half-toning can also be achieved directly in PostScript, a rather less cumbersome method than T_EX half-toning, albeit at the cost of device-independence and flexibility.

1.3.2 Methods of manipulation

The facilities in T_EX for manipulating graphics (e.g., scaling or rotating) included by one of the above methods appear at first glance to be relatively poor. If the graphic is included as one or more font characters, then scaling is in principle possible—at least from T_EX's point of view there is no problem since fonts can be loaded at any size. But at the printing stage the driver program then probably complains that it cannot find the font at the appropriate size, which means that one has to regenerate it at the size requested, a task that is (at the least) time consuming.

The only alternative is to resort again to the `\special` capabilities of the driver used. With some drivers, especially those that translate the `.dvi` files to PostScript, this method offers a large variety of possibilities including scaling, rotation, etc. However, until recently this approach had the drawback of different syntaxes for the argument of the `\special` command, a situation that was only resolved with the introduction of the graphics package interface discussed above.

The situation is slightly different when the half-tone approach is chosen, as then the fonts in question are METAFONT fonts and can be generated automatically at the requested size by a modern driver, assuming that their sources are available.

1.3.3 T_EX's graphic hooks

Graphics are not always around in some form just waiting to be included—often we have to produce them in the first place. In the following sections we explore the facilities available in the T_EX world for generating graphics.

Using T_EX's built-in commands

As already observed, T_EX does not provide a rich set of graphic primitives. What we find are built-in functions that let us to draw horizontal or vertical lines of arbitrary thickness—even sloped lines are lacking. In addition, T_EX allows us to position objects with high accuracy anywhere on the page. The only other functions offered are primitives for placing objects in matrix structures where the exact placement is determined automatically by the size of the objects (e.g., the `tabular` and `array` environments in L^AT_EX). But even with this minimal set it is possible to define powerful graphics languages, especially if we consider the

character “.” as a building block, since arbitrary lines and curves can be drawn by placing hundreds of tiny dots next to one another.

Using fonts

Instead of using a “.” character as an individual graphic object, Leslie Lamport provided L^AT_EX with a basic set of picture-drawing macros that use special fonts containing line segments at various angles and circle and curve fragments at various sizes. These macros have allowed programmers to produce surprisingly sophisticated output, beyond the quite respectable uses to which the `picture` environment of basic L^AT_EX can be put. Other languages based on T_EX typesetting of small fragments of drawing have also been devised (see e.g., Ramek (1990) and Waldschmidt (1988)), but the standard of L^AT_EX picture mode and its fonts is now well established. The Xy-pic package (see Chapter 5) follows a similar approach with its own set of fonts for arrowhead styles.

The approach of using fonts is not limited to working with technical drawing fragment fonts created with METAFONT; music, chess, and Go fonts are available, that allow you to typeset musical scores, chessboards and Go boards using T_EX as a layout engine; these are described in Chapters 7 and 8.

Using `\special` commands

We have already seen that the `\special` command can be used to access driver capabilities for graphics inclusion if provided. Clearly such an extension can also offer extended drawing capabilities if the driver provides such constructs.

To understand how the `\special` command works, you can think of it as producing an invisible space annotated in some way with the text of its argument. Depending on where in the source the `\special` command was encountered, this “invisible space” appears somewhere on the typeset page, just as a word in the source finally appears somewhere in a paragraph. This means that after typesetting a document with T_EX each `\special` command is associated with a position on a page.

When a driver encounters such an “invisible space” produced by a `\special`, it knows its position on the current page. It then reads the annotation (i.e., the argument of the `\special` command) and (if it understands it) carries out the action requested. Therefore, if the driver offers such capabilities, it is possible to denote points and regions for special treatment ranging from drawing a line between two points to rotating an area after typesetting by T_EX. This is the mechanism by which some of the more complicated graphic functions of some packages described in this book are implemented. Back in 1982 Knuth wrote about using the `\special` command and conventions for the syntax within its argument:

[...] the author anticipates that certain standards for common graphic operation will emerge in the T_EX user community, after careful experiments have been made by different groups of people; then there will be a chance for some uniformity in the use of the `\special` extensions.

Unfortunately, such a standard has not emerged despite various efforts (cf. Vanderburg and

Reid (1987), a discussion of the problem in Rogers (1989), and Reid and Hosek (1989)). For graphics inclusion we now have a high-level interface that abstracts from the underlying driver facilities. However, the situation with respect to other drawing capabilities of driver programs via the `\special` command is less satisfactory, as there exists neither a standard nor a high-level interface that hides the capabilities of different drivers. On the other hand, as these capabilities depend so strongly on the target language of the driver, it may be that useful standardization cannot be achieved. Given the dominance of PostScript in the printer and typesetting market, standardization of `\special` primitives for PostScript drivers is probably the most useful thing to do. In effect, this is the route taken by PSTricks, described in detail in Chapter 4.

There are only two sets of conventions for the argument of the `\special` command supported on more than one platform. The first is the “`tpic`” set, defined originally by Tim Morgan to implement the “`pic`” language for use with T_EX and implemented by a variety of drivers, including `xdvi`, `dvips`, `dviwin`, and `dvipsone`. The second is the `emTEX` set, initially supported by all the screen and printer drivers in Eberhard Mattes’ MSDOS and OS/2 T_EX implementation; other drivers, including `dvips`, have also implemented them. While not as powerful as the “`tpic`” set, the `emTEX` set can nowadays be used with many types of devices and should be regarded as a minimal set.

Both the “`tpic`” and the `emTEX` set have been used to implement back ends for object-oriented graphics languages built on T_EX. Their interfaces are described in Appendixes A.1 and A.2.

1.4 Graphics languages

There are a number of distinct ways of producing graphics each with advantages and disadvantages in terms of ease of generation, flexibility, device independence, and ability to include arbitrary T_EX text. As discussed in Section 1.3.2, it is possible to manipulate a graphic object as a whole using various standard operations, e.g., scaling, rotation, etc. However, to manipulate individual parts of a graphic these parts need to be addressable in a suitable manner—i.e., the source of the graphic must be in some abstract graphic language.

The remainder of this chapter contains a “roadshow” of graphics languages explaining how they make use of the interfaces provided by T_EX or, in case of external graphics languages, how they can interact with the L^AT_EX system to produce impressive documents. Several of these languages are discussed in detail in later chapters; the others are included to provide a first-level introduction.

1.4.1 T_EX-based graphics languages

Several graphic languages have been developed over the years that use the hooks provided by T_EX. They differ in their approach and focus and thus have a wide range in portability, resource usage and flexibility.

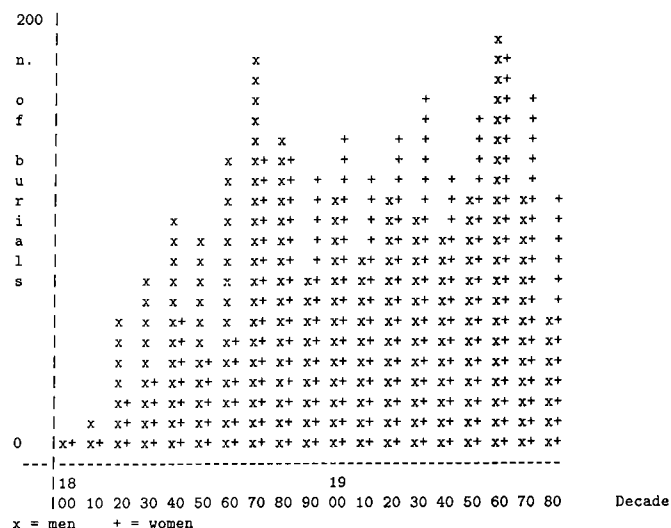


Figure 1.7: Using ASCII symbols to make a histogram.

Character-based diagrams and pictures

Pictures built from fonts with fixed-width characters can be produced on any machine with nearly every editor, and can easily be incorporated into L^AT_EX using the `verbatim` environment. They have the disadvantage of being crude, very limited, and cumbersome to generate, and are normally of fixed size, though T_EX's normal font sizing can be used to "scale" them (as done in Fig. 1.7 using `\footnotesize`). This histogram illustrates the number of burials for men and women per decade in the Protestant Cemetery, Rome (showing that the number of women has gradually overtaken that of men—see Rahtz (1987)), but it was impossible to annotate the diagram sensibly. It is not easy to include text in variable-wide fonts; this can be achieved by using the `alltt` environment, but at the cost of losing `\`, `{`, and `}` as directly accessible drawing characters. See *The L^AT_EX Companion* (Goossens et al., 1994, p. 66) for a discussion of this and similar environments.

A similar idea was used ingeniously in the `sprite` package written by Martin Costabel to generate new characters in T_EX by providing a bit-map of a character and scaling it to size; thus the code in Fig. 1.8 generates a hollow five-pointed star 0.3cm square, which we can use to typeset an EEC-type logo (Fig. 1.9). Each of the Bs is set as a tiny rule whose size is determined by the "resolution" of the picture, and the intended height and width of the character are defined by the arguments to the `\sprite` command.

It is clear that the star generated is fairly crude; if we specify its size so as to create a 3cm square, it looks truly awful (Fig. 1.10). The technique is limited, as it takes time to create the

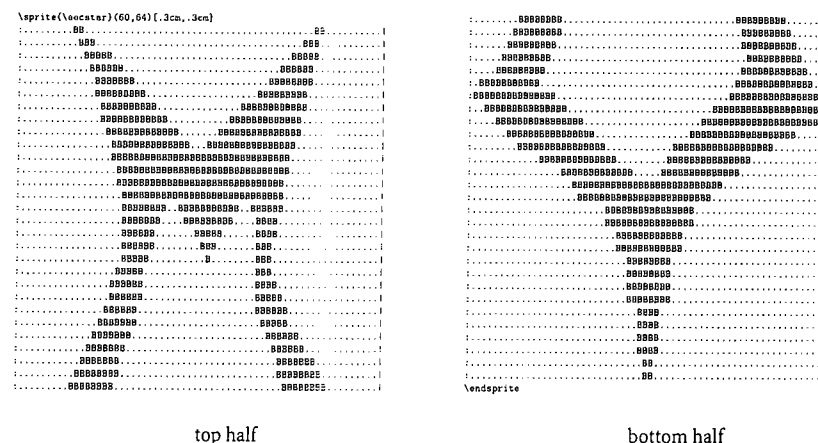


Figure 1.8: Bitmap input to generate the symbol in Fig. 1.9.

initial "bitmap" and it is very difficult to make smooth curves. The computation and storage overhead in T_EX are also considerable.

A front-end interface to the `sprite` package is the `bit2spr` program by Marc Rovner, which converts existing X11 bitmaps into `sprite` format. Since L^AT_EX needs to know the dimensions of the sprite, `bit2spr` takes the height and width of the bitmap and uses ratios to calculate the height and width of the sprite in inches. The default ratios are 1.5, but these can be overridden on the command line with `-width` and `-height` options. Ratios can be any rational number, but extremes may well make T_EX run out of memory. A simple example is shown in Fig. 1.11: the `dvi` icon designed by Donald Knuth (1993).

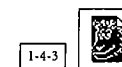
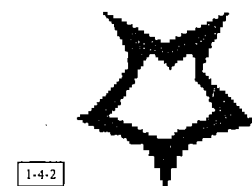
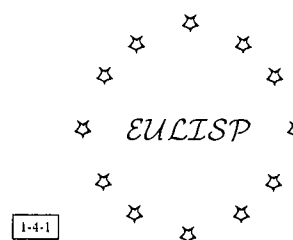


Figure 1.9: A "sprite" logo.

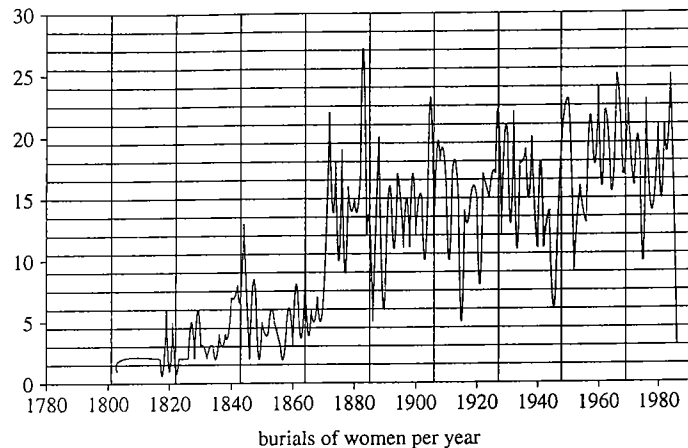
Figure 1.10: An expanded "sprite".

Figure 1.11: Output of `bit2spr`.


```

\usepackage {rawfonts,pictex}
\beginpicture
\normalgraphs
\longticklength=3pt
\setcoordinatesystem units <.02in,.08in>
\setplotarea x from 1780 to 1990, y from 0 to 30
\axis bottom label {burials of women per year} ticks
    numbered from 1780 to 1980 by 20 /
\axis left ticks numbered from 0 to 30 by 5 /
\grid 10 20 \setquadratic \plot "year.wom"
\endpicture

```

Figure 1.12: P_IC_TE_X source code and graph output.

P_IC_TE_X

A hybrid approach to picture drawing without any new fonts is adopted by Michael Wichura's brilliant P_IC_TE_X (Wichura, 1987, 1988), which implements a complete plotting language mainly by setting a myriad of dots taken from a standard T_EX font. Its main drawback is the horrific computation and memory overhead which make it necessary to run a very large T_EX (pictures can take several minutes to process on smaller machines, and it is not at all easy to use P_IC_TE_X with other packages without running out of T_EX "dimension" variables²). It has the advantage of scalability and ease of inserting plain T_EX text, and the complexity of the drawing language is more a design feature than a necessary concomitant to

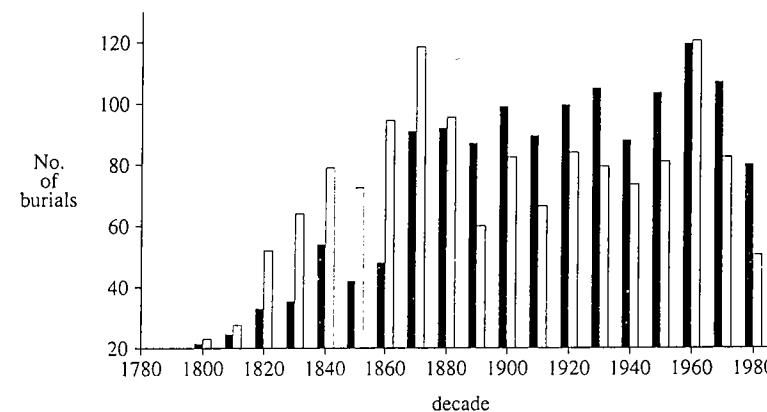
²It is possible to overcome the latter problem by redefining some fairly low-level macros, but this is not the sort of approach most users want to take.

1-4-4

```

\usepackage {rawfonts,pictex}
\beginpicture
\normalgraphs
\setcoordinatesystem units <.02in,.01in>
\setplotarea x from 1780 to 1990, y from 0 to 220
\longticklength=3pt
\axis bottom label {decade} ticks
    numbered from 1780 to 1980 by 20 /
\axis left
label {\stack {No.,of,burials}}
ticks withvalues 20 40 60 80 100 120 140 160 180 200 /
from 0 to 200 by 40 /
\setbars <-2pt,0pt> breadth <0pt> baseline at y = 0
\linethickness=4pt
\plot "decade.wom"
\setbars < 2pt, 0pt> breadth <4pt> baseline at y = 0
\linethickness=.25pt
\plot "decade.men"
\endpicture

```

Figure 1.13: P_IC_TE_X source code and histogram output.

1-4-5

the approach. Figures 1.12 and 1.13 use the data first seen in Fig. 1.7, and show the authors' input above the picture.

The advantages of a system like P_IC_TE_X are illustrated by Fig. 1.14, which shows a drawn and shaded histogram; the captions pass through normal T_EX, and so can include maths formulae, and are rotated using the L^AT_EX standard graphics package, giving the user considerable power to improve the appearance while retaining compatibility with T_EX fonts.

```

\usepackage {graphics,rawfonts,pictex}
\beginpicture
\setcoordinatesystem units <10pt,.03pt>
\setplotarea x from -1 to 24, y from 0 to 4500
\axis left label
\rotatebox{90}{Number of Coins (\$surd\$)} ticks
numbered from 0 to 4500 by 500 /
\shadirectangleson
\setbars breadth <15pt> baseline at y = 0
\baselabels ([t1] <2.5pt,-5pt>)
\plot "coins.dat"
\endpicture

```

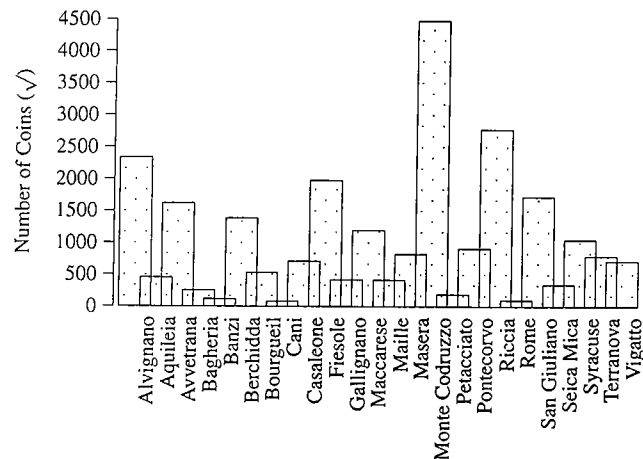


Figure 1.14: Histogram drawn with P_{CT}E_X, including rotation of text (graph courtesy of Kris Lockyear)

P_{CT}E_X was originally developed for use with the plainT_EX format and makes use of one font command (`\fivern`) that is no longer defined in L^AT_EX. Most installations provide a file `pictex.sty` that makes P_{CT}E_X available as a L^AT_EX package; if not, the relevant lines are:

```

\DeclareFixedFont{\fivern}{OT1}{cmr}{m}{n}{5pt}
\input{prepictex}
\input{pictex}          % main code and some documentation
\input{postpictex}
\endinput

```

The dratex and aldratex packages

A recent addition to the T_EX-based graphic languages is a drawing package developed by Eitan M. Gurari (1994) that allows one to draw most common diagrams in a convenient way. It has two levels, one for basic drawing commands (in the `dratex` package) and the other for higher-level constructs (in the `aldratex` package). They are implemented largely using T_EX primitives, but do use the L^AT_EX circle fonts on occasion. The two packages are described in detail in Gurari (1994); a simple example is the piechart in Fig. 1.15.

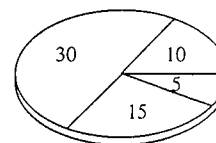
Building on the picture environment

L^AT_EX's `picture` mode provides a basic graphics language that is fully portable among different installations. Although not always easy to use, it has allowed generations of L^AT_EX users to produce diagrams of surprising complexity.

Various extensions to the L^AT_EX picture macros exist; the most widely used is Sunil Poudar's `epic` (1986), whose commands enhance the graphic capabilities of L^AT_EX and provide a friendlier and more powerful user interface by reducing calculations needed to specify the layout of objects. The `epic` package, together with Joachim Bleser's bar macros for drawing bar charts and the `curves` extension to the L^AT_EX picture mode, is described in Goossens et al. (1994).

The disadvantages of the limited facilities of the L^AT_EX picture fonts are clear, even when the enhancements in `epic` are used. Circle sizes and line angles are in a fixed range or appear jagged (because they are built from small line segments), and there is no facility for shading or coloring areas. On the positive side, the method is very portable and is well integrated with the rest of the text.

One important addition to L^AT_EX's picture mode has been to redefine the picture macros to be device-dependent by mapping them directly to PostScript `\specials`. This frees the user from the limited range of circle sizes and line angles. Early versions are described in Clark (1989) and Arbortext Inc. (1986), but in the future the functionality will be provided with the `pict2e` package as part of the standard L^AT_EX distribution. At present David Carlisle's `pspicture` already provides some of these functions.



```

\usepackage {dratex,aldratex}
\Draw
\Scale(1,0.6)
\PieChartSpec(1,50,25)()
\PieChart(10 & 30 & 15 & 5)
\Move(0,-6)
\DrawOvalArc(50,50)(180,360)
\EndDraw

```

Figure 1.15: Piechart produced with `aldratex`.

ande		andn		ands		andw	
capew		capns		exore		exorn	
exors		exorw		nande		nandn	
nands		nandw		nexore		nexorn	
nexors		nexorw		nfete		nfetn	
nfets		nfetw		nore		norm	
nors		norw		note		notn	
nots		notw		ore		orn	
ors		orw		ote		otn	
ots		otw		pfete		pfetn	
pfets		pfetw		resew		resns	
tgateew		tgateens		tje		tjn	
tjs		tjw		vddpin		vsspin	

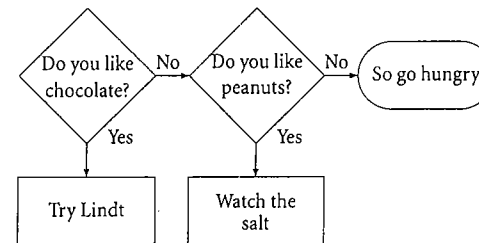
Figure 1.16: The *lcircuit* library of symbols.

For the epic package a reimplementation is also available with Conrad Kwok's *eeipic* package, using the `\tpic \specials`, which are understood by several drivers (see Section 1.3.3 and Appendixes A.1 and A.2.)

There are various approaches to building higher-level interfaces to the picture mode. The first approach is typified by Adrian Johnstone's *lcircuit*, a library of circuit schematic symbols for L^AT_EX picture mode. The set includes all basic logic gates in four orientations, FETs, power supply pins, transmission gates, capacitors, resistors, and wiring T-junctions (see Fig. 1.16). Each symbol is defined with a `\savebox` command and stored in an external file, thus making it possible to reuse the symbols as subpictures with ease. No attempt is made, however, to provide higher-level functionality, i.e., combining these symbols in a larger picture must be done manually or with a program like *TeXcad* distributed with the *emTeX* system.

The second and by far the most common approach is to define a little language entirely in L^AT_EX; Johannes Braams' *nassflow* is an example of this, a macro package for flow charts or Nassi-Schneidermann diagrams, implemented using the `picture` environment. Several examples of such an approach appear in later chapters.

A third approach is to define an specialized external language together with a preprocessor program that converts input in that language to T_EX primitives; the most sophisticated is perhaps the *gpic* program described below, but there are many smaller examples, e.g., the flow program by Terry Brown that translates simple-flow chart description files into commands of the L^AT_EX `picture` environment. The language allows variously framed boxes joined together with arrows, with a choice of direction. Multiple directions are supported by setting "tags" on a stack and reverting to them. An example is shown in Fig. 1.17.



```

Right
Choice . . No Yes
  Do you like
    chocolate?
Tag
Choice . . No Yes
  Do you like
    peanuts?
Tag
Right
Oval
  So go hungry
ToTag
Down
Box
  Try Lindt
ToTag
Down
Box
  Watch the
    salt
  
```

Figure 1.17: Example of flow language.

Xy-pic

Xy-pic is a language for typesetting graphs and diagrams with T_EX originally developed by Kristoffer H. Rose, with the help of Ross Moore. The system in its default setup uses standard T_EX and METAFONT, i.e., it makes use of the font approach by supplying its own arrow fonts, etc., but output for a specific driver (like *dvips*) can be generated. It can typeset complicated diagrams in several application areas, including category theory, automata theory, algebra, neural networks, and database theory. The system is built around an object-oriented *kernel drawing language*. Each "object" in a picture has a "method" describing how it should be typeset, stretch, etc. A set of enhancements to the kernel have also been built, called *extensions*. Xy-pic is reviewed in detail in Chapter 5.

PSTricks

PSTricks takes advantage of the extremely powerful PostScript page-description language used to drive many printers and typesetters. It uses `\special` commands to embed fragments of PostScript in the T_EX source, which are passed on by conforming drivers (including *dvips*, *dvipsone* and *Textures*). This allows effects ranging from line drawing through color, shading, and character transformation (such as setting text on a curve) and right up

to pseudo-3D pictures. PSTricks packages most of the power of PostScript in the familiar T_EX syntax, and the low-level functionality is extended with a range of high-level packages for, e.g., trees and graphs. PSTricks is reviewed in detail in Chapter 4.

1.4.2 External graphics languages and drawing programs

There are many commonly used graphics languages and interactive packages, ranging from the very low-level to the extremely specialized, and translators have been written to convert some of them to forms T_EX can handle, whether a L^AT_EX picture, commonly used `\special` sets, or even METAFONT. We look at a selection of them here, but our catalogue is by no means exhaustive. Many of the widely used drawing or drafting packages come with a range of output formats, some of which can be converted to T_EX-compatible forms, so the possibilities are numerous. Free-standing graphics conversion programs (like Hijaak, Image Alchemy or Adobe File Utilities) deal with yet a different range of conversions (largely but not exclusively to do with bitmaps).

HPGL

Very many general-purpose drawing and plotting packages support output in HPGL (Hewlett Package Graphics Language), and this can be translated to METAFONT using Dirk Donath's `hptomf`, or to a wide variety of T_EX-compatible formats using Heinz W. Wernthes' `hp2xx`. This converts to METAFONT, T_EXcad, `epic`, EPS, "em" (emT_EX `\special` commands), HPGL vector formats, and PCX, PIC, IMG, and PBM bitmap formats, with good control over features like scaling, pen sizes and colors. Of the T_EX-specific options, the "epic", "em", and T_EXcad conversions cannot be used in many cases because of limitations of T_EX memory or package capabilities; the METAFONT option can also generate output that strains the memory of METAFONT or the `gftopk` program (the example in Fig. 1.18 needed a larger version built for it), but can produce good results. Figure 1.18 was generated by converting plotter output from AutoCAD into METAFONT; the detailed contour lines on this plan of the Comoran island of Moheli consist of about 10,000 line segments.

HPGL is, however, a fairly low-level language, and after translation to another format is unlikely to be easily edited or manipulated. It is also a language of some age and its capabilities may be said to be restricted to remain compatible with the printing and plotting hardware of earlier times.

gnuplot — a plotting package

gnuplot (by Thomas Williams, Colin Kelley and others) is a general-purpose 2D (and 3D) plotting program that has its own language for describing graphs and plots. It comes with drivers for many output devices, including pic, L^AT_EX (using emT_EX `\special` commands), METAFONT, PostScript, PSTricks, T_EXdraw and `eeepic`. A typical graph, again using the data introduced with Fig. 1.7, is presented in Fig. 1.19, with the simple gnuplot commands that generated it shown above.



Figure 1.18: AutoCAD plotter output converted to METAFONT.

```
set terminal latex
set xlabel "Protestant Cemetery decades"
set ylabel "Number \\of \\burials"
plot 'decade.wom' with lines, 'decade.men' with linespoints
```

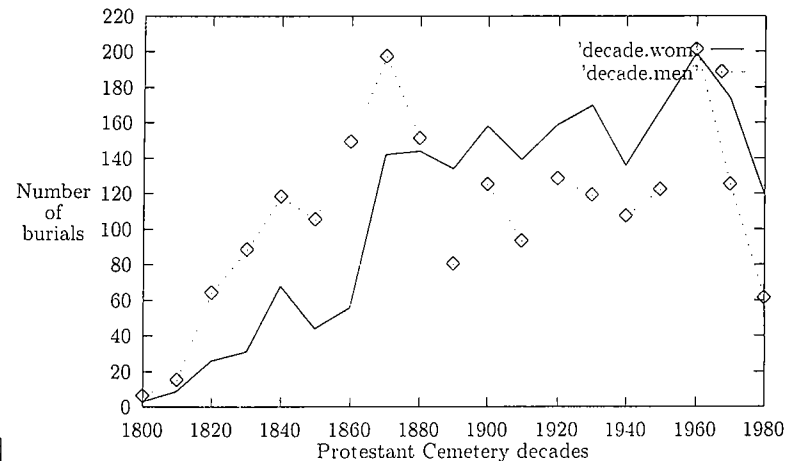


Figure 1.19: Graph generated by gnuplot using L^AT_EX picture commands.

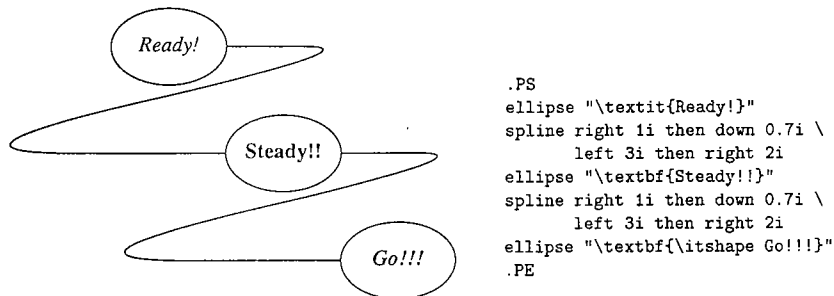
The pic language

Those used to typesetting in traditional Unix are familiar with the `troff` program. For this program Brian Kernighan designed a low-level picture language and developed a preprocessor (`pic`) to translate files written in that language into statements understood by `troff`. The program `tpic` is a version of the `pic` preprocessor that performs the same job on `pic` statements embedded in a T_EX document, but replaces them with `\special` commands, that request graphics primitives from the dvi-driver. The `tpic-\special` syntax is described in Appendix A.2 on p. 464.

`troff` and some of its preprocessors have been reimplemented by James Clark as part of work for the Free Software Foundation. Within this suite of programs, `gpics` implements the functionality of the original `pic` processor, but unlike `pic`, `gpics` can also be instructed to output T_EX code containing `\special` commands with `tpic` syntax by calling it with the option `-t` or `-c`.

The `gpics` program reads a T_EX file and leaves all lines unchanged until it encounters a line containing only the statement `.PS`. From that point on it assumes that statements in the `pic` language follow, and translates them until it finds a line containing only `.PE` statement.

The `pic` language defines basic graphics objects like `line`, `box`, `arc`, `arrow`, `ellipse`, `circle`, `spline`, and text strings; named blocks can be created for a higher-level construct than the simpler objects; looping, conditionals, variables, and elementary functions (like `sin`, `cos`, `atan`, `sqrt` and `rand`) are part of the language. The position of an object can be given by absolute location or relative to previously specified objects or locations. A current location and drawing direction are defined at all times. To demonstrate the effect, consider the following lines of code:



The ellipses are drawn at the current default position, but the lines joining them are given an explicit direction and size; the text can contain arbitrary L^AT_EX code. Further examples of `pic` code are given in Figs. 1.20 and 1.21.

The `pic` language has an intuitive way of expressing simple graphic relationships and in this respect is superior to using, say, the `picture` environment. However, the problems with the `gpics` approach are threefold. First, it requires a preprocessing stage, i.e., a separate compiled program for the computer platform where it is used; second, it uses the `tpic-\special`s, which not all drivers support; and third, the preprocessor knows nothing about

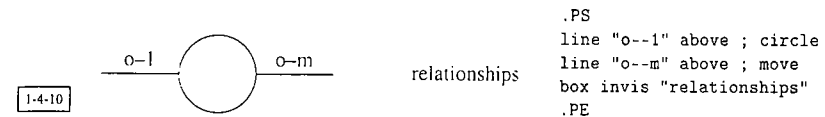


Figure 1.20: Simple example of `pic` code.

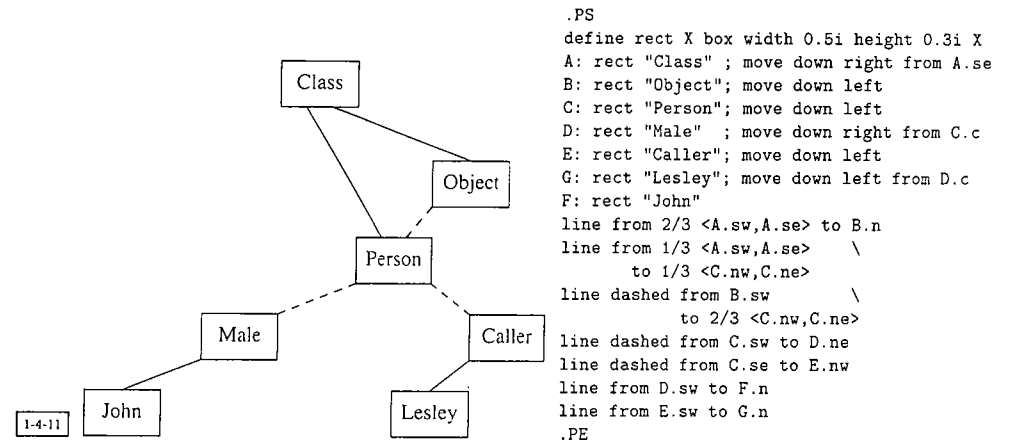


Figure 1.21: Example of `pic` code showing use of stored names.

how text is typeset and so cannot, for example, fit boxes accurately around formulae. With a sufficiently powerful set of common `\special`s, then a `pic`-like language can be implemented within T_EX, so that different devices could interpret the commands as best they can. This was attempted by Rolf Olejniczak-Burkert with `texpic` (1989), which provides a great deal of the functionality of `pic`. The advantages of this system are considerable, but the language is not yet as complete as `pic` and the input syntax (looking like T_EX macros) does not look as "clean" as `pic`'s. But the closer integration with T_EX means that some things are done better—for instance, the size of a box containing text can be made exactly to reflect the width of the contents: this would be impossible in `pic`, which has no idea of font metrics.

Special-purpose external graphics languages

In some circumstances, we may consider adding yet another layer to our drawing system: we devise an entirely new language for our application, and write a translator to one of the

```

frame invis ht 3 wid 4 left solid bot solid
label bot "Male and female burials per decade ($n \approx 4300$)"
ticks left out from 50 to 400 by 50
ticks bot out from 1800 to 1986 by 20
copy "decade.dat" thru X
tot=$2+$3
line from $1,0 to $1,tot
line from $1,tot to $1+10,tot
line from $1+10,tot to $1+10,0
X

```

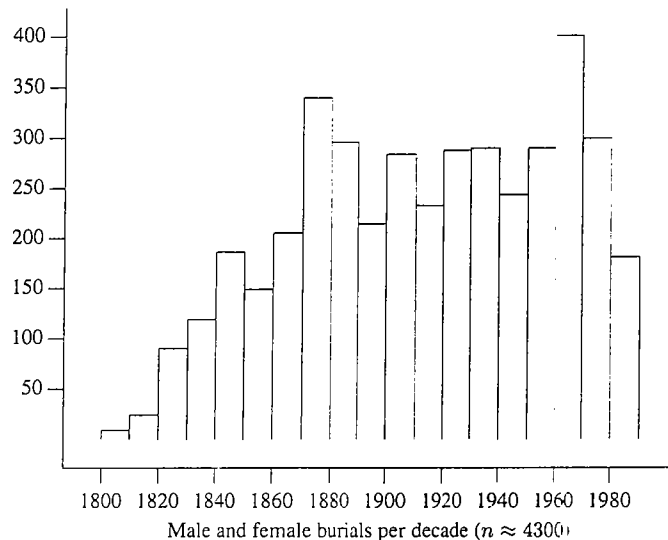


Figure 1.22: Example of grap output.

existing device-independent languages, and translate from there to T_EX. This approach of defining small, problem-oriented languages is discussed in Bentley (1986).

The original Unix pic (Kernighan, 1984) has a companion set of higher-level languages which generate pic output; there are ones for general graphs (grap, described in Bentley and Kernighan (1984)), chemical formulae (chem) and music (music). We took as an example some data for that Rome cemetery and used the grap language to create a reasonable histogram (similar to Fig. 1.7, summarizing burials per decade) very quickly. The input to grap, and the result, are shown in Fig. 1.22.

Although it is not the intention here to describe the grap language, the reader can observe some of main characteristics. Note that T_EX mathematical typesetting can be used (in

the label for the graph) and that the graph is scaleable (by the ht 3 wid 4 clause of the initial frame command—the units are inches).

METAFONT and METAPOST

T_EX has a companion font-creation program, METAFONT. While METAFONT was designed for producing beautiful character shapes in fonts, it can also be used very successfully to create drawings for T_EX. It has very powerful techniques and data structures well suited to many types of drawing, and has the advantages that it is available anywhere that T_EX is and that its output (pk fonts after conversion) is understood by practically all dvi-drivers. It has two disadvantages, but both have solutions:

1. The syntax of METAFONT is completely different from T_EX, and some users find it hard to learn; an ingenious “wrapper” is available in the mfpic L^AT_EX package, which lets you describe your picture in familiar L^AT_EX syntax and have it written out in METAFONT form to generate the picture;
2. The bitmap font output of METAFONT is awkward to handle in some drivers and requires regeneration for each different output device; it renders very badly in the popular Adobe Acrobat program. METAFONT also lacks support for some basic building blocks like color. A nice solution for many users is METAPOST, a reimplementaion by John Hobby of METAFONT to produce device-independent PostScript output. At the same time he added support for color and provided some high-level graph-drawing support.

METAFONT, mfpic, and METAPOST are explored in Chapter 3.

1.5 Choosing a package

Most people do not choose a graphics program or macro package on the basis of the method T_EX adopts to handle the output. The *type* of picture they want to produce, not the *way* it is done, is usually the important consideration. For many people, there is an excellent selection of commercial, interactive packages designed for their particular facilities in the relevant subject area (such as AutoCAD for archaeologists, ChemDraw for chemists, Adobe Illustrator for graphic artists, SPSS for statisticians, Mathematica for mathematicians, ArcInfo for geographers, xfig for computer scientists, Excel for business people, and so on). For others, however, creating graphics by describing them in a special notation seems far more natural.

If you *are* in a position to make a completely free choice about what package to use, you might consider the following points:

- The basic decision you make depends on the relationship between the picture and the text. If the picture is a “black box”, then generate it separately and include it at the printing stage. If its contents should have the same style as the text, then investigate drawing packages integrated with T_EX.

- How important is *total* portability in the T_EX world to you? If it is vital, use the packages that draw pictures using tiny dots or T_EX fonts. You could also choose the portability provided by common (but not standard) `\special` primitives, portability provided by PostScript, or portability using black-box bitmap graphics. The future use intended affects your choice of package.
- If you want to do “art” graphics, which need interactive drawing or painting, then choose a package that suits your subject area and that can drive the printer of your choice. If you have a T_EX driver for a printer, then it will probably allow you to automatically include files destined for the same printer. In practice, the most flexibility is offered by the huge array of software written for the Macintosh and Microsoft Windows, but it is sensible to choose one that can write encapsulated PostScript, because this is the most widely portable format for publishing.
- If you want to include PostScript pictures, use the standard graphics package discussed in Chapter 2, so that your documents are not dependent on the vagaries and syntax of a particular driver.
- If you have half-tones to reproduce, you can easily scan them and include them in T_EX. Scan them at the highest resolution you can afford (in terms of equipment and disk storage); but be sure that you understand the issues of scaling bitmaps—scanning at 1200 dpi and then printing at 1270 dpi can produce unpleasant results due to the tiny scaling that has to be performed.
- If you want to plot data in relatively simple ways, the choice of software will depend on your normal working environment. If your data is in a spreadsheet, then you probably have adequate facilities there. If you use a database, it might be easier to write retrieval programs that generate pictures in a plotting language like L^AT_EX `picture`, `pic` or P_CT_EX; the software is not likely to be very portable outside your own environment.
- If you are creating algorithmic pictures in which the *layout* is determined by your data rather than the *contents*, and if the output includes a lot of textual material (particularly mathematics or non-Roman scripts), then you need to look at the macro packages that implement drawing directly in T_EX, so that your text is processed by T_EX.

A drawback to including material like graphs as external pictures is that the drawing has no interaction with T_EX; we would like to be able to add arbitrary T_EX text (such as a mathematical label for a graph) in the same font as the rest of the text. This has been recognized as an issue for some time, and Karney (1988) suggested a scheme whereby the author would nominate captions, etc., for tagged points on a graph, pass these through T_EX as `\specials` and have the dvi driver integrate these typeset captions back into the graph being printed. The PSfrag package discussed in Section 11.7 on p. 460 offers an interesting alternative solution.

We try in this book to show you the good points of many programs and L^AT_EX macro packages, but we cannot tell you which one is right for your needs. We hope that the variety and quantity of pictures in the following chapters demonstrate that graphics in L^AT_EX is alive and well and can meet almost any need.

CHAPTER 2

The L^AT_EX 2_ε graphics bundle

L^AT_EX 2_ε provides a generalized driver-independent interface to include external graphic material and to scale and rotate L^AT_EX boxes; a generalized package for color is also available, as discussed in detail in Chapter 9.

We start by looking at graphics file inclusion; L^AT_EX offers both a simple interface (`graphics`), which can be combined with the separate rotation and scaling commands, and a more complicated interface (`graphicx`), which has its own powerful set of manipulation options. We then see how the free-standing scaling and rotation works, and end by briefly describing some of the wrapper packages provided with the graphics bundle that offer alternative syntax and integration with other packages.

2.1 Loading the graphics packages

The features of the graphics bundle are not in the L^AT_EX 2_ε kernel, but are loaded by the standard, fully supported `color`, `graphics`, and `graphicx` extension packages. These packages rely on features that are not in T_EX itself but must be supplied by the “driver” used to print the dvi file. Unfortunately, not all drivers support the same features, and even the internal method of accessing these extensions varies among drivers. Consequently all these packages take options such as “`dvi`ps” to specify which driver is being used.

`graphics` and `graphicx` can both be used to scale, rotate, and reflect L^AT_EX material, or to include graphics files prepared with other programs. The difference is that `graphics` uses a combination of macros with a “standard” or T_EX-like syntax, while the “extended” or “enhanced” `graphicx` package presents a key-value type of interface for specifying optional parameters to the `\includegraphics` and `\rotatebox` commands.

2.1.1 Package device driver options

It is important to note that, when using L^AT_EX's graphics packages, the necessary space for the typeset material after performing a file inclusion or applying some geometric transformation, is reserved on the output page. It is, however, the task of the *device driver* (dvips, xdvi, dvipsone etc.) actually to perform the inclusion or transformation in question and show the correct result. Some drivers, such as previewers, are incapable of performing certain of the desired functions, and hence may display the typeset material overlapping the surrounding text. The driver you use must therefore be specified as an *option* to the graphics packages, e.g.,

```
\usepackage{dvips}{graphics}
```

if you use Tom Rokicki's dvips program, or

```
\usepackage[textures]{graphicx}
```

if you work on a Macintosh and use Blue Sky's Textures program. Table 2.1 on the facing page shows the drivers currently supported and their possible limitations. Support for further drivers is added once in a while, so it is worth checking the online documentation of the package for a driver not listed in this table.

It is also possible to specify a default driver in a *configuration* file `graphics.cfg` as follows:

```
\ExecuteOptions{emtex}
```

In this case the graphics packages pick up driver code for the emTeX T_EX system on a PC if the package is called without a driver option.

In addition to the driver options, the packages support some options controlling which features are enabled (or disabled):

- draft** Suppress all "special" features, such as including external graphics files in the final output. The layout of the page will not be affected, since the size information concerning the bounding box of the external material is still read by L^AT_EX. This option is of particular interest when a document is under development and you do not want to download the (often huge) graphics files each time. When draft mode is activated, the picture is replaced by a box of the correct size containing the name of the external file.
- hiresbb** Look for bounding box comments in PostScript files of the form `%HiResBoundingBox` (which typically have real values) instead of the standard `%BoundingBox` (which should have integer values).
- final** The opposite of `draft`. This option can be useful when, for instance, "draft" mode was specified as a global option with the `\documentclass`, e.g., for showing overfull boxes, but you do not want to suppress the graphics as well.

Table 2.1: Overview of color and graphics capabilities of device drivers.

<i>Option</i>	<i>Author</i>	<i>Features</i>
dvips	T. Rokicki	all functions
dvialw	N. Beebe	file inclusion with scaling only
dvipdf	S. Lesenko	all functions
dvilaser	Arbortext	file inclusion with scaling only
dvipsone	Y&Y	all functions
dvitops	J. Clark	all functions, but no nested rotations
dviwin	H. Sendoukas	file inclusion
dwiwindo	Y&Y	all functions
dvi2ps	original	file inclusion with scaling only
emtex	E. Mattes	file inclusion only, no scaling
ln	B. H. Kelly	file inclusion for DEC's LN03 printer
oztex	A. Trevorrow	file inclusion, color, rotation
pctexps	PCTeX	file inclusion, color, rotation
pctexwin	PCTeX	file inclusion, color, rotation
pctex32	PCTeX	all functions
pctexhp	PCTeX	file inclusion only
psprint	A. Trevorrow	file inclusion only
pubps	Arbortext	rotation, file inclusion
truetex	Kinch	graphics inclusion and some color
tcidvi	Kinch	TrueTeX with extra support for Scientific Word
textures	Blue Sky	all functions for Textures

With the `graphicx` package this and the previous option are also available locally for individual `\includegraphics` commands.

- hiderotate** Do not show the rotated material (for instance, in case the previewer cannot rotate material and produces error messages).
- hidescale** Do not show the scaled material (for instance, when the previewer does not support scaling).

2.2 Inclusion of graphics files

L^AT_EX 2_ε offers a uniform syntax for the inclusion of every kind of graphics file that can be handled by the different drivers. Generally, it relies on the file having an extension that identifies the type of file. Driver configuration files list the file extensions that the driver can handle. This list can be extended using the declarations described in Section 2.2.4. Although most of our examples use PostScript inclusions, the system also copes with files of type TIFF, PCX, PICT and so on (of course appropriate drivers must be available for each case).


```

%! A small Encapsulated PostScript image used in the examples
%%BoundingBox:100 100 172 172
100 100      translate % put origin at 100 100
  0  0      moveto    % define current point
 72 72      rlineto    % trace diagonal line
 72 neg 0    rlineto    % trace horizontal line
 72 72 neg  rlineto    % trace other diagonal line
stroke      % draw (stroke) the lines
  0  0      moveto    % redefine current point
/Times-Roman findfont % get Times-Roman font
 72         scalefont % scale it to 1 inch
           setfont    % make it the current font
(W) show    % draw an uppercase W

```

Figure 2.1: The contents of the file `w.eps`.

2.2.1 `\includegraphics` syntax in the graphics package

A graphics file *file* can be included using the graphics package with the command:

```
\includegraphics*[llx,lly] [urx,ury] {file}
```

Without optional arguments, the size of the graphic is determined by reading the external file *file* (containing the graphics itself or a description thereof, see below).

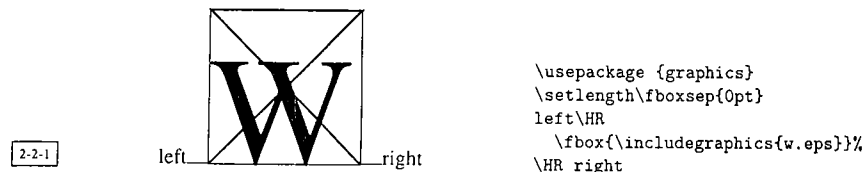
If [*urx,ury*] is present, then it should specify the coordinates of the top right corner of the image as a pair of T_EX dimensions. The default units are big (PostScript) points, i.e., [1in,1in] and [72,72] are equivalent. If only one optional argument is given, the lower left corner of the image is assumed to be at [0,0]. Otherwise, [*llx,lly*] specifies the coordinates of that point.

The starred form of the `\includegraphics` command “clips” the graphics image to the size of the specified bounding box. In the normal form (without the *), any part of the graphics image that falls outside the specified bounding box over-prints the surrounding text.

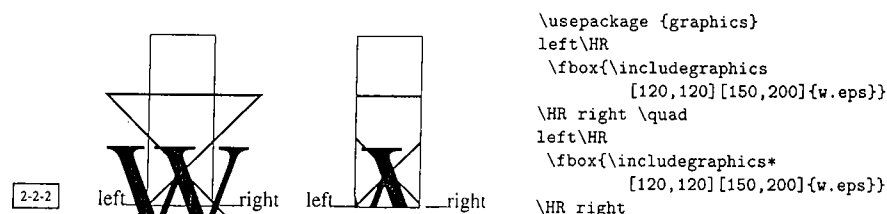
In the following examples we combine the `\includegraphics` command with other commands of the graphics package to show various methods of manipulating an included image. (Their exact syntax is discussed in detail in Section 2.3.) For this we use a small PostScript program (in a file `w.eps`) that paints a large uppercase letter W, and a few lines. Its source is shown in Fig. 2.1. Note the `BoundingBox` command, which stipulates that the image starts at the point 100, 100 (in big points), and goes up to 172, 172, i.e., its natural size is one inch by one inch.

In the examples that follow the `\fbox` command shows the space that L^AT_EX reserves for the included graphic (in all examples, although shown only on the first one, the `\fboxsep` parameter is set to zero to suppress extra space normally added by the `\fbox` command, and the baseline is indicated by the horizontal rules produced by the `\HR` command, defined as an abbreviation for `\rule{1em}{0.4pt}`).

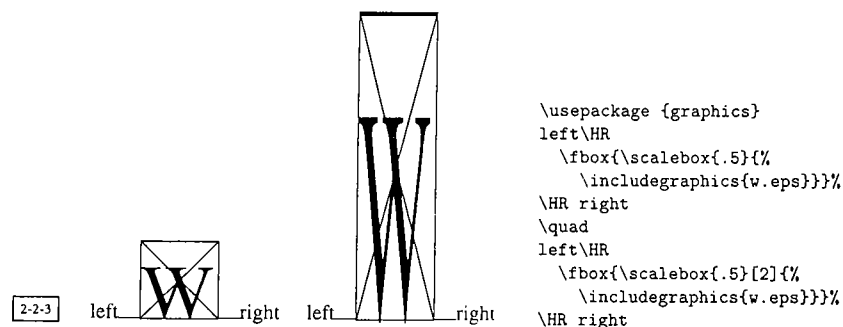
The first example shows the inclusion of the `w.eps` graphic at its natural size. Here the picture and its bounding box coincide nicely.



Next we specify a box that corresponds to a part of the picture (and an area outside it) so that some parts fall outside its boundaries, so as to overlay the material surrounding the picture. If the starred form of the command is used then the picture is clipped to the box, as shown on the right.



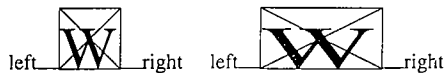
The next examples show how the `\scalebox` and `\resizebox` commands can be used together with `\includegraphics`; in both cases we can either specify a change in one dimension and have the other scale proportionally, or specify both dimensions to distort the image. The syntax of these commands is discussed in detail in Section 2.3.



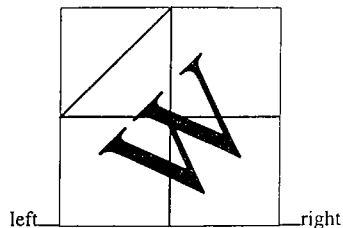
```

\usepackage {graphics}
left\HR
\fbbox{\resizebox{10mm}{!}{\%
\includegraphics{w.eps}}}%
\HR right
\quad
left\HR
\fbbox{\resizebox{20mm}{10mm}{\%
\includegraphics{w.eps}}}%
\HR right

```



Adding rotations makes things even more interesting:



```

\usepackage {graphics}
left\HR
\fbbox{\rotatebox{45}{\%
\includegraphics{w.eps}}}%
\HR right

```

2-2-4

2-2-5

2.2.2 \includegraphics syntax in the graphicx package

The extended graphics package `graphicx` offers a syntax for including external graphics files that is somewhat more transparent and user-friendly.

```
\includegraphics* [key val list] {file}
```

The starred form of the command exists only for compatibility with the standard version of `\includegraphics`, as described in Section 2.2.1. It is equivalent to specifying the `clip` key.

The *key val list* is a comma-separated list of *key=value* pairs for keys that take a value. For Boolean keys, specifying just the key is equivalent to *key=true*, not specifying the key is equivalent to *key=false*. Possible keys are:

<code>bb</code>	The “bounding box” of the graphics image. Its value field must contain four dimensions, separated by spaces.
<code>bbllx</code>	Lower left <i>x</i> coordinate (obsolete ¹).
<code>bbllx</code>	Lower right <i>y</i> coordinate (obsolete ¹).
<code>bburx</code>	Upper right <i>x</i> coordinate (obsolete ¹).
<code>bbury</code>	Upper right <i>y</i> coordinate (obsolete ¹).

¹Kept for backward compatibility only. [`bbllx=a`, `bbllx=b`, `bburx=c`, `bbury=d`] is equivalent to [`bb=a b c d`], so the latter form should be used.

<code>hiresbb</code>	This key makes L ^A T _E X search for %%HiResBoundingBox comments instead of the normal %%BoundingBox. Some applications use this to specify more precise bounding boxes, because the numbers can normally only have integer values. It is a Boolean, either “true” or “false”.
<code>viewport</code>	This key takes four arguments (like <code>bb</code>), but in this case the origin is taken with respect to the bounding box specified in the file. So to view a 20 bp square at the lower left-hand corner of the picture, specify <code>viewport=0 0 20 20</code> .
<code>trim</code>	Similar to the <code>viewport</code> key, but the four dimensions correspond to the amount of space to be trimmed (cut off) at the left-hand side, bottom, right-hand side and top of the included graphics.
<code>natheight</code>	Natural height of figure. ²
<code>natwidth</code>	Natural width of figure. ²
<code>angle</code>	Rotation angle (in degrees, counterclockwise).
<code>origin</code>	Origin for rotation, similar to the <code>origin</code> parameter of the <code>\rotatebox</code> command described on p. 46 and in Fig. 2.2 on p. 46.
<code>width</code>	Required width (the width of the image is scaled to that value).
<code>height</code>	Required height (the height of the image is scaled to that value).
<code>totalheight</code>	Required total height (the total height of the image is scaled to that value). This key should be used instead of <code>height</code> if images are rotated over 90 degrees, since the height can disappear (and become the depth) and L ^A T _E X will have difficulties satisfying the user's request.
<code>keepaspectratio</code>	Boolean variable that can have the values “true” or “false” (see above for defaults). When true, specifying both width and height parameters does not distort the picture, but the image is scaled so that neither of the width or height <i>exceeds</i> the given dimensions.
<code>scale</code>	Scale factor.
<code>clip</code>	Clip the graphic to the bounding box. It is a Boolean, either “true” or “false”.
<code>draft</code>	Locally switch to draft mode. A Boolean valued key, like <code>clip</code> .
<code>type</code>	The graphics type.
<code>ext</code>	The file extension of the file containing the image data.
<code>read</code>	The file extension of the file “read” by L ^A T _E X.
<code>command</code>	Any command to be applied to the file.

If for the first seven keys (`bb` through `trim`) the size is given without units, then T_EX's “big points” (equal to PostScript points) are assumed.

The first nine keys (`bb` through `natwidth`) specify the size of the image. This needs to be specified in case the file cannot be read by T_EX, or contains incorrect size information, or when you wish to clip the image to a certain rectangle.

²These arguments can be used for setting the lower left coordinate to (0 0) and the upper right coordinate to (natwidth natheight) and are thus equivalent to `bb=0 0 w h`, where *w* and *h* are the values specified for these two parameters.

The next seven keys (`angle` through `scale`) have to do with scaling or rotation of the included material. Similar effects can be obtained using the `graphics` package and the `\includegraphics` command by placing the latter inside the argument of a `\resizebox`, `\rotatebox` or `\scalebox` command (see the examples in Section 2.2.1 above and the in-depth discussion of these commands in Section 2.3).

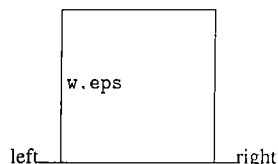
It is important to note that keys are read left to right, so that `[angle=90, totalheight=2cm]` means rotate by 90 degrees and then scale to a height of 2cm, while `[totalheight=2cm, angle=90]` would result in a final *width* of 2cm.

By default L^AT_EX reserves for the image the space specified either in the file or in the optional arguments. If any part of the image is actually outside this area, it will overprint the surrounding text. If the starred form is used or the `clip` option specified, any part of the image outside this area is not printed.

The last four keys (`type`, `ext`, `read`, `command`) suppress the parsing of the filename. When they are used, the main *file* argument should have no file extension (see the description of the `\DeclareGraphicsRule` command below).

Below we repeat some of the examples of Section 2.2.1 using the syntax of the `graphicx` package, and also show extra facilities offered by the extended package. In most cases the new form is easier to understand than the earlier version. In the simplest case without any optional arguments, the syntax for the `\includegraphics` command is the same in both packages.

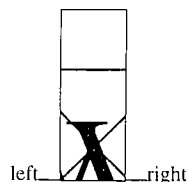
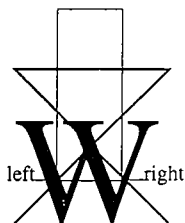
If we use the `draft` key we get just a frame showing the bounding box. This feature is not offered by the `graphics` package on the level of individual `graphics`.



```
\usepackage {graphicx}
left\HR
\includegraphics[draft]{w.eps}%
\HR right
```

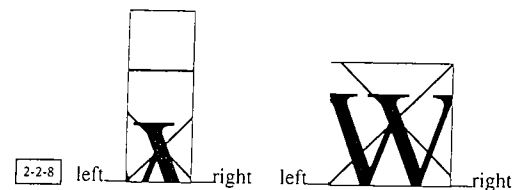
2-2-6

The effects of the `bb`, `clip`, `viewport` and `trim` keys are seen in the following examples.



```
\usepackage {graphicx}
left\HR\fbbox{%
\includegraphics
[bb=120 120 150 200]%
{w.eps}}\HR right
\qqquad
left\HR\fbbox{%
\includegraphics
[bb=120 120 150 200,clip]%
{w.eps}}\HR right
```

2-2-7



```
\usepackage {graphicx}
left\HR\fbbox{%
\includegraphics
[viewport=20 20 50 100,clip]%
{w.eps}}\HR right
\qqquad
left\HR\fbbox{%
\includegraphics
[trim= 5 5 10 10,clip]%
{w.eps}}\HR right
```

If you want to apply a scale factor to the image, use the `scale` key; with this, however, you can only scale the picture equally in both directions.

2-2-9



```
\usepackage {graphicx}
left\HR
\fbbox{\includegraphics
[scale=.5]{w.eps}}%
\HR right
```

To make the dimensions of an image equal to a given value, use the `width` or `height` keys:

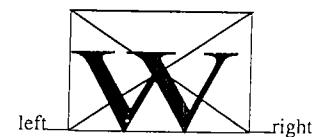
2-2-10



```
\usepackage {graphicx}
left\HR
\fbbox{\includegraphics
[width=10mm]{w.eps}}%
\HR right
```

If you use both keys simultaneously, you can distort the image to fit a specified rectangle as shown in the following example:

2-2-11



```
\usepackage {graphicx}
left\HR
\fbbox{\includegraphics
[height=20mm,width=30mm]{w.eps}}%
\HR right
```

However, you can make sure that the aspect ratio of the image itself remains intact by additionally specifying the `keepaspectratio` key. L^AT_EX then fits the image as best it can

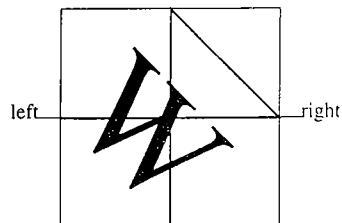
to the rectangle you specify.



```
\usepackage {graphicx}
left\HR
\fbbox{\includegraphics
[height=20mm,width=30mm,%
keepaspectratio]{w.eps}}%
\HR right
```

2-2-12

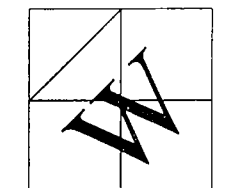
Rotations using the `angle` key add another level of complexity. Note especially how large the rotated L^AT_EX box becomes and how the image is placed in respect to the baseline. Depending on the `angle` used, the graphic might move up or down, as can be seen in the next examples.



```
\usepackage {graphicx}
left\HR
\fbbox{\includegraphics
[angle=-45]{w.eps}}%
\HR right
```

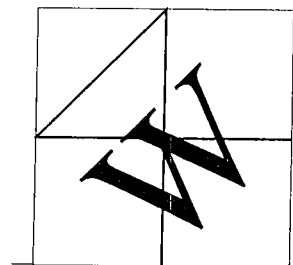
2-2-13

The fun really starts if you specify both a dimension and a rotation angle for an image, since the order in which they are given matters. The `graphicx` package interprets the keys *from left to right*. You should pay special attention if you rotate images and want to set them to a certain height. The next examples show the difference between specifying an angle of rotation before and after a scale command: in the first case the picture is rotated, and then the result is scaled while in the second the picture is scaled and then rotated.



```
\usepackage {graphicx}
\HR\fbbox{\includegraphics
[angle=45,width=30mm]%
{w.eps}}%
\HR
```

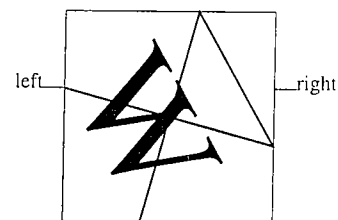
2-2-14



```
\usepackage {graphicx}
\HR\fbbox{\includegraphics
[width=30mm,angle=45]%
{w.eps}}%
\HR
```

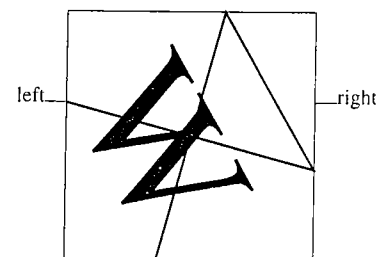
2-2-15

L^AT_EX considers the height and the depth of the rotated bounding box separately, and the `height` key refers only to the height, i.e., it does not include the depth. In general, you want the total height of a (rotated) image to fit in a given space, and in that case you should use the `totalheight` key (see Fig. 2.2 on p. 46 for a definition of the various dimensions defining a L^AT_EX box). Of course, to obtain special effects you can manipulate rotations and combinations of the height and width parameters at will. Here we show some of key combinations and their results.



```
\usepackage {graphicx}
left\HR\fbbox{%
\includegraphics[angle=-60]
{w.eps}}\HR right
```

2-2-16



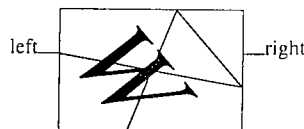
```
\usepackage {graphicx}
left\HR\fbbox{%
\includegraphics[angle=-60,%
height=15mm]%
{w.eps}}\HR right
```

2-2-17



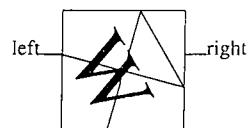
```
\usepackage {graphicx}
left\HR\fbbox{%
  \includegraphics[angle=-60,%
    totalheight=15mm]%
    {w.eps}}\HR right
```

2-2-18



```
\usepackage {graphicx}
left\HR\fbbox{%
  \includegraphics[angle=-60,%
    totalheight=20mm,%
    width=30mm]%
    {w.eps}}\HR right
```

2-2-19



```
\usepackage {graphicx}
left\HR\fbbox{%
  \includegraphics[angle=-60,%
    totalheight=20mm,%
    width=30mm,%
    keepaspectratio]%
    {w.eps}}\HR right
```

2-2-20

2.2.3 Setting default key values for the graphicx package

Instead of specifying the same set of key value pairs over and over again on individual `\includegraphics` commands, you can specify global default values for keys associated with such commands by using the `\setkeys` declaration provided by the `keyval` package, which is automatically included when `graphicx` is used.

```
\setkeys{identifier}{key val list}
```

The *identifier* is an arbitrary string defined by the macro designer; for example, for `\includegraphics` the string `Gin` was chosen. The *key val list* is a comma-separated list of key value pairs.

As an example, consider the case where `graphicx` is used and all figures are to be scaled to the width of the line. Then you would specify the following:

```
\setkeys{Gin}{width=\linewidth}
```

As a result, all images included with the `\includegraphics` command are automatically scaled to the current line width. (Using `\linewidth` in such a case is usually preferable to using `\columnwidth`, as the former changes its value depending on surrounding environments, such as `quote`, etc.)

You can specify defaults in a similar way for any key used with the `\rotatebox` command (the other command that has a key value syntax when `graphicx` is used). It has the *identifier* `Grot`; thus

```
\setkeys{Grot}{origin=ct}
```

specifies that `ct` should be used for the `origin` key on all `\rotatebox` commands unless locally overwritten.

2.2.4 Declarations guiding the inclusion of images

While key value pairs can only be set when the `graphicx` package is used, the declarations described in this section can be used with both the `graphics` and the `graphicx` package.

A list of directories where L^AT_EX should search for graphics files can be specified by the command `\graphicspath`, whose syntax is:

```
\graphicspath{dir-list}
```

where *dir-list* is a list of directories, each inside a pair of braces `{}` (even if the list contains only one directory). For example, `\graphicspath{{./eps/}{./tiff/}}` causes L^AT_EX to look in the subdirectories `eps` and `tiff` of the current directory. By default L^AT_EX looks for graphics files in the same directories where it looks for other files.

```
\DeclareGraphicsExtensions{ext-list}
```

The `\DeclareGraphicsExtensions` command lets you specify the behavior of the system when no file extension is given in the argument of the `\includegraphics` command. The argument *ext-list* is a comma-separated list of file extensions. Full filenames are constructed by appending each extension of the list *ext-list* in turn until a file corresponding to the generated full filename is found.

Since the previous algorithm tests for the existence of a file to determine which extension to use, when the `\includegraphics` command is specified without extension the graphics file must exist at the time L^AT_EX is run. However, if a file extension is specified, e.g., `\includegraphics{gr.eps}` instead of `\includegraphics{gr}`, then the graphics file need not exist at the time of the L^AT_EX run.¹ Since L^AT_EX needs to know the size of the image, this must be specified in the arguments of the `\includegraphics` command or in a file actually read by L^AT_EX (this can be the graphics file itself or another file specified with the `read` key or constructed from the list of file extensions. In the latter case that file must exist at the time L^AT_EX is run).

¹For instance, it can be created on the fly by specifying the *command* argument in the command `\DeclareGraphicsRule`.

With the declaration shown below, the `\includegraphics` command will first look for a file `file.ps` and, if no such file exists, for a file `file.ps.gz`:

```
\DeclareGraphicsExtensions{.ps,.ps.gz}
\includegraphics{file}
```

If you want to make sure that a full filename must always be specified, then the following declaration can be used. Note also that, in the cases below, the size of the (bitmap) image is specified explicitly on the `\includegraphics` command in each case.

```
\DeclareGraphicsExtensions{}
\includegraphics[1in,1in]{file.pcx}
\includegraphics[75pt,545pt][50pt,530pt]{file.pcx}
\includegraphics[bb=75 545 50 530]{file.pcx}
```

The action that has to take place when a file with a given extension is encountered is controlled by the following command:

```
\DeclareGraphicsRule{ext}{type}{read-file}{cmd}
```

Any number of these declarations is allowed. The meaning of the arguments is described below.

ext The extension of the file. It can be specified explicitly or, if the argument to `\includegraphics` does not have an extension, can be determined from the list of extensions specified in argument *ext-list* of the `\DeclareGraphicsExtensions` command. A star (*) can be used to specify the default behavior for all extensions that are not explicitly declared; i.e.,

```
\DeclareGraphicsRule*{eps}{*}{*}{*}
```

causes all undeclared extensions to be treated as EPS files, and the respective graphics files are read to search for a `%%BoundingBox` comment.

type The “type” of the file involved. All files of the same type are input with the same internal command (which must be defined in the corresponding driver file). For example, files with extensions `.ps`, `.eps`, or `.ps.gz` should all be classified as of type `eps`.

read-file The extension of the file that should be read to determine the size of the graphics image. It can be identical to *ext*, but, in the case of compressed or binary images, which cannot be interpreted easily by L^AT_EX, the size information (the bounding box) is normally put in a separate file (e.g., for compressed gzipped PostScript files characterized by the extension `.ps.gz`, the corresponding readable files could have extension `.ps.bb`). If the *read-file* argument is empty, i.e., `{}`, then the system does not look for an external file to determine the size, and the size must be specified in the arguments of the `\includegraphics` command. If the driver file specifies a procedure for reading size files for *type*, then that procedure is used, otherwise the procedure for reading `.eps` files is used. Therefore, in the absence of any specific other format, you

Table 2.2: Arguments of `\DeclareGraphicsRule`.

	<i>ext</i>	<i>type</i>	<i>read-file</i>	<i>cmd</i>
<i>basic PostScript</i>	<code>.ps</code>	<code>eps</code>	<code>.ps</code>	
	<code>.eps</code>	<code>eps</code>	<code>.eps</code>	
<i>dynamic decompression</i>	<code>.pz</code>	<code>eps</code>	<code>.bb</code>	<code>'gunzip -c #1</code>
	<code>.ps.gz</code>	<code>eps</code>	<code>.ps.bb</code>	<code>'gunzip -c #1</code>
	<code>.eps.gz</code>	<code>eps</code>	<code>.eps.bb</code>	<code>'gunzip -c #1</code>
<i>DOS-related formats</i>	<code>.tif</code>	<code>tiff</code>		
	<code>.pcx</code>	<code>bmp</code>		
	<code>.bmp</code>	<code>bmp</code>		
	<code>.msp</code>	<code>bmp</code>		
<i>Mac-related formats</i>	<code>.pict</code>	<code>pict</code>		
	<code>.pntg</code>	<code>pntg</code>		

can select the size of a bitmap picture by using the syntax for PostScript images, i.e., with a `%%BoundingBox` line.

cmd The command to be inserted in the `\special` argument instead of the filename. In general *cmd* is empty, but for compressed files you might want to uncompress the image file before including it in the file to be printed if such an operation is supported by the driver. For instance, with the dvips driver, you could use

```
\DeclareGraphicsRule{.ps.gz}{eps}{.ps.bb}{'gunzip #1}
```

where the argument `#1` denotes the full filename. In this case the final argument causes dvips to use the `gunzip` command to uncompress the file before inserting it into the PostScript output.

Various possibilities for the arguments of the `\DeclareGraphicsRule` command are shown in Table 2.2.

The system described so far can give some problems if the extension *ext* does not correspond to the *type* argument. One could, for instance, have a series of PostScript files called `file.1`, `file.2`, The graphics package cannot detect automatically that these are PostScript files unless a `type=eps` key setting is used for each file on the `\includegraphics` command. To handle this situation, you can define a default type by using a `\DeclareGraphicsRule` declaration for a type `*` as explained above.

2.3 Graphical manipulation of L^AT_EX objects

In addition to the `\includegraphics` command, the graphics and graphicx packages implement a number of graphical manipulation commands. With the exception of the `\rotatebox` command, which also supports a key value pair syntax in the graphicx package, the syntax for these commands is identical in both packages.

2.3.1 Scaling a L^AT_EX box

The `\scalebox` command lets you magnify or reduce text or other L^AT_EX material by a scale factor:

$$\boxed{\backslash\text{scalebox}\{scalefact\}\{material\}}$$

The first of the two (mandatory) arguments, (*scalefact*), specifies the factor by which both dimensions of the *material* are to be scaled. The following example shows how this works.

This text is large.

This text is normal.

This text is tiny.

```
\usepackage {graphicx}
\scalebox{2}{%
  This text is large.}\\
  This text is normal.\\
\scalebox{0.5}{This text is tiny.}
```

2-3-1

A supplementary optional argument, if present, is used to specify a separate vertical scaling factor.

$$\boxed{\backslash\text{scalebox}\{h\text{-scale}\}[v\text{-scale}]\{\text{material}\}}$$

This is demonstrated in the following examples, which also show how multiple lines can be scaled by using the standard `\parbox` command.

America
&
Europe

America & Europe

```
\usepackage {graphicx}
\framebox{\scalebox{2}{%
  \parbox{.5in}{America &\\Europe}}}
\framebox{\scalebox{2}[1.5]{%
  \parbox{.5in}{America &\\Europe}}}
```

2-3-2

```
\reflectbox{material}
```

This is a convenient abbreviation for `\scalebox{-1}[1]{material}`, as seen in the following example:

America? ? ? ? ?
America? ? ? ? ?

```
\usepackage {graphicx}
America?\reflectbox{America?} \\
America?\scalebox{-1}[1]{America?}
```

2-3-3

More interesting special effects can also be obtained. Note in particular the use of the zero-width `\makebox` commands, which hide their contents from `LATEX` and thus offer the

possibility of finetuning the positioning of the typeset material.

America?
America?
America?
America?
America?

2-3-4

```
\usepackage {graphicx}
America?\scalebox{-1}{America?}  \\\
America?\scalebox{1}{-1}{America?}\\
America?\makebox[0mm]{r}{%
    \scalebox{-1}{America?}}\\
\makebox[0mm]{l}{America?}%
    \scalebox{1}{-1}{America?}
```

2.3.2 Resizing to a given size

It is possible to specify that L^AT_EX material should be typeset to a fixed horizontal or vertical dimension:

```
\resizebox*{h-dim}{v-dim}{material}
```

When the aspect ratio of the material is to be maintained, then it is enough to specify one of the dimensions, replacing the other one with a “!” sign.

London,
Berlin &
Paris

2-3-5

```
\usepackage {graphicx}
\framebox{\resizebox{5mm}{!}{%
  \parbox{14mm}{London, \\\
    Berlin \&\\Paris}}}%
\framebox{\resizebox{!}{10mm}{%
  \parbox{14mm}{London, \\\
    Berlin \&\\Paris}}}%

```

When explicit dimensions for both *h-dim* and *v-dim* are supplied, then the contents can be distorted. In the following the baseline is indicated by an horizontal rule drawn with the `\HR` command.

Köln	Lyon	Oxford
—Rhein	Rhône	Thames—

Köln	Lyon	Oxford
—Rhein	Rhône	Thames—

2-3-6

```
\usepackage {graphicx}
\HR\begin{tabular}{@{}l@{}}
  K"o1n & Lyon & & Oxford \\\
  Rhein & & Rh\one & & Thames
\end{tabular}\HR\par\bigskip
\HR\resizebox{2cm}{.5cm}{%
\begin{tabular}{@{}l@{}}
  K"o1n & Lyon & & Oxford \\\
  Rhein & & Rh\one & & Thames
\end{tabular}}\HR
```

As usual, L^AT_EX commands involving box dimensions can refer to the natural lengths `\depth`, `\height`, `\totalheight`, and `\width` as dimensional parameters:

```

\usepackage {graphicx}
\HR\framebox{\resizebox{\width}
               {.7\height}{%
               \parbox{14mm}{London, \\\
               Berlin \&\\Paris}}}\HR
\framebox{\resizebox{\width}
               {.7\totalheight}{%
               \parbox{14mm}{London, \\\
               Berlin \&\\Paris}}}\HR

```

2-3-7

The unstarred form `\resizebox` bases its calculations on the height of the L^AT_EX material, while the starred `\resizebox*` command takes into account the total height (the depth plus the height) of the L^AT_EX box. The next `\parbox` examples, which have a large depth, show the difference.

```

\usepackage {graphicx}
\HR
\resizebox{20mm}{30mm}{%
\begin{tabular}{@{}l}
K\^o\l n & Lyon & & Oxford \\\
Rhein & Rh\^o\ne & & Thames
\end{tabular}}\HR
\HR
\resizebox*{20mm}{30mm}{%
\begin{tabular}{@{}l}
K\^o\l n & Lyon & & Oxford \\\
Rhein & Rh\^o\ne & & Thames
\end{tabular}}\HR

```

2-3-8

2.3.3 Rotating a L^AT_EX box

L^AT_EX material can be rotated through an angle with the `\rotatebox` command. The material in this section can be compared to Sebastian Rahtz's rotating package, described in Section 11.4 of Goossens et al. (1994). Although users are advised to use the commands described in this section instead, the package is still distributed, as it includes extra functionality for rotating floating objects.

```
\rotatebox{angle}{material}
```

The *material* argument is typeset inside a L^AT_EX box and rotated through *angle* degrees counterclockwise around the reference point.

2.3.3 Rotating a L^AT_EX box

```

\usepackage {graphicx}
\setlength{\fboxsep}{0mm}
\newcommand{\MyRot}[1]{%
\fbx{\rotatebox{#1}{%
rotation $#1\^{\circ}}}}
\MyRot{0} \MyRot{45} \MyRot{90}
\MyRot{135} \MyRot{180} \MyRot{225}

```

2-3-9

To understand where the rotated material is placed on the page, we need to look at the algorithm which is used. Below we show the individual steps carried out when rotating `\fbx{text}` by 75 degrees. Step 1 shows the unrotated text; the horizontal line at the left marks the base line. First the *material* (in this case `\fbx{text}`) is placed into a box. This box has a reference point around which, by default, the rotation is carried out. This is shown in step 2 (the original position of the unrotated material is shown as well for reference). Then the algorithm calculates a new bounding box, i.e., the space reserved for the rotated material, shown in step 3. After that the material is moved horizontally so that the left edge of the new and the old bounding box are in the same position (step 4). Then T_EX's typesetting position is advanced so that further material is typeset to the right of the bounding box in its new position, as shown the line denoting the base line in step 5. Step 6 shows the final result, again with the base line on both sides of the rotated material.

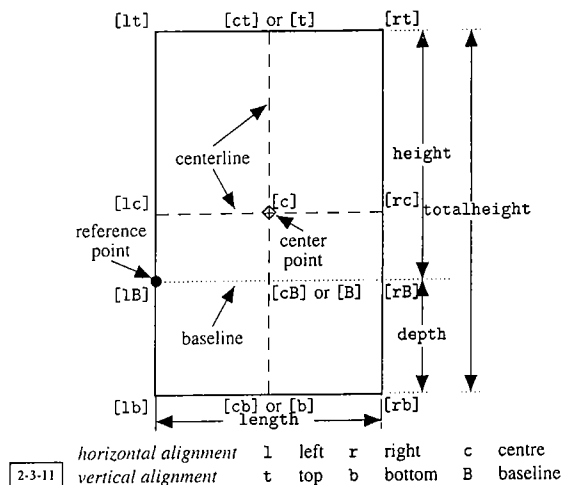
For more complex material it is important to keep in mind where the reference point of the resulting box is; the following example shows how it can be shifted by using the placement parameter of the `\parbox` command.

```

\usepackage {graphicx}
\setlength{\fboxsep}{0mm}
\providecommand{\HR}{\rule{1em}{0.4pt}}
\HR\fbx{\rotatebox{45}{%
\fbx{\parbox{3em}{%
{Red\\Green\\Blue}}}}}\HR
\HR\fbx{\rotatebox{45}{%
\fbx{\parbox[t]{3em}{%
{Red\\Green\\Blue}}}}}\HR
\HR\fbx{\rotatebox{45}{%
\fbx{\parbox[b]{3em}{%
{Red\\Green\\Blue}}}}}\HR

```

2-3-10

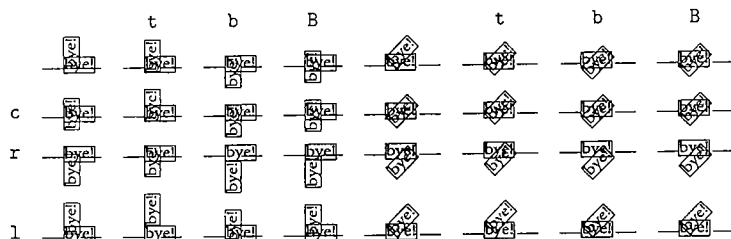
Figure 2.2: A L^AT_EX box and possible origin reference points.

The extended graphics package `graphicx` offers more flexibility in specifying the point around which the rotation is to take place by using *key val* pairs.

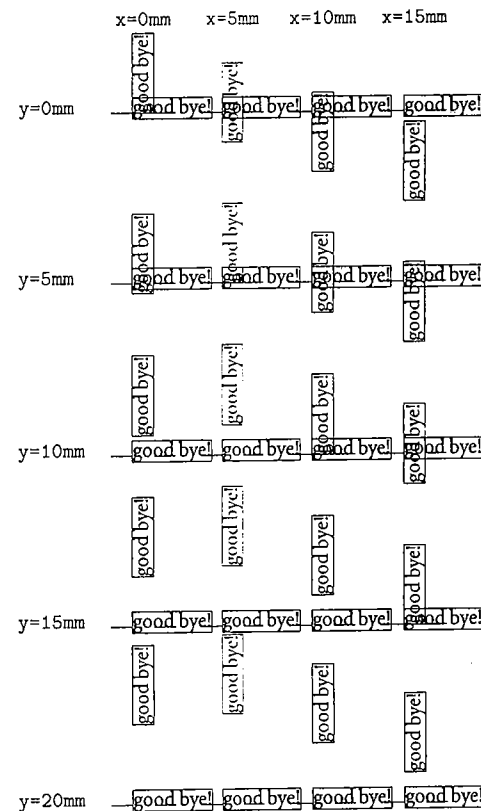
```
\rotatebox[key val list]{angle}{material}
```

The four possible keys in this case are `origin`, `x`, `y`, and `units`. The possible values for the `origin` key are shown in Fig. 2.2 (one value each for the horizontal and vertical alignments can be chosen), as are the actual position of these combinations with respect to the L^AT_EX box produced from *material*.

The effect of all possible combinations for the `origin` key on an actual L^AT_EX box can be studied below, where two matrices of the results are shown for 90°- and 45°-rotated boxes. To better appreciate the effects, the unrotated text is shown against a gray background.



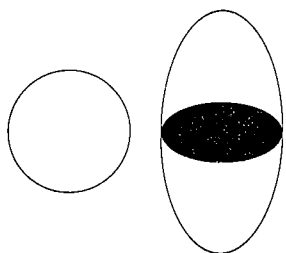
If the specification of the `origin` is not enough, it is also possible to supply the *x* and *y* coordinates (relative to the reference point) for the point around which the rotation is to take place. For this use the keys `x` and `y` and the format `x=dim`, `y=dim`. A matrix of some sample values and their effect of a box rotated by 90° are shown below.



The interpretation of the *angle* argument of `\rotatebox` can be controlled by the `units` keyword, which specifies the number of units counterclockwise in a full circle. The default is 360, so that using `units=-360` would mean that angles are specified clockwise. Similarly, a setting of `units=6.283185` changes the degree specification to radians. Rather than changing the `units` key on individual `\rotatebox` commands, you should probably set up a default interpretation using the `\setkeys` declaration as described in Section 2.2.3.

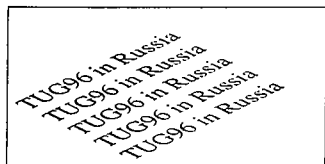
2.3.4 Combining effects

Rotation and scaling commands can be combined and nested *ad lib.*; we can also apply them to L^AT_EX picture environment objects.



```
\usepackage {graphics,pspicture}
\setlength{\unitlength}{1mm}
\begin{picture}(50,40)
\put(15,20){\circle{20}}
\put(40,20){%
\scalebox{1}{2}{\circle{20}}
\put(40,20){%
\scalebox{1}{.5}{\circle*{20}}}}
\end{picture}\qqquad
```

2-3-12



```
\usepackage {graphics,pspicture}
\fbbox{\resizebox{5cm}{20mm}{%
\rotatebox{45}{\parbox{3cm}{%
{\raggedright
TUG96 in Russia TUG96 in Russia
TUG96 in Russia TUG96 in Russia
TUG96 in Russia}}}}}
```

2-3-13

2.3.5 Applications

Rotating tabular material

Tabular material is a common candidate for rotation, both for whole tables and individual cells. The former is usually straightforward, but the latter sometimes involves L^AT_EX “tricks” to ensure that the amount of space allocated to a cell corresponds to what is visually correct, rather than what simple rotation produces. Using zero-width or zero-height rules usually does the trick. Thus the following table (a recoding using the graphics package of the example on page 324 of Goossens et al. (1994)) has rotated column heads, but they take up too much space:

Column 1	Column 2	Column 3
1	2	3
4	5	6
7	8	9

```
\usepackage {graphics,pspicture}
\begin{tabular}{rrr}
\rotatebox{45}{Column 1} & &
\rotatebox{45}{Column 2} & &
\rotatebox{45}{Column 3} \\
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{tabular}
```

2-3-14

If we use `\makebox` to hide the width of the rotated headings and center them on a fixed width over the columns, we achieve a much more pleasing result:

Column 1	Column 2	Column 3
1	2	3
4	5	6
7	8	9

2-3-15

```
\usepackage {graphics,pspicture}
% Zero height rule of width of one quad
\newcommand{\IR}[1]{\rule{1em}{0pt}%
\makebox[0cm]{c}{\rotatebox{45}{\ #1}}}}
\begin{tabular}{rrr}
\IR{Column 1} & & \IR{Column 2} \\
& & \IR{Column 3} \\
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{tabular}
```

Nested rotation inside tables is also possible, as are entire sideways floats: the examples in Tables 11.3 and 11.4 on pp. 325–326 of Goossens et al. (1994) can be easily recoded using the standard packages described in this chapter, together with the higher-level rotating package, which uses graphics.

Spanning braces

As we have just seen, material inside a tabular or array environment can be rotated at will. This functionality can be used to generate (rotated) braces that span multiple columns or rows.

The building block in the following examples is the `\Bpara` macro whose definition is shown below. The idea is to produce a brace of a certain height (argument 4), rotate it as needed (argument 3), but claim to L^AT_EX that no space is needed to typeset the result. That is why we use a `picture` environment of vanishing height and width. Arguments 1 and 2 are used in the `\put` statement, thus letting us fine-tune the placement by moving the reference point of the brace (the middle) horizontally or vertically by a number of points.

As we have seen in previous examples, rotation shifts the material up and down depending on the angle and the dimensions of the material rotated. To avoid this side effect, we pretend that the brace takes up no space in either direction by using a `\raisebox` and a `\makebox` command and specifying zero dimensions in all arguments. The actual brace is constructed by using `\left` and `\right` in math mode surrounding an invisible `\rule` that controls the height of the brace. Similar commands can be constructed for other extensible symbols.

```
% #1,#2 x and y coordinates of reference point of brace in pt
% #3 rotation angle for large brace in degrees
% #4 total height of brace in pt
\newcommand\Bpara[4]{%
\begin{picture}(0,0)
\setlength{\unitlength}{1pt}%
\put(#1,#2){\rotatebox{#3}{\raisebox{0mm}{\makebox[0mm]{%
\left\{\rule{0mm}{#4pt}\right\}}}}}%
\end{picture}}
```

Using the above definition means manually determining appropriate values for the four arguments, but this is a comparatively easy task: on the first run you specify some approximate values and once the data in your table is fixed you measure it with a ruler and correct them. As the command does not take up any space, it does not change the position of the tabular material.

A		13
		23
		33
41	42	43

```
\usepackage {graphics}
\begin{array}{cccc}
a_{11} & & & \\
a_{21} & & a_{22} & \\
a_{31} & & a_{32} & & \\
a_{33} & \Bpara{0}{7}{61}{65} & & \\
a_{41} & & a_{42} & & \\
a_{43} & & & a_{44} & \\
a_{51} & & a_{52} & & \\
a_{53} & \Bpara{-8}{-6}{-90}{58} & & \\
& & a_{54} & & a_{55}
\end{array}
```

2-3-16

```
\usepackage {graphics}
\Large
\begin{tabular}{|c|c|c|} \hline
\multicolumn{2}{|c|}{A} & 13 \\
\multicolumn{2}{|c|}{A} & 23 \\
\multicolumn{2}{|c|}{A} & 33 \\
\hline
41 & 42 & 43 \\
\hline
\end{tabular}
```

2-3-17

2.4 Other packages in the graphics bundle

2.4.1 The epsfig and rotating packages

Sebastian Rahtz's L^AT_EX 2.09 style files `epsfig` and `rotating` (see Sections 10.3 and 10.4 in Goossens et al. (1994)) have been rewritten as interfaces to the `graphicx` package, so that users of those styles can continue to use the syntax they are familiar with. For new documents, it is advisable to use L^AT_EX's native graphics commands directly as described in this chapter. It should be noted, however, that the `rotating` package offers some supplementary high level functionality, like the `sidewaysstable` and `sidewaysfigure` environments, that are not provided in the standard graphics packages. The `rotating` package also allows

the user to control the direction of rotation with the package options `clockwise` (the default) and `counterclockwise`.

2.4.2 The lscape package

David Carlisle's `lscape` package has also been rewritten from the L^AT_EX 2.09 version to use the standard `graphics`; it requires and takes the same options as the `graphics` package, and defines a `landscape` environment within which page bodies are rotated through 90°. The page head and foot are not affected: they appear in the standard (portrait) position.

2.4.3 The trig package

The `trig` package is not intended to be used directly in documents. It calculates the sine, cosine, and tangent trigonometrical functions that are used to calculate the space taken up by a rotated box. This package is used by Alan Jeffrey's `fontinst` package, which converts PostScript font files to a form usable by T_EX (see p. 393).

2.4.4 The keyval package

This package is also intended to be used by other packages, and a full description of its workings is beyond the scope of this chapter. It implements a generic way of setting "keys" as used by the `graphicx` package and splitting up the comma-separated lists of `key=value` pairs.

Working with METAFONT and METAPOST

In designing the $\text{\TeX}$ typesetting system, Donald Knuth soon realized that he would also have to write his own font design program. He devised METAFONT, a language for describing shapes, and a program to interpret that language and turn the shapes into a pattern of dots for a printing or viewing device. The result of Knuth's work was $\text{\TeX}$, METAFONT, and the extensive Computer Modern font family written in METAFONT. METAFONT has also been used to create special-purpose symbol fonts and some other font families.

The development of METAFONT as a font description language paralleled to some extent that of the PostScript language, which also describes character shapes very elegantly. PostScript's strategy, however, is to leave the rendering of the shape until the final printing stage, whereas that of METAFONT is to precompute the bitmap output and print it on a fairly dumb printing device.

Font design is a decidedly specialist art, one that most of us are ill equipped to tackle. But METAFONT is a very powerful system that can cope with most graphical tasks; and now a sibling program, METAPOST, is also available that uses essentially the same language but generates PostScript instead of bitmaps. In this chapter we consider how to use METAFONT and METAPOST (henceforth we use META to mean "both METAFONT and METAPOST") to draw pictures and shapes other than characters in fonts. The META language is unusually well suited to pictures that graphically express some mathematical object (like Hilbert or Sierpiński curves); this is not surprising, given that Knuth's aim was to describe font shapes mathematically.

We start with a brief look at the META language; our aim is to give readers new to META some ideas of its facilities and the level at which pictures can be designed. We try to

explain commands as they are used, but some examples may contain META code that is not explicitly described.

We next discuss how to run the METAFONT and METAPOST programs, and consider in some detail two high-level packages available to harness the power of METAPOST, providing box drawing and graphing facilities. We then look at programs which write META command for you, concentrating on the `mfpic` (L^AT_EX) package. We conclude with a brief description of how to make METAFONT write editable Encapsulated PostScript files.

3.1 The META language

The full intricacies of METAFONT are described in loving detail in Knuth (1986c); the manual for METAPOST (Hobby, 1992) not only describes the differences between the two systems, but is itself a good introduction to META. Alan Hoenig's book *T_EX Unbound* (1997) provides a wealth of material on METAFONT techniques; articles over many years in the journal *TUGboat* are also vital reading for those who want to delve deeply into METAFONT.

META lets you describe objects to be filled, scaled, rotated, reflected, skewed, and shifted, along with other complex transformations. Indeed, META programs can be regarded as specialized equation-solving systems that have the side effect of producing pictures.

A variety of complex data types are supported: `boolean`, `numeric`, `pair`, `path`, `pen`, `picture`, `string`, and `transform`. Let us look at some of these in more detail:

pair "Points" in two-dimensional space are represented in META with the type `pair`.

Constants of type `pair` have the form (x, y) , where x and y are both `numeric` constants. A variable p of type `pair` is equal to the `pair` expression `(xpart p, ypart p)`.

path A path is a continuous curve, which is composed of a chain of *segments*. Each segment has a shape determined by four *control points*. Two of the control points, the *key points*, are the segment's end points; very often we let META determine the other two.

pen Pens, a distinctive feature of META, are filled convex shapes that are moved along paths and affect the way lines are drawn in the result. Two pens are initially present in META, `nullpen` and `pencircle`. `nullpen` is the single point $(0, 0)$; it contains no pixels and can be used to fill a region without changing its boundary. By contrast, `pencircle` is circular, with the points $(\pm 0.5, 0)$ and $(0, \pm 0.5)$ on its circumference. Other pens are constructed as convex polygons via `makepen c`, where c is a closed path; the key points of c become the vertices of the pen. Pens themselves can be transformed.

picture A `picture` is a data type that can be used to store a sequence of META drawing commands; the result of a complete META program is often built up from the interaction of a set of pictures. The meaning of $v + w$ in METAFONT, for example, is a picture in which each pixel is the sum of the two pixels occupying the same position in pictures v and w respectively.

transform Affine transforms are the natural transformations of Euclidean geometry: the linear transformations augmented by translation. META can construct any affine transform and provides seven primitive ones (Knuth, 1986c, p. 141): *shifted*, *scaled*, *xscaled*, *yscaled*, *slanted*, *rotated*, and *zscaled*. The effect of most of the operations is self-evident; the last one, *zscaled*, uses a pair of numbers, interpreted as a complex number in Cartesian coordinates, i.e., complex multiplication.

Since a lot of writing META programs is describing geometrical shapes, the numeric support is extensive; for instance, Pythagorean addition (`++`) and subtraction (`+-`) are directly supported. Useful numeric functions include `length x` (absolute value of x), `sqrt x` (square root of x), `sind x` (sine of x degrees), `cosd x` (cosine of x degrees), `angle (x: y)` (arctangent of y/x), `floor x` (largest integer $\leq x$), `uniformdeviate x` (uniformly distributed random number between 0 and x), and `normaldeviate` normally distributed random number with mean 0 and standard deviation 1.

Program flow control is provided by a `for ... endfor` construct, with the usual conditionals. You can write parameterized macros or subroutines, and there are facilities for local variables and grouping to limit the scope of value changes.

Finally, META is famous for its ability to solve linear equations, including equations that involve points. You can thus define a point in terms of other points, e.g., `z3=1/2[z1,z2]` defines z_3 as the point in the middle of the line from z_1 to z_2 .

3.1.1 Examples of META programs

Here we give some examples of META code, all drawn using METAPOST. You should have little difficulty making them run under METAFONT as well, except that you will encounter problems with high-resolution output devices, as METAFONT will run out of memory when composing large pictures. This book was typeset at 2400 dpi, and some METAFONT examples were impossible to run at this resolution. Your only recourse is to work at a lower resolution (e.g., 300 dpi), or to break your picture into separate "characters" in a font and join them together in L^AT_EX. If you have a PostScript printer, it is almost certainly easier to use METAPOST.

We do not show the "wrapper" code that makes these examples runnable; see notes in Section 3.2.1 on p. 60 for how METAFONT creates a character and Section 3.2.2 on p. 62 for how METAPOST creates a figure.

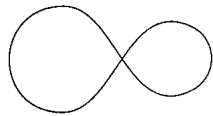
The simplest statement is `draw`, which takes a sequence of points separated by `..` and connects them with curves:

3-1-1



```
draw (0,0) .. (50,20) .. (40,30) .. (30,20)
```

The default unit here and elsewhere in these examples is a PostScript point (1/72 inch, TeX's "big point"). To close a object smoothly between its last and first points, the sequence can be terminated by `cycle`:



```
draw (0, 50)..(0,0)..
      (60,40)..(60,10)..cycle;
```

3-1-2

Straight lines are drawn by putting `--` instead of `..` between the points (the lines are actually implemented as specially constrained curves):



```
draw (0,0)--(50,20)--(40,30)--(30,20)
```

3-1-3

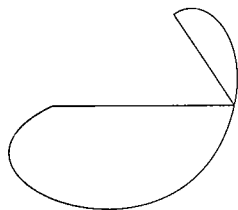
There are several ways of controlling curves: one can vary the angles at the start and end of the curve with `direction` (shortened to `dir`), the points that are to be the extremes (the upmost, the leftmost, etc.), and the inflection of the curve (with `tension` and `curl`). Thus the following code draws a crude coil by judicious use of `dir`:



```
u=2.5cm;
path p;
p= (0,0){dir 130}..
    {dir -130}(0.25u,0){dir 130}..
    {dir -130}(0.5u,0){dir 130}..
    {dir -130}(0.75u,0){dir 130}..
    {dir -130}(u,0);
draw p rotated -90;
```

3-1-4

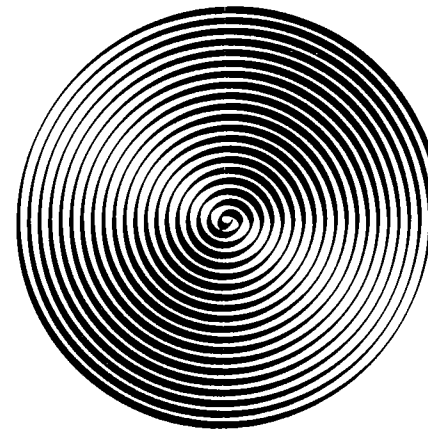
The following code shows the effect of `curl`: a straight line is drawn between three points and then a curve is drawn between the same points, with `curl` values:



```
path p,q;
u=.5cm;
q=(0u,0u)--(6u,0u)--(4u,3u);
draw q;
p=(0u,0u){curl 4000}..(6u,0u)
    ..{curl 4000}(4u,3u);
draw p;
```

3-1-5

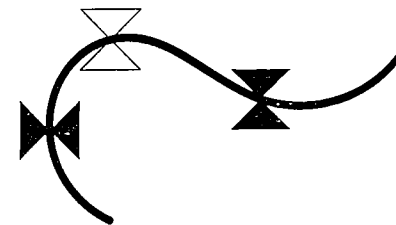
To demonstrate META's unusual "pens," we approximate a spiral drawn with a strange "nib". A colored version of this drawing appears in Color Plate I(a).



```
pickup pencircle scaled 4pt
  yscaled .2pt rotated 60;
n:=5;
for i := (n*20) step -(n) until (n):
  draw (i,0)..(0,i)..(-i,0)
    ..(0,-(i-n))..(i-n,0);
endfor
```

3-1-6

A very characteristic technique with META is creating a path and then using it several times with different transformations. The following code is an extract from a drawing of a kite's tail:



3-1-7

```
u=1cm;
path p[];
p1=(.5u,.5u)--(1.5u,.5u)--(.5u,1.5u)
  --(1.5u,1.5u)--(.5u,.5u)--cycle;
fill (p1 shifted (0,2.5u))
  rotatedaround ((u,3.5u),90);
draw p1 shifted (u,4u);
fill p1 shifted (3.5u,3u);
p2=(2u,2u)..(u,3.5u)..(2u,5u)
  ..(4.5u,4u)..(7u,5u);
pickup pencircle scaled 4pt;
draw p2;
```

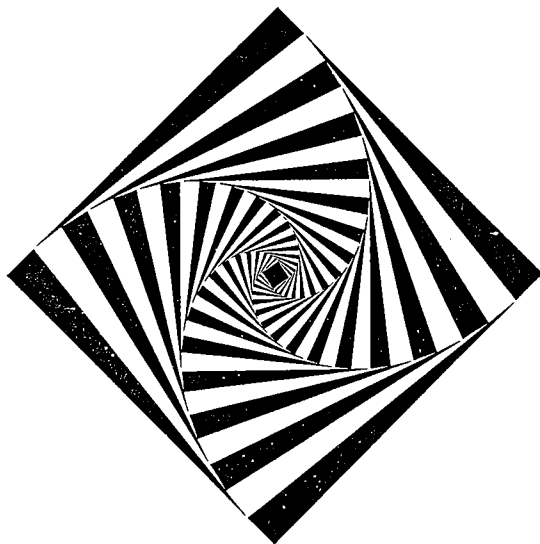
A more complicated picture developed by Alan Hoenig for his book *TeX Unbound* (1997) demonstrates the looping commands. Boxes of gradually decreasing size are drawn alternately white and black, each one rotated slightly with respect to the previous one.

```
boolean timetofillbox; timetofillbox := true;
partway := 0.9;
l := .75in; u := 1.75in; n := 4;
theta := 360/n;
z1 = (0,u);
for i := 2 upto n: z[i] = z1 rotated ((i-1)*theta); endfor
forever:
  path p;
```

```

p := z1
for j := 2 upto n: --z[j] endfor --cycle;
if timetofillbox:
  fill p;
  timetofillbox := false;
else:
  unfill p;
  timetofillbox := true;
fi
pair Z[];
for j := 1 upto n: Z[j] := partway[z[j-1],z[j]]; endfor
Z1 := partway[z[n],z1];
for j := 1 upto n: x[j] := xpart Z[j]; y[j] := ypart Z[j]; endfor
if not timetofillbox: l := abs(z1); fi
exitif l < .05u;
endfor

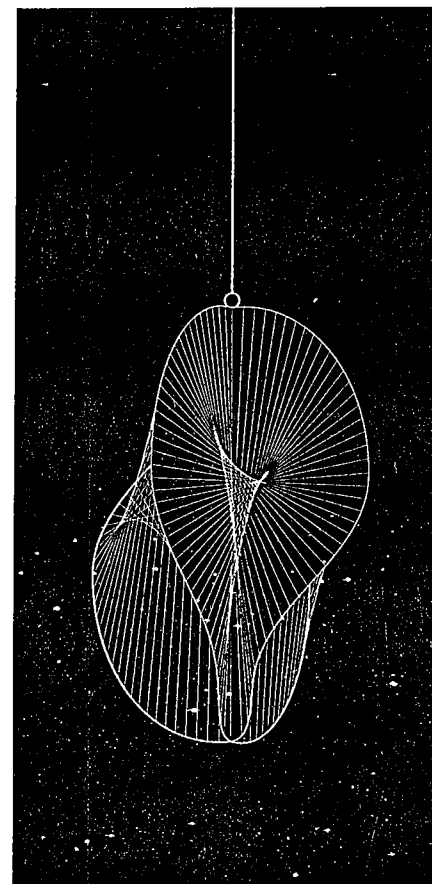
```



3-1-8

Finally, a pleasing side effect of META's curve-drawing abilities is to make it a nice artist's tool. Kees van der Laan demonstrates¹ in Fig. 3.1 how METAPOST lends itself to moving into 3D description on the one hand, and art on the other, with a picture modelled on Naum Gabo's constructivist art.

¹The code for this picture can be found on CTAN with the other examples from this book.



3-1-9

Figure 3.1: META picture after Naum Gabo, by Kees van der Laan

3.2 Using the META programs

3.2.1 Running METAFONT

To some extent METAFONT is like TeX; it has a low-level language on top of which you can write user-level macros. Most users do not start from scratch but load a prebuilt library of useful commands ("base" files), such as the "plain" base written by Donald Knuth. When a system implements "METAFONT," you can normally assume that the "plain" library is installed.

Each character, or object, in an METAFONT file is created by enclosing the code in a `beginchar...endchar` group. The `beginchar` has four parameters, the name of the letter being produced, the width, the height, and the depth. The following code, consisting of a very simple line, creates a "letter" at position "X" in the font:

```
beginchar ("X", 50, 50, 0);
draw (0,0)..(50,50)..(50,0)
endchar;
end
```

You write a font source simply by creating an METAFONT file with a set of characters defined. Now run the program on it to create the bitmap `gf` format (see Section 10.2.1 on p. 363 for more details on this and other font formats) output. An interactive session with METAFONT to run a font file might look like this (on a Unix machine):

```
> mf
This is METAFONT, Version 2.718 (C version 6.1)
**\mode=qms;input logo10
(/cdrom/texmf/fonts/source/public/mflogo/logo10.mf
(/cdrom/texmf/fonts/source/public/mflogo/logo.mf [77] [69] [84] [65]
[70] [80] [83] [79] [78]) )
Font metrics written on logo10.tfm.
Output written on logo10.300gf (9 characters, 1124 bytes).
Transcript written on logo10.log.
```

The line with the `**` prompt tells us that METAFONT is waiting for input. At that point we indicate the device for which we want to generate bitmapped fonts (see below) by using the `mode` command, and ask METAFONT to read a font source file and generate a bitmap font. Now the resulting `gf` file is transformed to `pk` format for use with a previewer or printer driver (though it is possible to use it directly):

```
> gftopk -v logo10.300gf
This is GFtoPK 2.3 (C version 6.1)
[./logo10.300gf->logo10.300pk]
'METAFONT output 1996.11.08:2035'
1124 bytes packed to 516 bytes.
```

You can also ask METAFONT to preview the character it is creating on the screen as it goes along.

The *mode* is very important; it is here that you tell METAFONT about your printer or screen. Since it is creating bitmaps, it needs to know how many pixels there are to an inch and how they are made. There is, for instance, a considerable difference between printers that (conceptually!) start with a white sheet of paper and put black dots on it and those that start with a black sheet and remove the areas that should be white. Mode descriptions for a large number of devices are available in a special METAFONT file called `modes.mf` maintained by Karl Berry (available with most METAFONT distributions and on CTAN). Typical names are "cx" (the Canon 300 dpi laser printer engine used in many machines), "ljfour" (HP LaserJet IV at 600 dpi) and "linoone" (Linotronic typesetter at 1270 dpi). The setup for each device provides values for a number of METAFONT internal variables, as this example for a LaserJet IV shows:

```
mode_def ljfour =
  mode_param (pixels_per_inch, 600);
  mode_param (blacker, .25);
  mode_param (fillin, 0);
  mode_param (o_correction, 1);
  mode_common_setup;
enddef;
```

This is explained in Knuth (1986c), pp. 90–93. After setting the mode, you should normally put the command `mode_setup`, which initializes the internal setup appropriately for the mode.

The *dpi* (dots per inch) in the mode description is used to name the output file in conjunction with the magnification. METAFONT's bitmap output is a `gf` (*generic font*) file, which can be compressed into an equivalent `pk` (*packed*) format by the auxiliary program `gftopk`. If users specify an unknown mode or no mode at all they will obtain filenames of extension `.2602gf`. This behavior invokes METAFONT's "proof" mode, used by font designers to get a magnified view of their fonts at a resolution of 2601.72 dpi—36 pixels per point (hence the 2602 file extension).

If we run METAFONT again, this time with the mode for a Linotronic typesetter at medium resolution, the output file has a different name and the `.gf` file is three times larger:

```
> mf
This is METAFONT, Version 2.718 (C version 6.1)
**\mode=linoone;input logo10
(/cdrom/texmf/fonts/source/public/mflogo/logo10.mf
(/cdrom/texmf/fonts/source/public/mflogo/logo.mf [77] [69] [84] [65]
[70] [80] [83] [79] [78]) )
Font metrics written on logo10.tfm.
Output written on logo10.1270gf (9 characters, 3900 bytes).
Transcript written on logo10.log.
> gftopk -v logo.1270gf
This is GFtoPK 2.3 (C version 6.1)
[./logo10.1270gf->logo10.1270pk]
'METAFONT output 1996.11.08:2036'
3900 bytes packed to 1568 bytes.
```


Besides the `gf` file, METAFONT usually creates a metric file (extension `tfm`); see Section 10.2.1 on p. 363. The metric file should always be the same, regardless of the mode or magnification selected; $\TeX$ can scale the information in the `tfm` files, but the glyphs in the bitmap files cannot be scaled. If you need a bigger character, you must run METAFONT again to generate the bitmap images at the correct size and resolution.

Since $\TeX$ font sizes increase in geometric ratios (so-called “magsteps,” which go in steps of 1.2), you can specify values for those magsteps to METAFONT and create larger characters, e.g.,

```
> mf
This is METAFONT, Version 2.718 (C version 6.1)
**\mode=cx; mag=magstep(1); input logo10;
(/cdrom/texmf/fonts/source/public/mflogo/logo10.mf
(/cdrom/texmf/fonts/source/public/mflogo/logo.mf [77] [69] [84] [65]
[70] [80] [83] [79] [78]) )
Font metrics written on logo10.tfm.
Output written on logo10.360gf (9 characters, 1280 bytes).
Transcript written on logo10.log.
```

The `mag=magstep(1)` means that, starting from the resolution defined in the mode (300 dpi in this case) a font of 1.2×300 dpi is created, i.e. `.360gf`.

There is a difference between a font of letters 10 points high (e.g., `cmr10`) run through METAFONT with a magnification of 1.2 so as to create letters 12 points high, and a font of letters 12 points high (e.g., `cmr12`) run through METAFONT without magnification. In the former case, the 10 point design is simply scaled up, while the latter has a real design for the larger size. In the context of drawing pictures, however, rather than fonts, this distinction is seldom made; pictures are usually created at the “right” size by specifying the scale in METAFONT.

3.2.2 An alternative to METAFONT — METAPOST

Derived by John Hobby from the source of METAFONT, METAPOST understands a language almost identical to that of METAFONT but generates PostScript files. These output files can be included as figures in a $\LaTeX$ document with the standard graphics package. METAPOST is designed not for creating fonts but for drawing general pictures and graphs, and differs from METAFONT in some important ways:

1. Since the output is not a bitmap but PostScript code, it is *not* device-dependent and you need not be concerned with *modes*;
2. The program is intended as a general tool, so it allows incorporation of normal $\LaTeX$ code in the picture for labels, captions, etc.;
3. Color support has been added.

The METAPOST manual (Hobby, 1992) details the differences between the METAFONT and METAPOST languages. Almost all drawing commands that work in METAFONT also work in METAPOST, but the latter has extra commands. One particularly useful METAPOST macro library, to make it easy to draw and link boxes, is described in Section 3.3.1 on p. 68.

METAPOST does not create high-quality PostScript Type 1 fonts from METAFONT sources, unfortunately, though it can be used to create the less sophisticated Type 3 font format.

Unlike METAFONT programs, which usually consist of `beginchar ...endchar` pairs, a METAPOST file usually consists of pairs of `beginfig` and `endfig` and an `end` statement after the final pair. The `beginfig` has a parameter that is the extension of the output file corresponding to each figure. You need not specify the size of the picture: METAPOST works that out and inserts the correct `%%BoundingBox` line in the output. The default units are PostScript points (what $\TeX$ calls “big points”, 72 to the inch rather than 72.27), though explicit lengths can of course be given. The following very simple METAPOST program draws a one-inch square:

```
beginfig(1);
draw (0,0)--(0,72)--(72,72)--(72,0)--cycle;
endfig;
end
```

If this is named `test.mp` and run through METAPOST, it produces the PostScript file `test.1`, which looks like this:

```
%!PS
%%BoundingBox: -1 -1 73 73
%%Creator: MetaPost
%%CreationDate: 1996.11.08:2155
%%Pages: 1
%%EndProlog
%%Page: 1 1
0 0 5 dtransform truncate idtransform setlinewidth pop [] 0 setdash
1 setlinejoin 10 setmiterlimit
newpath 0 0 moveto
0 72 lineto
72 72 lineto
72 0 lineto
closepath stroke
showpage
%%EOF
```

This is a normal EPS file² that can be included in your L^AT_EX document with the graphics package or any other application. If you use text labels, however, matters can become more complicated (see next section).

Color in METAPOST is represented by a triple defining it explicitly in terms of red, green, and blue, or alternatively in terms of one of the constants `black`, `white`, `red`, `green`, `blue`. The color triple is three real numbers between 0 and 1 for each of red, green and blue, and can be used in expressions just like `pair`; this makes possible color specifications like `.3blue` and `(.4, .2, .5) - (.9, .7, .3)`. If the resulting color component goes under 0 or over 1, it is automatically brought back to a legal value.

Adding text

One of the great strengths of METAPOST over METAFONT is that you can annotate your pictures with text; METAPOST can typeset text with ordinary PostScript fonts, or you can tell it to use T_EX. This text handling can be quite intricate, so we delay discussing some of it to the next section.

The most common way to add text is the `label` command, used as follows:

`label`_{suffix} (*string expression*, *pair expression*)

The *string expression* is printed at the position specified by the *pair expression* (i.e., a coordinate). The *suffix* is optional and can be one of `lft` (*left*), `rt` (*right*), `top` (*top*), `bot` (*bottom*), `ulft` (*upper left*), `urt` (*upper right*), `llft` (*lower left*), `lrt` (*lower right*); this specifies the relative position of the label to the *pair expression*. The distance from the point to the label is set by the variable `labeloffset`.

The *string expression* can take two forms:

1. If it is a META string (i.e., delimited by double quotes in the simplest case), then METAPOST sets the text directly in PostScript using the font specified by the `defaultfont` variable (usually set to be `cmr10`). This font must have a T_EX font metric available; note that ligatures and kerning in the `.tfm` file are *not* used.
2. Alternatively, if you bracket the text (do not put it in quotes this time) with `btex ...etex`, it is passed to T_EX for typesetting. This lets you use any T_EX or L^AT_EX commands, enter math mode, and so on.

In either form, the variable `defaultscale` (default 1) is applied to the basic font size (normally 10pt).

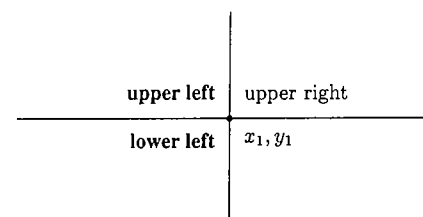
A special form of `label` prints a dot at the coordinate as well as the label:

`dotlabel`_{suffix} (*string expression*, *pair expression*)

²Notice that the bounding box is larger than you might expect, due to the width of the line drawing the box.

We demonstrate these commands with the following example:

```
u=2pt;
labeloffset:=10pt;
defaultfont:="ptmb8r";
dotlabel.ulft ("upper left", (100u,100u));
label.ur (btex upper right etex, (100u,100u));
label.llft ("lower left", (100u,100u));
label.lrt (btex $x_1,y_1$ etex, (100u,100u));
draw (50u,100u)--(150u,100u);
draw (100u,125u)--(100u,75u);
```



3-2-1

Two of the labels are META strings and are set in the font 'ptmb8r' (converted to Times-Bold—see Section 10.1.2 for the font-naming convention used here), and the other two are passed to T_EX and set in the default Computer Modern.

There is also a command `dotlabels` which can be used to label a set of points which you have previously created:

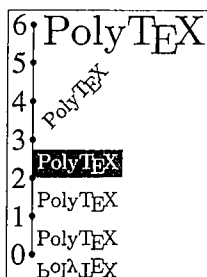
`dotlabels`_{suffix} (*pair*₁...*pair*_{*n*})

A dot is printed at each point in the list, and it is labelled with the name of the point variable, placed according to the *suffix*. This is used in the next example.

A label defined by a string has a default font and size; however, the label can also be passed to T_EX for typesetting with the format `btex typesetting commands etex`. If the latter form is used, the output is no longer completely portable, as it relies on a special structured comment in the PostScript output that only a few dvi drivers understand (dvips is the reference implementation for this support).

To save the label in a META "picture" variable, you can use the command `thelabel`; this lets you use the same label several times, with different transformations. Another useful technique is to apply the `bbox` command to a saved label, which returns the rectangular path enclosing a META picture. The following example uses this method to draw white text

on a green box and shows some of the other transformations you might need; it also shows how to apply `bbox` to the picture we are drawing, as an easy way of framing it:



```

defaultscale:=1.4;          u=1pt;
picture p,q;                gap:=18u;
for i := 0 upto 6:
  z[i]=(0u,i * gap * u);
endfor;
draw z0--z6;
dotlabels.lft(0,1,2,3,4,5,6);
q=thelabel.urt(btex\TeX\etex ,z0);
draw q;
draw q reflectedabout(z0,z0+(2u,0));
draw q shifted(0,gap) withcolor red;
fill bbox q shifted(0,2*gap*u) withcolor green;
draw q shifted(0,2*gap*u) withcolor white;
draw q scaled 2 shifted(0,5*gap*u);
p=q rotatedabout(z0,45);
draw p shifted(10u,3*gap*u);
draw bbox currentpicture withcolor green;

```

3-2-2

Adding text — some gory details

In the previous section we passed text to TeX using `btex ... etex`; but “TeX” in this case was plain TeX, not L^ATeX. Here we examine the process in more detail and see how to use L^ATeX; we also look carefully at font handling.

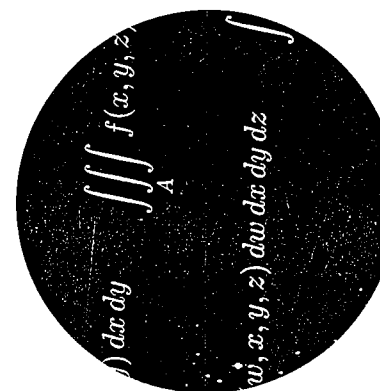
When METAPOST first meets `btex` it calls a script `makempx`; this runs a program called `mptotex` that scans the `mp` file and extracts all the `btex...etex` fragments into a temporary file, surrounding them by appropriate macros, so that each fragment is a “page” in this `TEX` file. It then calls `TEX` to typeset the material and passes the `dvi` file to another program (`dvitomp`) that rewrites it into an `mpx` file which METAPOST can read back in. When METAPOST continues after `makempx` has finished, it inserts a picture expression read from the `mpx` file for each `btex...etex` fragment. It is also possible in some installations to get device-independent troff output by invoking METAPOST with a `-T` command line option, which calls different internal scripts.

You can control this process in three ways:

1. Any text between commands `verbatimtex` and `etex` is written to the external file, but not used to generate a picture; this lets you define macros, set up L^AT_EX document-class or package loading, or change defaults.
2. When the script runs behind the scenes, it looks for an environment variable called `TEX` to work out which program to run (`tex`, `latex`, `amstex`, etc.); you must set this appropriately if, for instance, you have used `verbatimtex ...etex` to write `\documentclass{article}` to the file.

3. You can set the METAPOST variable `prologues`, which determines how fonts are handled. With the value 0, the output is dependent on an application (e.g., dvips) that can resolve references to TeX fonts which have to be downloaded and included in the output file. If the value is greater than 0, METAPOST tries to generate free-standing PostScript output using either troff (1) or TeX (any positive value e.g., 2, not equal to 1).

For example, suppose you want to typeset a couple of complex equations, needing the $\text{\texttt{AMS-L\TeX}}$ math package, as a $\text{\texttt{METAPOST}}$ “label”;³ the following picture shows what to do. When the fragment is run through $\text{\texttt{L\TeX}}$, it is in *horizontal mode*, so we cannot simply type in the displayed equation. Instead we put it inside a `minipage`:



3-2-3

```

verbatimtex
\documentclass{article}
\usepackage{times,amsmath}
\begin{document}
etex;
picture p; path q;
p:=thelabel{btex}\Large
\begin{minipage}{4in}
\begin{gather}
\iint\limits_A f(x,y)\,dx\,dy\qqquad
\iiint\limits_A f(x,y,z)\,dx\,dy\,dz\\
\iiint\limits_A f(w,x,y,z)\,dw\,dx\,dy\,dz
\qqquad\idotsint\limits_A f(x_1,\dots,x_k)
\end{gather}
\end{minipage}
etex,(0,0);
q:=fullcircle scaled 2.5in;
fill q withcolor black;
clip p to q;
draw p rotated (90) withcolor white;

```

This example also demonstrates how the shape of one picture can be used as a clipping path for another (we see only that portion of the equation that falls within the circle).

You do not need to add `\end{document}`, as METAPOST does this for you. Before running METAPOST, the environment variable `TEX` must be set to `"latex"`.

When `prologues` is set to 0, `METAPOST` does not try to write self-contained PostScript, but inserts special comments for each font in the output that give the name and size needed, like this:

```
%*Font: ptmr8r 9.96265 9.96265 28:c06
%*Font: cmsy10 9.96265 9.96265 01:8
%*Font: cmr7 6.97385 6.97385 31:8
```

³Remember, of course, that equation numbers and labels are *not* visible from the main L^AT_EX file that includes this picture.

When the picture is included in a L^AT_EX document, the dvi-to-PostScript program needs to read these commands and supply the fonts requested, either as bitmaps or in PostScript Type 1 form. Currently, only Tom Rokicki's dvips (see Section 11.2) understands this convention.

To ask METAPOST to write PostScript that can be included by any application or edited by programs like Adobe Illustrator, set prologues to 2 (avoid 1, this is for troff). Now, each time you use a font (either as a META string or in a T_EX fragment), METAPOST looks it up in dvips's psfonts.map control file (see Section 11.2.4) to find the real PostScript font name, and writes the PostScript file accordingly; it does *not* embed the font in the output, so unless your fonts are built into your printer, you will have to supply them explicitly with your job. If you use Type 1 Computer Modern fonts, you must ensure that they are listed in psfonts.map and downloaded to the printer.

The drawback of this approach is that you cannot control the font encoding when using PostScript fonts. For instance, the entry for "ptmb8r" in psfonts.map has instructions for re-encoding it differently from the standard encoding; however, the METAPOST setup currently ignores these instructions, so any characters outside the standard ASCII range are likely to be wrong. We recommend, if you want the METAPOST output to work with nonstandard characters, that you use font names and T_EX font metrics that are set up to use Adobe Standard Encoding (see Section 10.3.3) for text fonts. For mathematics, you have to make sure that no re-encoding is required. Using Alan Jeffrey's L^AT_EX package mathptm (see Section 10.1.1) helps, but this still uses Computer Modern for e.g., square root signs. As a last resort, you can create genuinely freestanding PostScript files by including the picture in a L^AT_EX document that is otherwise empty, and running that to PostScript (see Section 11.6 on p. 455 for a discussion of this technique).

3.3 METAPOST macro libraries

3.3.1 Boxing macros

One motivation for developing METAPOST was to provide tools comparable to those available to troff users. One of the best troff tools is the pic language (see Section 1.4.2 on p. 22, and Kernighan (1984)) which is often used for drawing and linking boxes. John Hobby's boxes library of METAPOST macros combined with standard METAPOST facilities can do a similar job.

The boxes package is loaded with the command

```
input boxes
```

The idea behind the package is that drawings consist of four types of statement:

1. Creating named boxes (META pictures);
2. Expressing relationships among boxes;

3.3.1 Boxing macros

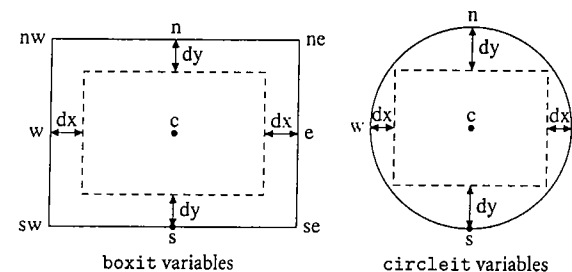
3. Asking the system to place the boxes;
4. Joining the boxes with lines and arrows.

This means that you need not specify exact positions for objects, but only their relationship to other objects.

The basic boxes commands to create boxes are

```
boxit.suffix(picture)
circleit.suffix(picture)
```

which draw rectangular and circular frames around the *picture*. An additional package, rboxes, provides another command, rboxit, that draws a rectangular frame with rounded corners. The result of these commands is named *suffix*, and you have access to a set of variables *suffix.c*, *suffix.e*, *suffix.n* etc., which are the coordinates of the edges of the object.



The variables *dx* and *dy* are the gap between the picture and the surrounding box; you can either set these explicitly or allow METAPOST to use the values defaultdx and defaultdy. The circmargin variable (default 2 points) determines the minimum gap around the text in circleit.

The relationships between boxes are specified as a set of equation statements:

```
boxjoin(equations)
```

Within the equations, you describe the relationships between notional boxes *a* and *b*; these are applied to each successive call of boxit. You can also explicitly write equations to specify the relationship between any boxes.

Once you have created a series of boxes, you can commit them to the page with:

```
drawboxed(box1,box2,...box_n) drawunboxed(box1,box2,...box_n)
drawboxes(box1,box2,...box_n) pic(box)
```

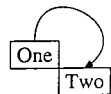
which simply require a list of objects defined with boxit or circleit. The first draws

the boxes and their contents, the second draws just the contents, and the third draws just the boxes. The fourth command returns the drawn box as a META "picture" that can be rendered with `draw` or `fill` in the usual way.

You can now join the boxes with normal META lines or curves. Two useful standard METAPOST macros make lines with arrowheads at one or both ends:

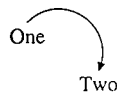
```
drawarrow path
drawblarrow path
```

The *path* specifications can use the special coordinates available for each box; e.g., `drawarrow one.n--two.s` draws a line between the top of box "one" and the bottom of box "two". The *dir* qualifier is useful for drawing curved lines to connect boxes, as these examples demonstrate:



```
boxjoin(a.se=b.nw);
boxit.one("One");
boxit.two("Two");
drawboxed(one,two);
drawarrow one.n{up}..two.n;
```

3-3-1



```
boxjoin(a.se=b.nw-(10,-10));
boxit.one("One");
boxit.two("Two");
drawunboxed(one,two);
drawarrow one.n{dir 45}..two.n;
```

3-3-2

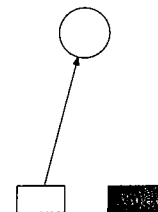
Often one wants to draw lines that conceptually link the center points of two boxes but stop at their boundaries; this can be accomplished with three useful standard METAPOST macros:

```
bpath box name
cutafter path
cutbefore path
```

`bpath` produces the bounding rectangle of a box as a path; when `cutafter` and `cutbefore` follow a drawing command, they control its interaction with their following *path*. Thus in the next example we say that the line joining the center of two boxes should start upon leaving the bounding rectangle of the first box, and stop when it reaches the

bounding rectangle of the second.

3-3-3



```
boxjoin(a.n=b.s-(20,60));
boxit.one("One");
circleit.two("Two");
drawboxes(one,two);
boxjoin();
boxit.three("Three");
three.w=one.e+(20,0);
fill bpath three withcolor green;
draw bpath three;
draw pic(three) withcolor red;
drawarrow one.c--two.c
cutbefore bpath one
cutafter bpath two;
```

We also show here how to specify the position of the third box in absolute relation to the first box: the default rules are turned off with an empty `boxjoin` equation.

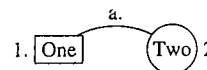
Labeling the boxes is usually easy; but what about labeling the lines joining the boxes? Here we can use the `point` macro:

```
point distance of path
```

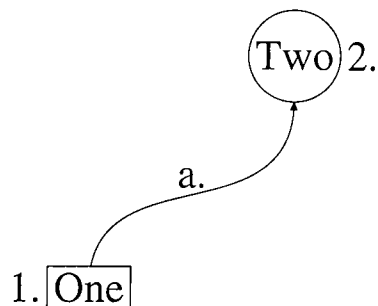
which returns a coordinate of a point *distance* along *path*. This can be used, with the macro `length` to compute the length of a path, as the parameter for `label`:

```
boxjoin(a.e=b.w-(30,0));
boxit.one("One");
circleit.two("Two");
drawboxed(one,two);
label.lft("1.",one.w);
label.rt("2.",two.e);
label.top("a.",
point .5*length (one.c{dir 45}..two.c)
of (one.c{dir 45}..two.c) );
draw one.c{dir 45}..two.c
cutbefore bpath one
cutafter bpath two;
```

3-3-4



A more elegant version of this example puts both drawing the line, and calculating the label coordinate into a META `vardef` macro (borrowed from the METAPOST graph documentation). This contains two statements, separated by `;`. The first draws the connecting line, and the second generates a coordinate that is returned as the value of the overall macro. Judicious use of `tension` and variations in direction make the joining curve a little more interesting.



```

vardef labelarrow
(suffix BoxA,BoxB) expr Line =
drawarrow Line
cutbefore bpath BoxA
cutafter bpath BoxB;
point (.5*length Line) of Line
enddef;
defaultscale:=2;
boxjoin(a.e=b.w-(.75in,1.5in));
boxit.one("One"); circleit.two("Two");
drawboxed(one,two);
label.lft("1.",one.w);
label.rt("2.",two.e);
label.top("a.",labelarrow(one,two)
one.c{dir90}..tension0.8..{dir90}two.s);

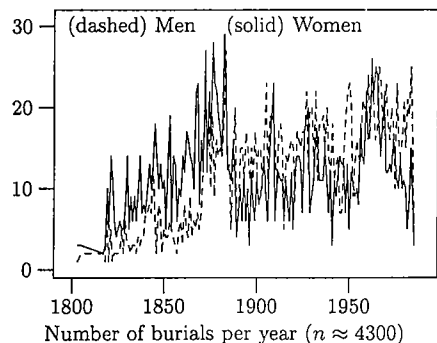
```

3-3-5

We conclude this section with an extended example in Fig. 3.2 on the next page showing how to use the boxes package for typical computer science diagrams. This is also printed in (rather arbitrary) color in Color Plate 1(b) to show some of the ways to use color with the boxes package. The same diagram is drawn with the Xy-pic T_EX package on p. 182; comparison of the code is interesting for those who need to produce this sort of picture.

3.3.2 The METAPOST graph package

METAPOST's author, John Hobby, has written a high-level library of METAPOST macros (Hobby, 1993) to provide a sophisticated package for drawing graphs, comparable to *grap* (see Bentley and Kernighan, 1984). We have already seen an example of *grap* in Chapter 1 (Fig. 1.22); we can draw a similar graph (using data from the Protestant Cemetery, Rome—see Rahtz (1987)) with METAPOST's *graph* (dash patterns, as used here, are a METAPOST addition to the META language):

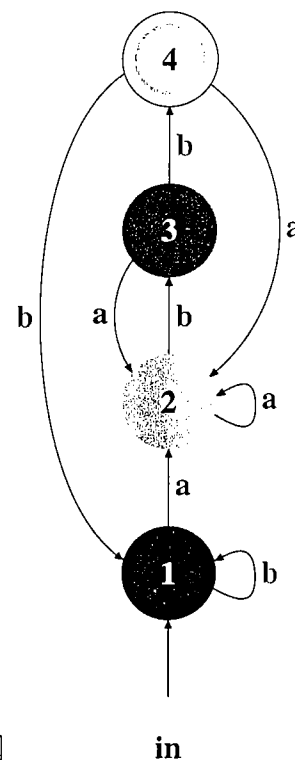


```

draw begingraph(2.5in,1.75in);
gdraw "yearm.dat" dashed evenly;
gdraw "yearw.dat";
glabel.lft
(btex (solid) Women etex, 1960,30);
glabel.lft
(btex (dashed) Men etex ,1870,30);
glabel.bot
(btex Number of burials per year
($n \approx 4300$) etex,OUT);
endgraph;

```

3-3-7



3-3-6

```

defaultfont:="ptmb8r";
vardef labelarrow
(suffix BoxA,BoxB) expr Line =
drawarrow Line
cutbefore bpath BoxA
cutafter bpath BoxB;
point (.5*length Line) of Line
enddef;
color yellow,orange;
yellow:=red+green; orange:=red+(green/2);
boxjoin(a.n=b.s-(0,.5in));
defaultscale:=1.5; circmargin:=4pt;
circleit.in("in"); circleit.one("1"); circleit.two("2");
circleit.three("3"); circleit.four("4");
boxjoin(); circmargin:=16pt; circleit.X(""); X.c=Four.c;
drawunboxed(one,two,three,four,in,x);
drawarrow in.n--one.s;
label.rt("a",labelarrow(one,two) one.c--two.c)
withcolor red;
label.rt("b",labelarrow(two,three) two.c--three.c)
withcolor green;
label.rt("b",labelarrow(three,four) three.c--four.c)
withcolor green;
label.rt("a",labelarrow(four,two) four.c{dir335}..
{dir205}two.c) withcolor red;
label.lft("a",labelarrow(three,two) three.c{dir205}..
{dir335}two.c) withcolor red;
label.lft("b",labelarrow(four,one) four.c{dir180}..
tension2..one.c) withcolor green;
label.rt("b",labelarrow(one,one) one.c{dir45}..one.c+(40,0)
..{dir120}one.c) withcolor green;
label.rt("a",labelarrow(two,two) two.c{dir65}..two.c+(40,0)
..{dir100}two.c) withcolor red;
fill bpath one withcolor blue;
fill bpath two withcolor yellow;
fill bpath three withcolor orange;
draw bpath four; draw pic two;
draw pic one withcolor white;
draw pic three withcolor white;
pickup pencircle scaled 2pt;
draw bpath x dashed evenly withcolor (1 , .75, .8);

```

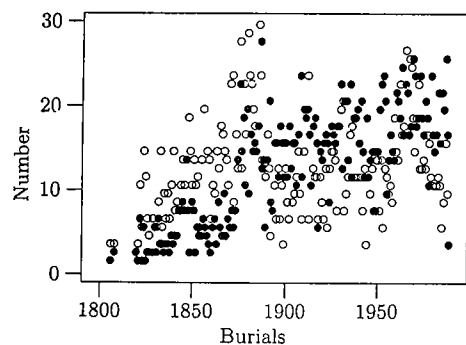
Figure 3.2: Finite-state diagram drawn with METAPOST

The graph package provides the commands for plotting data from external data files and labeling that you would expect, and:

- automatic scaling of data;
- automatic generation and labeling of tick marks or grid lines;
- multiple coordinate systems in the same picture;
- linear and logarithmic scales;
- plotting with arbitrary symbols;
- handling multiple columns in the same data file, with user-specified procedures.

You start by inputting the graph macro package and then surround each plot with a `begingraph...endgraph` pair; this returns a picture that you can render with `draw` (and on which you can perform other transformations like rotation). `begingraph` has a parameter (x and y dimensions separated by a comma) giving the size of the graph; the data is scaled to fit automatically with special “g”-prefixed forms of `draw` and `fill`. `gdraw` has the extra characteristic that, when followed by a file name, it draws the path created by reading coordinate pairs from the file. It can be followed by normal META qualifiers such as `withpen` or METAPOST’s `withcolor` and `dashed`. There are also “g” variants of other commands: `glabel`, `gdotlabel`, `gdrawarrow` and `gdrawbllarrow`.

`gdraw` can also be followed by a `plot` command with a parameter of a METAPOST “picture” to be plotted at each coordinate. This can be typeset by T_EX, as the following variation shows:



```
draw begingraph(2.5in,1.75in);
gdraw "yearm.dat"
plot btex $\bullet$ etex;
gdraw "yearw.dat"
plot btex $\circ$ etex;
glabel.bot
(btex Burials etex,OUT);
glabel.lft
(btex Number etex rotated 90,OUT);
endgraph;
```

3-3-8

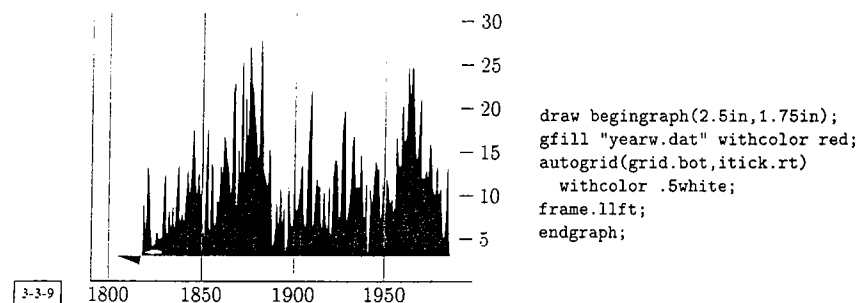
Notice that the `glabel` command has a special form of the position parameter, `OUT`, meaning that the text is to be placed outside the graph (it is normally used to place axis labels). For this graph we also added a label to the y axis, rotated by 90°.

Frames, ticks, grids, and scales

By default, graphs have a frame on all sides, no grid, and tick marks on the bottom and left. The frame can be altered with the `frame` command, which has the same set of optional suffixes as `label` (see p. 64). Grid lines and ticks are controlled with `autogrid`:

```
autogrid(x specification,y specification)
```

The specifications can have the values `grid`, `itick` or `otick`, which produce grid lines, inner ticks, or outer ticks; they can be suffixed with `.top` or `.bot` for the x axis and `.lft` and `.rt` for the y axis, as the following example shows:



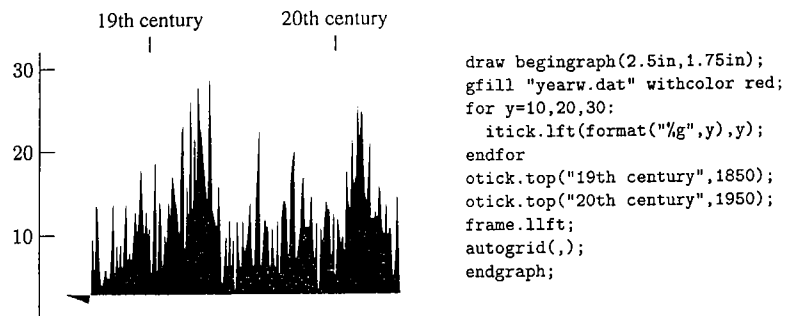
3-3-9

To override graph’s choice of where to put tick marks and how to write labels, you can add explicit ticks with `itick` or `otick` and grid lines with `grid`. These have the same suffixes as `autogrid` and are followed by a METAPOST picture variable containing a label or a `format` command, and a coordinate. `format` is used to control how numbers are printed:

```
format(specification,number)
```

The *specification* consists of an optional initial string, a percent sign, an optional number indicating precision (default 3), a conversion letter (e, f or g) and an optional final string. The conversion letter determines whether or not scientific notation is used; %g will use decimal format for most numbers. How the scientific notation used by `format` is typeset depends on a METAPOST macro called `init_numbers` (see manual); since this uses the `btex...etex` system, you may need to look at it carefully if you are concerned about precisely which fonts are used.

The following graph shows both types of explicit labeling; remember to turn off the normal marks at the end!



3-3-10

The labeling can also be changed by `setcoords`

```
setcoords(x style,y style)
```

The parameters for *x* and *y* can be set to `log`, `-log`, `linear`, or `-linear`.

While the program's scaling of data to fit the graph usually gives the right results, it can be overridden with `setrange`:

```
setrange(min,max)
```

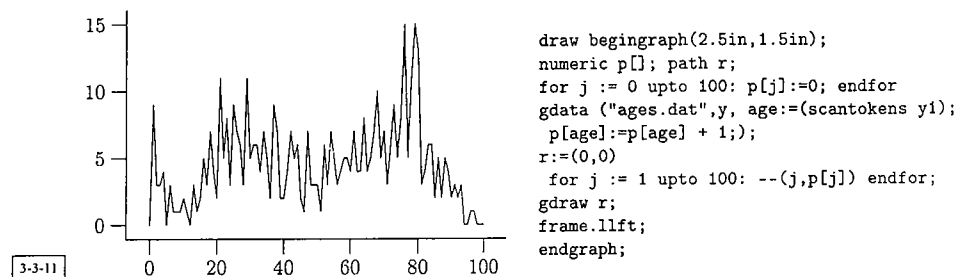
You need to supply the minimum and maximum coordinates. The special constant `origin` is a useful shorthand for `(0,0)`. To leave any value to be figured out by METAPST, specify whatever. If you specify no range at all, METAPST works it out from the data values and adds a small border.

Reading data files

Although the `gdraw` and `gfill` commands often suffice, we can get more control over the data read from a file by using `gdata`:

```
gdata(filename, variable, commands)
```

The *commands* are executed for every line of data in *filename*, with the values for each column available as, e.g., `c1`, `c2` ... `cn` for the variable name *c*. *filename* is a META string, so simple names should be enclosed in quotes (file names can also be computed from META variables.) Using some more data from the Protestant Cemetery in which each line consists of a person's age at death, we can show the distribution of mortality by age by accumulating data in an array and using that to create a path:

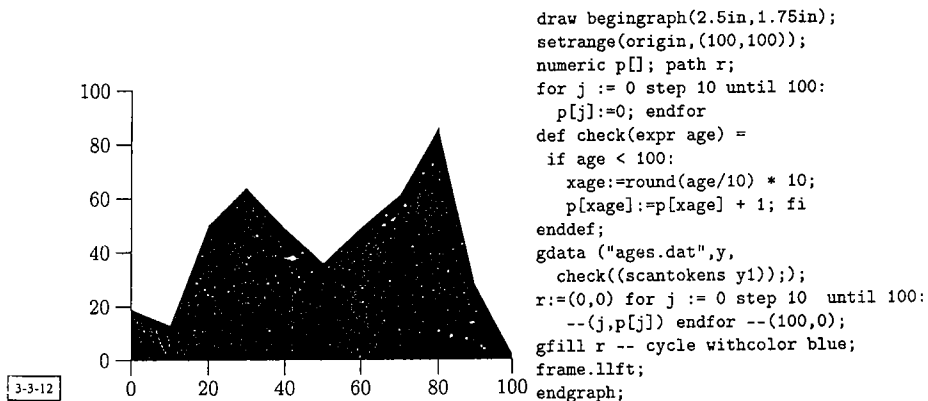


3-3-11

The only complications are the need to initialize the array and the conversion of the string representation read from the data file into a numeric value with `scantokens`.

When `gdata` reads data files, it stops when it reaches a blank line or end of file; if you start `gdata` again with the same file name, it carries on reading another set of data. This allows you to put all your data sets in one file, but use it with care. One problem is that data files remain open if there is a blank line at the end, since METAPST thinks some more data might follow; if you have many small data files, this situation can cause a METAPST error—check the end of your files.

This display in the example above is not very readable; it might be better to accumulate data per decade of death from the file. As this gets a little more complicated, we abstract the job into a METAPST macro called by the `gdata` command:



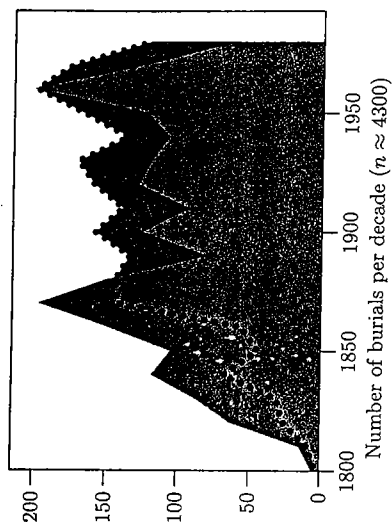
3-3-12

It is often useful to accumulate points on a path for each line read from the data file; the macro `augment` is provided for this. Given a suffix of a variable name of type "path" and a parameter of a coordinate, `augment` creates the path if it does not exist or adds the point

to an existing path. We use this to show the gravestone data again, this time processed to provide separate figures of deaths per decade for women (column 2) and men (column 3):

```
1800 3 6
1810 9 15
1820 26 64
1830 31 88
...
```

For each decade, we keep track of the last point reached and augment separate paths for male and female; these are then shaded in different colors (this drawing also appears as Color Plate I(c)) to show how the male and female patterns vary over time. We need to know the last decade in order to establish a sensible corner for the filled shape. The female pattern appears as a dotted line on top of the male shading.



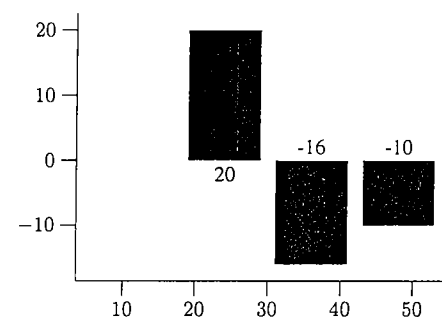
```
path m,w,last;
draw begingraph(3in,2in);
setrange((1800,0),(whatever,whatever));
gdata("decade.dat",y,
  last:=((scantokens y1),0);
  augment.w(y1,y2);
  augment.m(y1,y3));
gfill(1800,0)--w--last--cycle
  withcolor red;
gfill(1800,0)--m--last--cycle
  withcolor green;
pickup pencircle scaled 3pt;
gdraw w dashed withdots;
pickup pencircle scaled .75pt;
glabell.bot (btex Number of burials per decade
  ($n \approx 4300$) etex,OUT);
endgraph
rotated 90;
```

3-3-13

The example demonstrates that the graph macros return a META picture that can then be transformed (in this case rotated).

Different graph types

With a little effort, graph can draw bar charts; to demonstrate this, we copy a chart from Goossens et al. (1994), p. 287, that was made with the L^AT_EX bar package. Our technique is to make a single path out of all the bars and fill the result at the end:

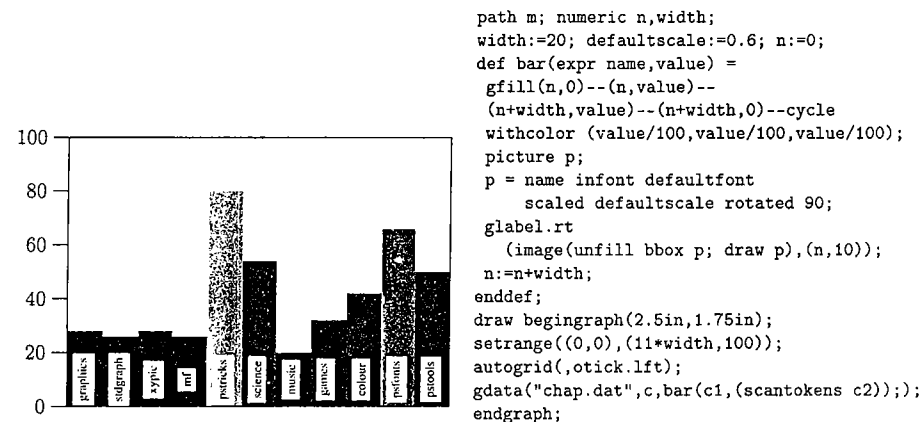


3-3-14

```
path s; numeric x,y;
draw begingraph(2.5in,1.75in);
gdata("students.dat",c,
  x:=(scantokens c1)*12;
  y:=(scantokens c2);
  augment.s((x-5,0)--
  (x-5,y)--(x+5,y)--
  (x+5,0));
  if y < 0: glabell.top(c2,(x,0)); fi
  if y > 0: glabell.bot(c2,(x,0)); fi
);
gfill s--cycle withcolor .5white;
frame.llft;
endgraph;
```

We explicitly work out the corners of each bar and allow for their width by multiplying the x values by 12; the bars themselves span 5 units on either side of the data point, so there is a gap of 2 units between each one.

A similar technique is used in the next chart (also printed in Color Plate I(d)) which shows the number of pages in each chapter of this book at one stage in its production; this time we draw each bar separately, so that they can be shaded according to the values. The work is delegated to a macro, which also prints a rotated label for each bar. Because explicit x labels are supplied, labeling of the x axis is suppressed.



3-3-15

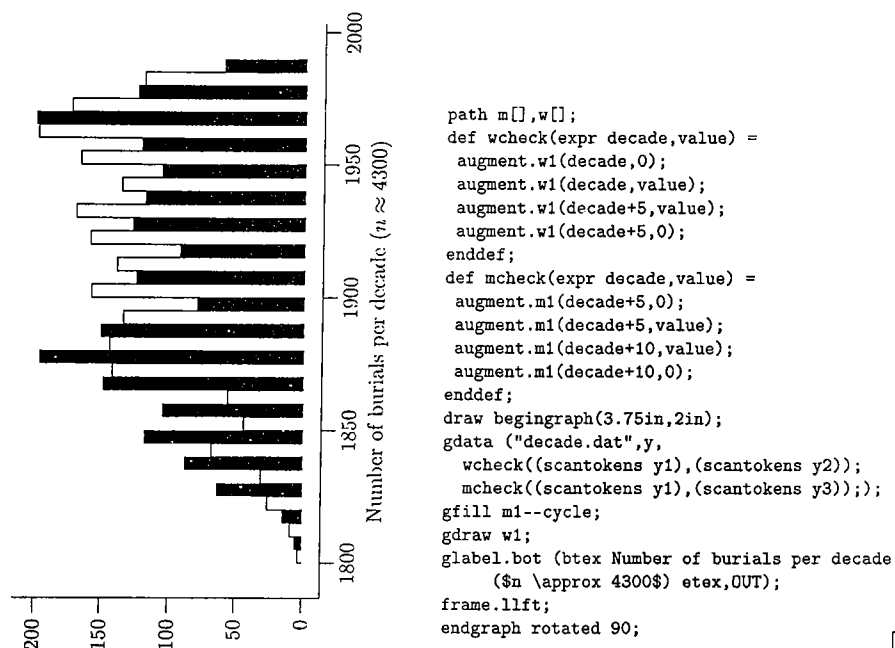
```
path m; numeric n,width;
width:=20; defaultscale:=0.6; n:=0;
def bar(expr name,value) =
  gfill(n,0)--(n,value)--
  (n+width,value)--(n+width,0)--cycle
  withcolor (value/100,value/100,value/100);
  picture p;
  p = name infont defaultfont
    scaled defaultscale rotated 90;
  glabell.rt
    (image(unfill bbox p; draw p),(n,10));
  n:=n+width;
enddef;
draw begingraph(2.5in,1.75in);
setrange((0,0),(11*width,100));
autogrid(.otick.lft);
gdata("chap.dat",c,bar(c1,(scantokens c2)));
endgraph;
```

The string value read from the first data column is put into a METAPOST picture variable by using the low-level command `infont`. This lets us use the `bbox` technique to give the extent of the text, which is made white with `unfill`. `image` is a useful macro that yields the picture resulting from a sequence of drawing commands; we use that as a label. The data

for this graph starts as follows:

```
graphics 28
stdgraph 26
xypic 28
mf 26
...
```

We can also present our earlier “decade” data as a dual bar chart, with male and female figures side by side. To do this we maintain two separate paths, fill one and leave the other as an outline:

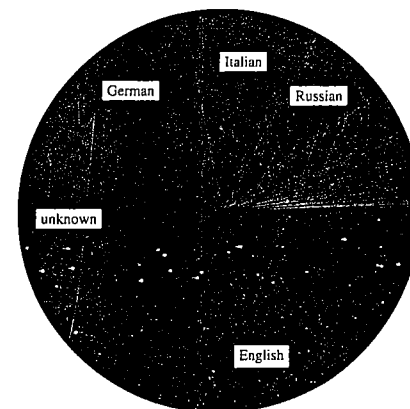


With care, we can even draw pie charts using similar ideas. The following example reads data about gravestones in the Protestant Cemetery in the following form:

```
Romanian 1 0.02796420582
Czech 2 0.05592841163
.....
```

```
Italian 391 10.93400447
German 508 14.20581655
unknown 599 16.75055928
English 1462 40.8836689
```

Here the second column is the number of gravestones per nationality and, to make the code less complicated, the third column is the percentage of the total. For each pie wedge, we use the `buildcycle` macro to find the smallest enclosed shape from the union of a whole circle and two lines extending from the center at the starting and closing angle of the segment. The fill color of the wedge is derived from the percentage.

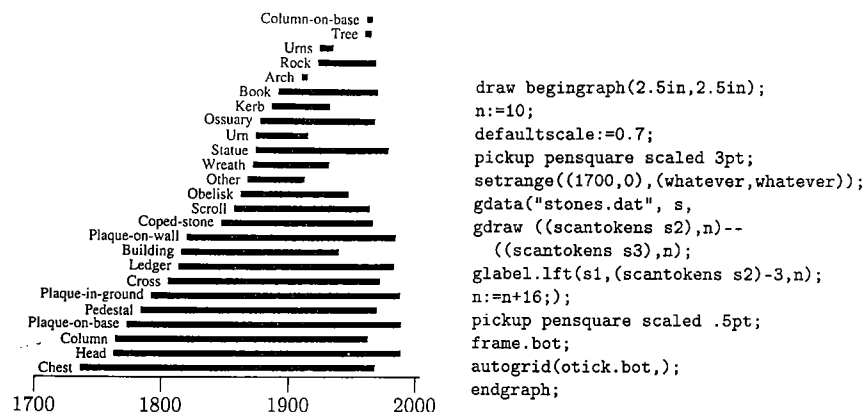


```
numeric r,last; path c,w;
r:=5; c:=fullcircle scaled 2r;
last:=0.0;
def wedge (expr lang,value,perc) =
  numeric current,n,half,xoff,yoff;
  picture p;
  n:=perc*3.6;
  current:=last+n; half:=last+(n/2);
  w:=buildcycle((0,0)--(2r,0) rotated last,
    c, (2r,0)--(0,0) rotated current);
  gfill w withcolor (0,0.8-(perc/100),0);
  gdraw w;
  if perc > 5:
    p = lang infont defaultfont
      scaled defaultscale;
    glabel(image(unfill bbox p; draw p),
      3/4r*dir(half));
  fi;
  last:=current;
enddef;
draw begingraph(3in,3in);
defaultscale:=0.7;
gdata ("langs.dat",c,
  wedge(c1, (scantokens c2),
    (scantokens c3)));
autogrid(,); frame withcolor white;
endgraph;
```

For a different rendering of this chart using PSTricks, see p. 134. The placement of the labels in the pie bears a little examination; they are placed in the center of each wedge, three quarters of way along the radius. An alternative algorithm to work out the coordinate would be $((r*3*\cosd(half))/4, (r*3*\sind(half))/4)$, using META's sine and cosine functions.

Another type of graph has a linear x scale and uses the y axis simply to compare sets of data. The following graph uses our cemetery data to show the first and last occurrences

of each type of gravestone. The code is straightforward except that we draw the lines with a different sized pen (with square ends) and revert to a thin line to draw the scale and frame (only on the bottom, since the y axis is not linear).



3-3-18

The data, ranked in order of first occurrence, starts like this:

```

Chest 1738 1966
Head 1765 1986
Column 1766 1960
Plaque-on-base 1775 1986
Pedestal 1786 1967
Plaque-in-ground 1794 1985

```

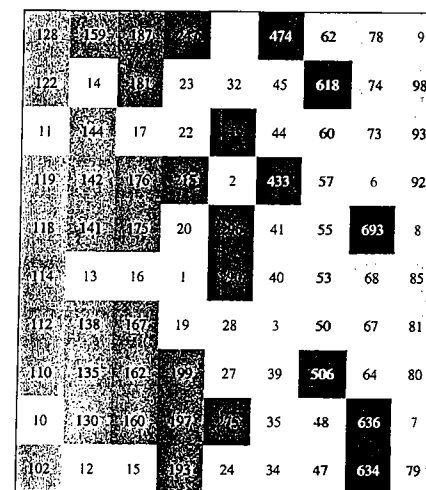
Our last example is more unusual. We want to plot data from a survey grid and shade each grid square according to its data value; in the data file the first two columns are the coordinates of the lower left corner of the grid square, the third column is the absolute data value, and the fourth column is a percentage version:

```

2 1 102 85
2 2 10 98
2 3 110 84
2 4 112 83
2 5 114 83
...

```

The text is printed in white or black depending on the percentage. This picture is also rendered using a $\text{\LaTeX}$ table and the color package on p. 325.



3-3-19

```

def sq(expr x,y,num,perc) =
  gfill(x,y)--(x+10,y)--
  (x+10,y+10)--(x,y+10)--cycle
  withcolor (perc/100,perc/100,perc/100);
  glabell(num,(x+5,y+5))
  if perc < 50: withcolor white fi;
enddef;
defaultscale:=0.7;
draw begingraph(70mm,80mm);
setrange((20,10),(110,110));
autogrid(,);
gdata ("pot.dat",c,
sq((scantokens c1)*10,
(scantokens c2)*10,
c3,(scantokens c4)););
endgraph;

```

3.4 $\text{\LaTeX}$ interfaces to META

Although META produces beautiful pictures, it is not an easy language for a casual user who wants relatively simple diagrams. There are three ways to deal with this:

1. Use of high-level libraries of META code; surprisingly, few such libraries exist — METAPOST's graph macros (see Section 3.3.2 on p. 72) are probably the most "packaged";
2. use of an interactive drawing package that can generate META output; there is a program `fig2mf` that converts Unix `xfig` files, but it has not been kept up to date;
3. use of a $\text{\LaTeX}$ package that generates the necessary META code.

Since most interactive drawing packages keep a rather simplistic internal representation of their pictures, META versions of their output are of limited use. Thus we could use `gnuplot` or `AutoCAD`, save the result as HPGL, and convert it to META, but the code is hardly editable. A map of East Africa (Fig. 3.3) drawn with `AutoCAD` and converted to METAFONT creates lines of code like this:

```

pickup pencircle scaled 0.1mm;
draw (10.732mm,11.563mm)--(10.651mm,11.517mm)--(10.592mm,11.505mm)--
(10.545mm,11.493mm)--(10.487mm,11.493mm)--(10.440mm,11.505mm)--

```

```
(10.405mm,11.540mm)--(10.346mm,11.563mm)--(10.299mm,11.599mm)--
(10.264mm,11.622mm)--(10.217mm,11.645mm); draw (10.217mm,11.645mm)--
(10.159mm,11.680mm)--(10.100mm,11.727mm)--(10.065mm,11.751mm)--
(10.007mm,11.786mm)--(9.960mm,11.798mm)--(9.913mm,11.821mm)--
(9.714mm,11.868mm)--(9.667mm,11.868mm);
```

This sort of code could hardly be edited by hand to shade the countries with different colors, as we have done in Fig. 3.4 on p. 94. The advantages of METAFONT in this case lie solely in its portability to all devices for which dvi drivers exist.

The most sophisticated L^AT_EX package that writes METAFONT is probably Thorsten Ohl's feynmf, discussed in detail in Section 6.4 on p. 228. feynmf was designed to create the Feynman diagrams used by high-energy physicists, and is a good example of a very specialized language carefully designed to solve one problem well. Its sibling package, feynmp, uses METAPOST instead of METAFONT. A more general purpose package, mfpic, is discussed in the next section.

3.4.1 The mfpic package

mfpic was designed and implemented by Thomas Leathrum and is now maintained and enhanced by Geoffrey Tobin. It consists of a T_EX package (usable with L^AT_EX) and a supporting library of METAFONT macros. This section describes the version current in November 1996, but the package is still under development and new facilities are planned (such as a METAPOST version).

mfpic's L^AT_EX macros write METAFONT code to an external file and read in the resulting drawing as a character in a special font (this requires two passes of L^AT_EX when the picture dimensions change and a run of METAFONT if the picture contents change at all); each picture is a single METAFONT character. The advantage of this system is that the L^AT_EX user has access to all the power of METAFONT as a drawing package without having to learn a new language. Typesetting of labels and text is handled transparently by the L^AT_EX macros, solving a major problem associated with using METAFONT (METAPOST is an easier system to use for this reason).

One of the main drawbacks of mfpic is that the user must be confident about running METAFONT on the "font" file that is generated and putting the results where T_EX can find them. You also have to worry about METAFONT capacity, which can easily be exceeded at high resolution. We had to limit the picture resolution in this book to 600 dpi in order not to push METAFONT beyond its limits. You must also keep track of whether the pictures have changed and whether the font needs rerunning after each pass of L^AT_EX; this is comparable, however, to running BibT_EX or makeindex as part of the L^AT_EX process and can be hidden in intelligent batch files.

When using mfpic, you must remember that you are dealing with a *font*; this means, first, that you cannot have more than 255 pictures in a single font and second, you must watch your T_EX implementation, which usually has a limit on how many fonts it can load. If you have several thousand pictures to create, you have to open a series of font files and keep a sharp eye on the font memory.

You would not normally expect to read or change the METAFONT font file, since the mfpic macros take care of it. If circumstances demand, however, the METAFONT code is quite readable.

You must surround the section of a document that is to use mfpic with macros to open and close a METAFONT file containing the pictures to be included:

```
\opengraphsfile{file} ... \closegraphsfile
```

You would normally have just one file per document, but you must start a new file if there are more than 255 pictures, and you may prefer to have a separate drawing file for each section.

All picture drawing takes place inside an mfpic environment:

```
\begin{mfpic}[x_scale][y_scale]{x_a}{x_b}{y_a}{y_b} ... \end{mfpic}
```

The parameters give the picture's coordinate system and size:

x_scale *x*-scale of the coordinate system, in multiples of the length \mfpicunit (the default value of which is 1 pt).

y_scale *y*-scale of coordinate system. If no *y_scale* is given, it is the same as the *x_scale*.

x_a Lower bound for *x*-axis coordinates.

x_b Upper bound for *x*-axis coordinates.

y_a Lower bound for *y*-axis coordinates.

y_b Upper bound for *y*-axis coordinates.

The four mandatory parameters set the size of the picture, and the two optional arguments allow it to be scaled. Like L^AT_EX's picture environment, the package does not verify whether elements of the drawing fall within the limits specified; they may well overlap the surrounding text.

mfpic supports a dozen or so primitive drawing commands, a set of object modifiers, and some higher-level macros for drawing functions and performing special transformations. By default each of the simple drawing commands draws a self-contained line or shape, but they can be combined to make more complex shapes:

```
\begin{connect} ... \end{connect}
```

Inside a connect environment you can join objects together; extra line segments are drawn from the endpoint of one object to the start of the next. Even if the objects are closed, like circles, the joining lines are still unfortunately drawn.

The default is to render objects with lines; you can change this with the \setrender command:

```
\setrender{commands}
```

This is a switch; all objects within the (L^AT_EX scope) of this command will be rendered using *commands*. This allows a more succinct notation; for instance, to draw a series of

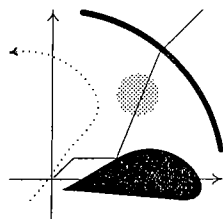
shapes in the same style, the single command

```
\setrender{\gfill}
```

causes all subsequent closed objects to be filled.

All coordinates and distances are normally expressed in TeX point dimensions; angles are expressed in degrees counterclockwise. The behavior of the drawing primitives depends on a set of variables that you can set globally; these are mostly length commands, which are changed with `\setlength`. `mfpic` also has a set of commands that can be used at any point in a drawing to affect subsequent commands.

Many of `mfpic`'s simple drawing commands and some of the modifiers are demonstrated in the following example; the last line has *multiple* modifiers (`\gfill` and `\bclosed`).



```
\usepackage {mfpic}
\begin{mfpic}[1]{-20}{80}{-20}{80}
\pen{.3pt}\axes
\lines{(1,1),(10,10),(30,10),(50,60),(70,80)}
\pen{3pt}
\arc[p]{(0,0),10,80,80}
\pen{1pt}
\shade\circle{(40,40),10}
\arrow\dotted
\curve{(-10,-10),(5,10),(20,40),(-20,60)}
\gfill\bclosed\lines{(60,-10),(5,-5),(30,10)}
\end{mfpic}
```

3-4-1

Basic drawing commands

```
\arc[c]{(x_0,y_0),(x_1,y_1),\theta}
```

Draw an arc with center at (x, y) , starting at point (x_1, y_1) , over θ degrees.

```
\arc[p]{(x,y),\theta_a,\theta_b,r}
```

Draw an arc with center at (x, y) and a radius r , between the angles θ_a and θ_b .

```
\arc[s]{(x_0,y_0),(x_1,y_1),\theta}
```

Draw an arc from the point (x_0, y_0) to the point (x_1, y_1) , and an angle of θ degrees.

```
\arc[t]{(x_0,y_0),(x_1,y_1),(x_2,y_2)}
```

Draw an arc that passes through the three points.

```
\axes
```

Draw axes to span the current picture. An arrowhead of length `\axisheadlen` is drawn at the end of each axis.

```
\circle{(x,y),r}
```

Draw a circle of radius r centered on the point (x, y) .

```
\curve{(x_0,y_0),(x_2,y_2),...,(x_n,y_n)}
```

Draw a Bézier path through the points.

```
\cyclic{(x_0,y_0),(x_1,y_1),...,(x_n,y_n)}
```

Draw a Bézier curve through the points, and join the last and first to create a closed object.

```
\ellipse[\theta]{(x,y),r_x,r_y}
```

Draw a centered ellipse (x, y) with a "width" of r_x and a "height" of r_y . If θ is given, rotate the ellipse by that amount.

```
\lines{(x_0,y_0),(x_1,y_1),...,(x_n,y_n)}
```

Draw line segments between points. `\polylines` is an alias for the `\lines`.

```
\point{(x_0,y_0),(x_1,y_1),...,(x_n,y_n)}
```

Draw small filled circles centered at the points. The diameter of the circles is the length `\pointsize`.

```
\polygon{(x_0,y_0),(x_1,y_1),...,(x_n,y_n)}
```

Draw a closed polygon with vertices at the points.

```
\rect{(x_0,y_0),(x_1,y_1)}
```

Draw a rectangle with corners at (x_0, y_0) and (x_1, y_1) .

```
\turtle{(x,y),(\theta_0,l_0),(\theta_1,l_1),...}
```

Draw a line segment starting at (x, y) , and then draw line segments of length l_i at the angle θ_i with respect to the direction of the previous segment.

```
\sector{(x,y),\theta_1,\theta_2,r}
```

Draw a wedge of a circle from θ_1 to θ_2 centered at (x, y) with radius r .

```
\xmarks{(x_0),(x_1),...,(x_n)}
```

```
\ymarks{(y_0),(y_1),...,(y_n)}
```

Place marks on the x and y axes at the points specified in the list. The length of the marks is `\hashlen`.

Drawing parameters

```
\dashlen (default 4pt)
```

Length of dashes

```
\dashspace (default 4pt)
```

Gap between dashes

```
\hashlen (default 4pt)
```

Length of hash marks on the axes

```
\shadespace (default 1pt)
```

Gap between dots drawn by `\shade`

```
\hatchspace (default 3pt)
```

Gap between lines drawn by `\hatch`

```
\mfpicunit (default 1pt)
```

Basic unit length for pictures

```
\pointsize (default 2pt)
```

Diameter of circle for `\pointmacro`

```
\pointfilled (default true)
```

Boolean variable determining whether circles drawn by `\point` are filled

```
\axisheadlen (default 3pt)
```

Length of arrowhead

```
\axisheadlen (default 5pt)
```

Length of arrowhead on `\axes`

Global modifier commands

`\headshape{ratio}{tension}{filled}`
 Set the shape of arrowheads. *ratio* is the ratio of the width of the arrowhead to its length (default 1); *tension* is the tension of the Bézier curves used to draw the arrowhead (default 1); and *filled* is a Boolean variable that determines whether arrowheads are filled (default true).

`\pen{pensize}`
 Set the width of the drawing pen (the default is 5pt).

`\dashlineset`
 Set `\dashlen` and `\dashspace` to 4pt.

`\dotlineset`
 Set `\dashlen` to 1pt and `\dashspace` to 2pt.

`\darker shade`
 Make shading denser by multiplying `\shadespace` by a factor of 5/6.

`\lighter shade`
 Make shading lighter by multiplying `\shadespace` by a factor of 6/5.

Figure modifier commands

Any object drawn with one of the simple commands, or a composite object created using `connect`, can be modified with one or more *prefix* commands. These apply just to the next object and are mostly used for the closed objects like circles, ellipses and so on.

`\arrow[length]{rθ}[bdistance]`
 Draw an arrowhead at the last specified point of an object. The three optional parameters let you change the shape: the length of the arrowhead is set by *length* (default is `\headlen`), the angle by θ , and the distance from the end point by *distance* (this lets you draw double arrowheads). The optional parameters can occur in any order but must be preceded by the special key characters.

`\bclosed`, `\lclosed`, `\sclosed`, `\cbclosed`
 Close an open object by joining the endpoints; `\lclosed` draws a straight line, `\bclosed` draws a Bézier curve, `\sclosed` draws a smooth curve, and `\cbclosed` draws a cubic B-spline.

`\dotted[dash][gap]`
 Draw object lines dotted or dashed; the length of dashes is set by `\dashlen`, but can be overridden by *dash*; similarly, the space between dashes can be set with *gap* (the default being `\dashspace`).

`\draw`
 Draw an outline for an object; this is needed if you want to draw *and* fill a object.

`\gclear`
 Erase everything inside a closed object.

`\gfill`
 Fill a closed object.

`\hatch[distance]`
 Draw cross-hatched shading to fill a closed object. The default gap between lines is `\hatchspace`, but can be overridden with the optional *distance* parameter. Various other hatching commands are defined.

`\reverse`
 Reverse the orientation of an object; this is useful in changing the sequence of points inside a

`connect` environment, so that a different endpoint is used.

`\rotatepath{(x,y)θ}`
 Rotate an object by θ degrees around (x,y) .

`\shade[distance]`
 Fill a closed object with dots; the default gap between dots is `\shadespace`, but can be overridden with the optional *distance* parameter.

The filling commands do *not* draw an outline for the object, and must come first if there is more than one modifier.

Analytical curves and functions

A common requirement in scientific data analysis is plotting functions and parametric curves; METAFONT is very suitable for this sort of work, and mfpic has five general commands. Each of them has an optional first parameter that can have the value *s*, meaning that the plot is drawn with a smooth Bézier curve, or *p*, meaning that it is drawn with straight lines. The default is *s* except for `\btwnfc` and `\plrregion`.

Each command has a mandatory parameter to express the minimum and maximum values for the function and the step. The other parameters are function(s) passed to METAFONT to evaluate.

`\function[type]{minimum,maximum,step}{expr}`
 Plot *expr*.

`\parafcn[type]{minimumi,maximumi,step}{(expr1,expr2)}`
 Plot the parametric path $(x(t), y(t)) = (expr_1, expr_2)$.

`\plrfcn[type]{minimum,maximum,step}{expr}`
 Plot the polar function *expr*.

`\btwnfc[type]{minimum,maximum,step}{expr1}{expr2}`
 Draw the region between functions $f(x) = expr_1$, and $g(x) = expr_2$, where the region is bounded also by the vertical lines at *minimum* and *maximum*. By default the function is drawn as a set of line segments.

`\plrregion[type]{minimum,maximum,step}{expr}`
 Plot the polar region determined by $r = expr(\theta)$. The θ values are angles (measured in *degrees*) between *minimum* and *maximum*. By default the function is drawn as a set of line segments.

Annotating drawings

Two commands for annotating drawings are processed entirely in L^AT_EX and pass nothing to the METAFONT file:

`\tlabel[pos](x,y){text}`

This labels a graph with *text* at (x,y) . By default the text is positioned with its lower left hand corner at (x,y) , but the optional position parameter *pos* can also be used. *pos* is similar to the placing parameters for boxes in the `picture` environment and specifies the relative placement of the label with respect to the reference point: the first character is for the *vertical* placement (t for top, c for center and b for bottom) and the second for the *horizontal* placement (l for left, c for center and r for right).

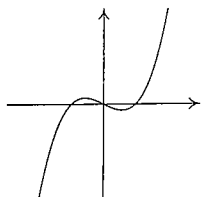
`\tcaption[maxwidth,linewidth]{text}`

This centers a caption underneath your drawing. You can determine how a long caption is broken into lines with the optional parameters; if the caption is wider than *maxwidth* × the width of the drawing, then the text is typeset in a paragraph with a line length of *linewidth* × the width of the drawing. The default is [1.2,1.0].

Examples

We conclude our description of `mfpic` with some examples; these are designed not to show all the features of the package but rather to demonstrate some typical applications.

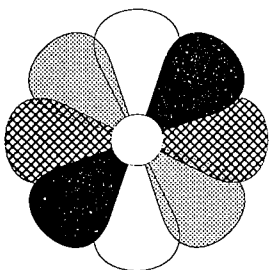
A simple example of function plotting shows the clarity of `mfpic` markup:



```
\usepackage {mfpic}
\begin{mfpic}[15]{-3}{3}{-3}{3}
\axes
\function{-2,2,0.1}{((x**3)-x)/2}
\end{mfpic}
```

3-4-2

A repetitive drawing can take advantage of $\text{\LaTeX}$'s programmability; below we define a macro to draw a flower petal and then call it several times with different modifiers. This example uses an affine transformation (see Section 3.4.1) to allow cumulative rotation.

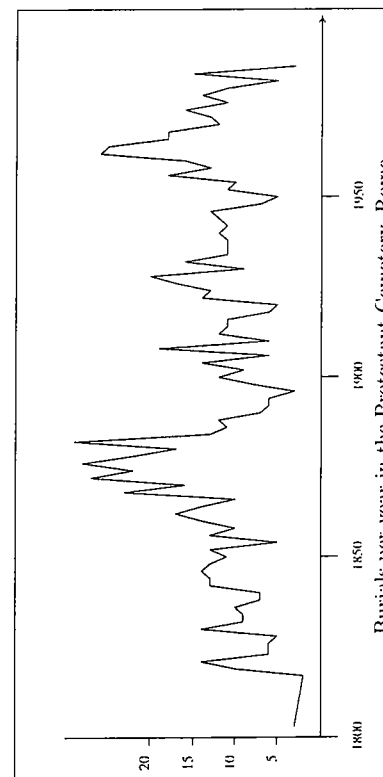


```
\usepackage {mfpic}
\begin{mfpic}[1.2]{-50}{50}{-50}{50}
\pen{.3pt}
\newcommand{\petal}%
{ \bclosed\lines{(30,10),(0,0),(10,30)} }
\newcommand{\halfflower}%
{ \begin{coords}
\gfill\petal
\turn{45}\petal
\turn{45}\shade\draw\petal
\turn{45}\hatch\draw\petal
\end{coords}
}
\halfflower
\turn{180}
\halfflower
\gclear\draw\circle{(0,0),10}
\end{mfpic}
```

3-4-3

Our next picture, a simple data plot, demonstrates the use of labels; again, we use a $\text{\LaTeX}$ macro to parameterize how these are drawn. Since the values in the *x* direction have

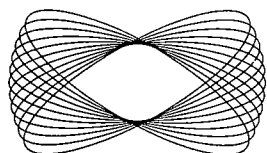
a much greater range than those in the *y* direction, the optional *y* scaling of the `mfpic` environment is convenient. This also shows how a drawing as a whole can be manipulated with the standard rotation macros.



3-4-4

```
\usepackage {graphics,mfpic}
\rotatebox{90}{\fbox{%
\begin{mfpic}[1.7][4]{-10}{200}{-5}{35}
\newcommand{\XmyLabel}[2]{%
\tlabel[bc]{#1,-5}{\scriptsize#2}}
\lines{(#1,0),(#1,-2)}%
}
\newcommand{\YmyLabel}[1]{%
\tlabel[br]{-5,#1}{\scriptsize#1}}
\lines{(0,#1),(-2,#1)}%
}
\XmyLabel{0}{1800}\XmyLabel{50}{1850}
\XmyLabel{100}{1900}\XmyLabel{150}{1950}
\YmyLabel{5}\YmyLabel{10}
\YmyLabel{15}\YmyLabel{20}
\arrow\lines{(0,30),(0,0),(200,0)}
\lines{(03,3),(17,2),(19,10),(21,14),
(23,6),(26,6),(28,5),(30,14),(32,9),
(34,9),(36,10),(38,7),(40,7),
(42,13),(44,13),(46,14),(48,13),
(50,11),(52,13),(54,5),(56,13),
(58,10),(60,14),(62,17),(64,14),
(66,10),(68,23),(70,16),(72,27),
(74,22),(76,28),(78,22),(80,17),
(82,29),(84,13),(86,11),(88,12),
(90,7),(92,6),(94,6),(96,3),(98,8),
(100,12),(102,9),(104,14),(106,6),
(108,19),(110,6),(112,12),(114,11),
(116,11),(118,6),(120,5),(122,14),
(124,13),(126,17),(128,20),(130,9),
(132,16),(134,11),(136,11),(138,11),
(140,12),(142,11),(144,12),(146,13),
(148,7),(150,5),(152,11),(154,10),
(156,18),(158,13),(160,16),(162,26),
(164,25),(166,18),(168,18),(170,12),
(172,13),(174,16),(176,11),(178,14),
(180,11),(182,5),(184,15),(186,3)}
\tcaption[.8,.75]{Burials per year in the
Protestant Cemetery, Rome.}
\end{mfpic}}}
```

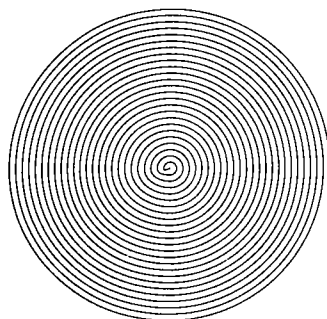
A useful tool in these circumstances is the standard L^AT_EX ifthen package, which lets us perform routine looping:



```
\usepackage {mfpic,ifthen}
\newcounter{mfcntA}
\begin{mfpic}[1.5]{-30}{40}{-25}{25}
\setcounter{mfcntA}{-30}
\whiledo{\themfcntA<31}{%
\ellipse[\themfcntA]{(0,0),40,12}
\addtocounter{mfcntA}{6}}%
\end{mfpic}
```

3-4-5

Another use of looping is to draw a spiral, using the same technique as on p. 56; this is not, of course, the “correct” way to draw such a shape, which might be better expressed using the \parafcn command:



```
\usepackage {mfpic,ifthen}
\begin{mfpic}[1.5]{-40}{50}{-48}{50}
\newcounter{mfcntA}
\newcounter{mfcntB}
\setcounter{mfcntA}{50}
\setcounter{mfcntB}{48}
\whiledo{\themfcntA>1}{%
\expandafter\curve{(\themfcntA,0),
(0,\themfcntA),
(-\themfcntA,0),
(0,-\themfcntB),
(\themfcntB,0)}
\addtocounter{mfcntA}{-2}%
\addtocounter{mfcntB}{-2}%
}
\end{mfpic}
```

3-4-6

Going further

mfpic defines a set of affine transformations (rotation, reflection, translation, scaling, and skewing) described in detail in the package documentation. We used one of these (\turn) in the “flower” example above, to keep turning the current drawing state. All the transformations are cumulative, but their effect can be conveniently limited with the coords environment (as demonstrated in the “petal” example above).

Users who want to build new facilities can use commands to write METAFONT directly to the output file, to store and use METAFONT paths directly, and to develop sophisticated fill patterns for objects.

3.5 mftoeeps: a direct link between METAFONT and PostScript

Sometimes you'd like to clean up or enhance the output from the META programs using one of the excellent interactive drawing packages. One way to do this is to use METAPOST and edit the result using Adobe Illustrator (the latest versions of which can cope with almost arbitrary PostScript), but a more specific result can be achieved with an elegant package called mftoeeps, written by Bogusław Jackowski, Piotr Pianowski and Marek Ryćko (Jackowski, 1995). This METAFONT program intercepts drawing commands and rewrites them to the log file as Encapsulated PostScript; utility programs are provided to extract this code, and the result can be directly edited by Corel Draw and Adobe Illustrator.

This system handles normal METAFONT code for the most part; to activate it, you must load the library of METAFONT code, mftoeeps.mf, and set the variable yesEPS to 1. Normal draw and fill commands are then replaced by special variants shown in the following simple example:

```
input mftoeeps;
EPS_mode_setup;
beginchar("A",2cm#,1cm#, 0cm#);
set_BB 0,-d,w,h;
write_preamble "test";
draw_C (0,0)..(1cm,1cm)..(2cm,0);
write_postamble;
endchar;
end.
```

The extra commands are as follows:

EPS_mode_setup

Use instead of mode_setup, and set to 1 to generate EPS generate.

write_preamble name

Initialize the process of writing PostScript to the log file (destined for a file name).

write_postamble

Stop writing PostScript to the log file.

find_BB

Followed by a list of METAFONT paths, work out the bounding box and write it to the PostScript code.

set_BB

Write out the following four numbers or two comma-separated number pairs to the PostScript code as a bounding box. Either this or find_BB should be used before write_preamble.

draw_C paths

Write out the paths (a comma-separated list) to the PostScript file as Illustrator polygon objects.

fill_C paths

Write out the paths (a comma-separated list) to the PostScript file as Illustrator filled areas.

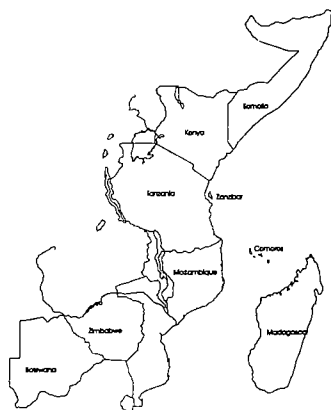


Figure 3.3: AutoCAD map converted to METAFONT.

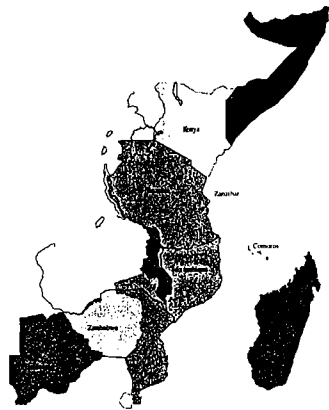


Figure 3.4: METAFONT drawing enhanced using Corel Draw.

`fix_line_width number`

Set the line thickness (default 0.4pt) for objects created by `draw_C`.

`fix_fill_cmyk four numbers`

Set the fill color of objects created by `fill_C`, using CMYK colors.

`fix_drawn_cmyk four numbers`

Set the line color of objects created by `draw_C`, using CMYK colors.

Fig. 3.3 shows a map drawn in AutoCAD and converted to METAFONT; Fig. 3.4 shows the same map converted to Adobe Illustrator format and colored using Corel Draw. While it would be possible to do the coloring with METAPOST, it is often more convenient to work on a map like this in an interactive program.

CHAPTER 4

Harnessing PostScript inside L^AT_EX: the PSTricks package

As we saw in Chapter 1, one way of drawing graphics with L^AT_EX is to embed low-level picture drawing primitives for the target device into L^AT_EX macros, so that full typesetting information is available and we can work in a familiar macro programming environment. When the target device is something as rich as the full PostScript language, this can result in a very powerful system. While many macro packages have implemented access to some parts of PostScript for this purpose the most complete is undoubtedly PSTricks. In this chapter, we survey its capabilities and demonstrate some of the power that results from combining L^AT_EX and PostScript.

We do not attempt to describe absolutely every PSTricks macro, nor do we give examples of all the possible combinations and tricks, as this would require a large book of its own. We have, however, tried to describe and give examples of all the important features.

Because there are a great many commands and parameters in PSTricks, we provide summary descriptions of most of them at the end of this chapter (Section 4.13 on p. 154).

4.1 The components of PSTricks

PSTricks, written by Timothy van Zandt, consists of a core of picture-drawing primitives implemented by `\specials` that pass PostScript to a driver. It also contains a set of higher-

Table 4.1: PSTricks supporting packages.

<i>Package</i>	<i>Function</i>
pst-3d	Three-dimensional drawing
pst-char	Stroking and filling character paths
pst-coil	Coil and zigzag objects
pst-eps	Export of objects direct to EPS files
pst-fill	Extra <code>\psboxfill</code> macro
pst-grad	Gradient color fills
pst-node	Placing and joining nodes
pst-plot	Data plotting
pst-text	Typesetting text on a path
pst-tree	Tree macros

level macros for particular applications. With it you can:

- draw lines, polygons, circles, and curves;
- place and manipulate T_EX text;
- plot data with complicated labeled axes;
- draw nodes and connectors (including trees);
- color lines and fill objects;
- define new graphical commands.

This is an extremely powerful package, and its facilities can take some time to understand. It is documented in van Zandt (1993a), and its implementation is described in van Zandt and Girou (1994). Girou (1994) provides an excellent demonstration of its capabilities, and we are grateful to Denis Girou for permission to reproduce some of his examples here.

The package relies on the ability of a dvi driver to pass through literal PostScript code and know that it will interact with the T_EX text in a controlled way. The dvips driver is the reference implementation, but the package works with many other drivers as well. The PSTricks installation guide explains what functionality the driver has to provide and how to set it up; see also Section 4.12.

PSTricks is not a tool for drawing just one type of diagram well, like many of the other packages described in this book. It is a programming environment for as close a combination of T_EX and PostScript as is possible with existing software; its strength is its modularity, extensibility, and ability to access all the power of PostScript.

Most of PSTricks is loaded in a single package, but some more complicated facilities need an extra file to be loaded with `\usepackage`; the packages are listed in Table 4.1. In the descriptions that follow, we do *not* normally indicate which file needs to be loaded for a particular function, since it is usually obvious. However, each example of code shows

the necessary `\usepackage` commands; we often load `pstcol` instead of just `pstricks`, for pictures where color is used.

New modules for PSTricks may be released, as the package is still under development; the material in this chapter was all tested with the release publicly available at the end of 1996.

4.2 Basic PSTricks concepts

Most of the commands in PSTricks draw some kind of object at specified coordinates, which are relative to the current point in T_EX. The objects do not usually take up any space in T_EX terms; they can either be mixed in with normal commands or used in a picture environment to construct a whole drawing. The special L^AT_EX environment:

```
\begin{pspicture}*[settings][baseline](x0,y0)(x1,y1)... \end{pspicture}
```

is normally used for pictures; space is reserved in T_EX for a rectangle with corners at (*x*₀, *y*₀) and (*x*₁, *y*₁). As with other PSTricks commands discussed later, the first coordinate pair (*x*₀, *y*₀) is optional and defaults to (0.0). The star form of the environment clips graphic objects that appear outside the frame. By default, the baseline is set at the bottom of the box, but the optional argument sets the baseline a fraction *baseline* from the bottom.

4.2.1 Commands and arguments

Almost all the commands have the same (complex) structure. They need some or all of the following arguments, each of which has its consistent delimiters; obligatory parameters are surrounded by braces, like {*arg*}; optional settings are in square brackets, like [*par*=*val*1, ...]; coordinates are in parentheses, like (*x*, *y*).

Because PSTricks macros can get very complex with lots of different parameters, the summary macro descriptions here (mostly in the tables at the end of this chapter) have a gray shading behind the optional parameters.

The general syntactical form of most commands is:

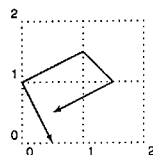
```
\command*[settings]{arrows/parameters}(coordinates)
```

The star form of the command generally means that the object being drawn is to be solid rather than an outline; the optional *settings* in square brackets consist of a set of key value pairs which override, for the current object, PSTricks's drawing defaults as discussed below.

Many commands that draw lines need an argument like a rotation angle or have an argument specifying whether, and how, arrowheads are to be drawn at either end; these arguments come in braces after the optional settings.

Most objects require one or more coordinates; these consist of two numbers in parentheses separated by a comma. Multiple coordinates each have their own set of parentheses.

A simple complete example follows in which a grid has been superimposed to show the coordinate system (the command `\showgrid` is explained on p. 142):



```
\usepackage {pstcol}
\begin{pspicture}(0,0)(2,2)\showgrid
\psline[linewidth=1pt,linestyle=gray]
{<->}(0.5,0)(0,1)(1,1.5)(1.5,1){.5,.5}
\end{pspicture}
```

4-2-1

4.2.2 Setting graphics parameters

PSTricks uses a notation similar to that introduced in Section 2.2.2 for the `graphicx` package, i.e., ‘key value’ pairs. This is used for setting a large number of graphical parameters that apply to almost all objects. These parameters can be set in two ways:

- On a per-object basis, with the optional ‘key value’ parameters in square brackets separated by commas; in this case the effect is local to the object with no further grouping needed;
- Within the scope of the current environment, using the `\psset` command. The syntax is:

```
\psset{par1=value1,par2=value2,...}
```

Extra spaces are allowed only after the comma that separates *par=value* pairs. The graphics parameters that can be set for objects are listed in Table 4.5 starting on p. 156; the first group can be applied to more or less anything, but the others apply only to particular commands.

4.2.3 Coordinates and units

By default the coordinate system is in units of 1 cm in the *x* and *y* directions, but that can be easily overridden as we see in a moment. Coordinate pairs can be given not only in the default units, but also in any T_EX dimensions. This applies to all “quantity” settings, so `linewidth=.5` means 0.5 of whatever the current unit is, but `linewidth=2pt` is an absolute size. You can use the default units when setting non-PSTricks dimensions as well, using the commands

```
\pssetlength{cmd}{dim}
\psaddtolength{cmd}{dim}
```

where *cmd* is a length and *dim* is a number with optional unit. To set standard L^AT_EX dimension parameters using PSTricks default units, use `\pssetlength` and `\psaddtolength` instead of L^AT_EX’s `\setlength` and `\addtolength` commands.

The command `\SpecialCoor` sets up PSTricks so that you can use polar coordinates in the form (ra) , where *r* is the radius (a dimension) and *a* is the angle. You can still use Cartesian coordinates.

The unit parameter actually sets three parameters, `xunit`, `yunit`, and `runit`. These are the default units for *x* coordinates, *y* coordinates, and all other coordinates, respectively. By setting these independently, you can scale the *x* and *y* dimensions in Cartesian coordinates separately.

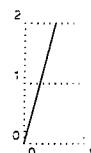
The effect of changing the default units can be seen by comparing the following two examples (the grid remains the same):

4-2-2



```
\usepackage {pstcol}
\begin{pspicture}(0,0)(1,1)\showgrid
\psline(0,0)(1,1)
\end{pspicture}
```

4-2-3



```
\usepackage {pstcol}
\begin{pspicture}(0,0)(1,2)\showgrid
\psset{xunit=.5cm,yunit=2cm}
\psline(0,0)(1,1)
\end{pspicture}
```

A similar effect can be achieved using the standard L^AT_EX graphics package and its `\resizebox` macro:

4-2-4



```
\usepackage {pstcol}
\resizebox{.5cm}{2cm}{%
\begin{pspicture}(1,1)
\psline(0,0)(1,1)
\end{pspicture}
}
```

Since some PSTricks defaults are given in absolute sizes (not in terms of the units) and are thus unaffected by changes of unit, the `\resizebox` scales *everything* more reliably if that is what is desired. For instance, it is probably a better tool for resizing an already prepared drawing for a new page size.

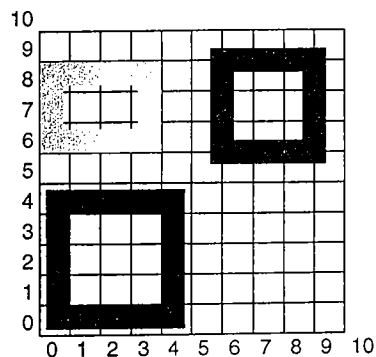
Angles, in polar coordinates and other arguments, should be a number giving the angle in degrees, by default. You can also change the units used for angles with the command `\degrees[num]`, where *num* is the number of units in a circle. Thus

```
\degrees[100]
```

could be used when making a pie chart where the data values are supplied as percentages. The command `\radians` is short for `\degrees[6.28319]`.

It is important to realize in drawing objects that lines can be drawn within, outside, or straddling the line between two points. The `dimen` parameter sets this, and it is usually

sensible to set it to 'middle'; the following example shows the effect of drawing the same square with the three variants of `dimen`.



```
\usepackage {pstcol}
\psset{unit=.5cm}
\begin{pspicture}(10,10)
\psset{subgriddiv=0,linewidth=.8}
\psgrid
\psset{dimen=inner}
\psframe
  (1,1)(4,4)
\psframe[dimen=middle,linewidth=red]
  (6,6)(9,9)
\psframe[dimen=outer,linewidth=yellow]
  (0,6)(4,9)
\end{pspicture}
```

4-2-5

4.2.4 Coloring objects

PSTricks comes with its own collection of color macros that provide a basic set of colors and let you define new color names in terms of RGB, CMYK or HSB models (see Chapter 9); however, we recommend that L^AT_EX users stick with the color package described in Chapter 2. The standard color package and PSTricks can be used together by loading the package `pstcol`, which comes with the color and graphics bundle. The `pst-grad` package adds facilities for gradations of color in filled objects.

The following colors are predefined: *black*, *darkgray*, *gray*, *lightgray*, *white*, *red*, *green*, *blue*, *cyan*, *magenta*, and *yellow*.

4.3 The graphic objects

We list in Section 4.13.1 on p. 154 the basic objects that PSTricks can draw; with the exception of the text-framing commands, they all take up no T_EX space, and so should be used inside a `pspicture` environment when creating a free-standing picture. In almost every case in which a pair of coordinates must be supplied, the first can be omitted and defaults to (0, 0). Lines and open curves can optionally be terminated with various symbols; to do this use the `arrows` parameter (see examples and Table 4.3 on p. 103).

To use any graphic object as a clipping path, the command

```
\psclip{graphics} ... \endpsclip
```

cuts off anything between `\psclip` and `\endpsclip` that falls outside the boundary of

4.3 The graphic objects

whatever is drawn by *graphics* (do not try to be too clever, just use the basic graphic objects), as shown in the following example:

4-3-1



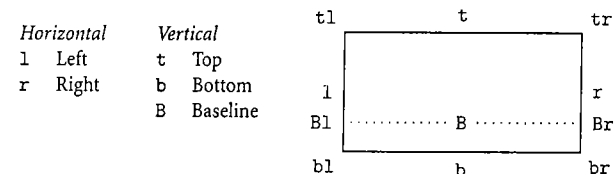
```
\usepackage {pstcol}
\begin{pspicture}(-1,0)(2,2)
\psclip{\psdiamond(1.5,0.2)(2.0,0.5)}
\Huge Scary Monsters
\endpsclip
\end{pspicture}
```

PSTricks has powerful commands for positioning (and rotating, if necessary) something, including normal L^AT_EX material; they are analogous to L^AT_EX's basic `\put` command in picture mode. The more common command is:

```
\rput*[settings]([refpoint])[angle](x_0,y_0){stuff}
```

The star form puts a `\psframebox` around *stuff*, which effectively blocks out anything underneath. *stuff* is rotated by *angle* if the argument is present; preceding *angle* by `*` has the effect of undoing all rotations in outer calls to `\rput`. It makes it easy to place text labels consistently in the desired orientation. Since rotations are often in steps of 90°, you can use the letter abbreviations in Table 4.2 on the next page.

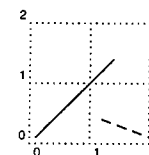
The argument *refpoint* describes the reference point of *stuff*, and it is this reference point that is placed at (x, y). By default, it is the center of the box. This can be changed by setting *refpoint* to one or two of the letters in the following diagram. This shows the reference point represented by the various combinations (the dotted line is the baseline).



Basically the same conventions are used by the `graphicx` package when specifying the rotation of boxes, as shown in Fig. 2.2 on p. 46.

It is important to realize that within the argument the coordinate system is locally modified so that (0, 0) refers to the coordinate of the `\rput`, as shown in the following example:

4-3-2



```
\usepackage {pstcol}
\begin{pspicture}(0,0)(2.2,2)\showgrid
\rput(.1,.1){\psline(0,0)(1.3,1.3)}
\psset{linestyle=dashed}
\rput{-20}(1.2,.4){\psline(0,0)(1,0)}
\end{pspicture}
```

Table 4.2: PSTricks letter abbreviations for common angles.

Letter		in degrees	Letter		in degrees
U	Up	0	N	North	*0
L	Left	90	W	West	*90
D	Down	180	S	South	*180
R	Right	270	E	East	*270
r	right	0	ur	up-right	45
u	up	90	ul	up-left	135
l	left	180	dl	down-left	225
d	down	270	dr	down-right	315

Because it is very often desired to put a label next to some object, a variant of `\rput` called `\uput` is provided:

```
\uput*[settings](labelsep)(refangle)(rotation)(x,y){stuff}
```

This places *stuff* in the direction *refangle* at a distance of *labelsep* from (*x*,*y*). If *labelsep* is not specified the current value of `\labelsep` (defaults to 5pt in standard L^AT_EX) is used. Since angles are often in steps of 45°, letter abbreviations are provided for common cases (Table 4.2).

Finally, there is a special macro to draw circled labels in one go:

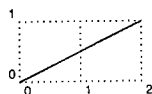
```
\cput*[settings](angle)(x,y){stuff}
```

This has the effect of an `\rput` and a `\pscirclebox` together. The reference point is always the center of the box, and the optional *settings* can be used to change the way the circle is drawn.

4.4 Examples of basic graphic objects

The following examples demonstrate some of the PSTricks building blocks and the use of the graphics parameters. We must remember that all these simple objects take up no space; the surrounding `pspicture` defines the space T_EX is to leave, but within that, we are drawing entirely by coordinates. A number of these pictures are reproduced in Color Plate II.

First, a simple line; the starting point defaults to (0,0) and all parameters have their default values.



```
\usepackage {pstcol}
\begin{pspicture}(0,0)(2,1)\showgrid
\psline(2,1)
\end{pspicture}
```

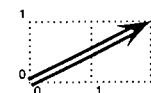
4-4-1

Table 4.3: PSTricks line terminators.

Values and examples						
-						None
<->		->		<-		Arrowheads
>-<		-<		>-		Reverse arrowheads
<<->		->>		<<-		Double arrowheads
>>-<<		-<<		>>-		Double reverse arrowheads
-		-		-		T-bars, flush to endpoints
*		- *		* -		T-bars, centered on endpoints
[-]		-]		[-		Square brackets
(-)		-)		(-		Parentheses
o - o		- o		o -		Circles, centered on endpoints
* - *		- *		* -		Disks, centered on endpoints
oo - oo		- oo		oo -		Circles, flush to endpoints
** - **		- **		** -		Disks, flush to endpoints
c - c		- c		c -		Extended, rounded ends
cc - cc		- cc		cc -		Flush round ends
C - C		- C		C -		Extended, square ends
<->		->		<-		T-bars and arrowheads
<*->		-> *		<*-		T-bars and arrowheads, flush

There is an optional parameter to `\psline` to set arrow-like terminators on lines (the possibilities are listed in Table 4.3), and we can vary the line by using the keys `linewidth`, `doubleline`, and `doublesep`:

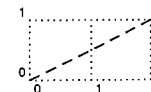
4-4-2



```
\usepackage {pstcol}
\begin{pspicture}(0,0)(2,1)\showgrid
\psline[linewidth=0.6mm,doubleline=true,
doublesep=0.5mm]{->}(2,1)
\end{pspicture}
```

We can also change the way line is drawn by using the `linestyle` parameter (see Table 4.5 on p. 156 for other possible values):

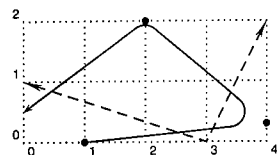
4-4-3



```
\usepackage {pstcol}
\begin{pspicture}(0,0)(2,1)\showgrid
\psline[linestyle=dashed](2,1)
\end{pspicture}
```

The `\psline` can be given as many coordinate parameters as you wish, as the following example shows. Here we draw two lines; for one, we have changed the color and the arrowhead length and altered the style to dashed, and for the other we have used `linearc` to set the style of angles through which the line turns. The `showpoints` parameter makes

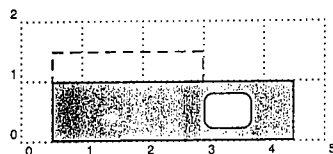
PSTricks print a dot at each of the coordinate points, although the line does not necessarily pass through them.



```
\usepackage {pstcol}
\begin{pspicture}(0,0)(4,2)\showgrid
\psline[linearc=0.25,showpoints=true]%
{->}(1,0)(4,0.3)(2,2)(0,0.5)
\psline[linestyle=dashed,
linecolor=blue,
arrowlength=3]{<->}(0,1)(3,0)(4,2)
\end{pspicture}
```

4-4-4

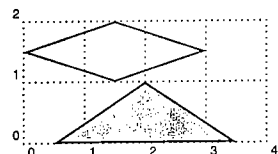
The simplest way of drawing boxes is with `\psframe`; this example shows how to make a “hole” in a solid object by superimposing a smaller solid white object.



```
\usepackage {pstcol}
\definecolor{pink}{rgb}{.75,.8}
\begin{pspicture}(0,0)(4.5,2)\showgrid
\psframe[linestyle=dashed]
(0.5,0.5)(3,1.5)
\psframe[fillstyle=solid,fillcolor=pink]
(.5,0)(4.5,1)
\psframe[fillstyle=solid,fillcolor=white,
framearc=0.5](3,0.2)(3.8,0.8)
\end{pspicture}
```

4-4-5

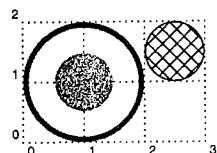
PSTricks has a general `\pspolygon` macro, but some common shapes are predefined:



```
\usepackage {pstcol}
\begin{pspicture}(0,0)(4,2)\showgrid
\psdiamond(1.5,1.5)(1.5,0.5)
\pstriangle[fillstyle=solid,
fillcolor=yellow](2,0)(3,1)
\end{pspicture}
```

4-4-6

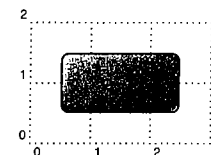
Objects can be filled with a solid color or a pattern:



```
\usepackage {pstcol}
\definecolor{lightblue}{rgb}{.68,.85,.9}
\begin{pspicture}(0,0)(3,2)\showgrid
\pscircle[linewidth=1mm,linecolor=green]
(1,1){1}
\pscircle[linestyle=none,fillstyle=solid,
fillcolor=lightblue](1,1){0.5}
\pscircle[fillstyle=crosshatch](2.5,1.5)
{0.5}
\end{pspicture}
```

4-4-7

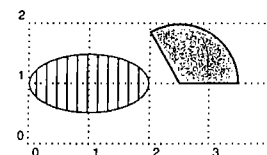
Loading the optional package `pst-grad` provides an extension to the solid color fill; given the names of two colors, the fill color changes across the extent of the object from one to the other.



4-4-8

```
\usepackage {pstcol,pst-grad}
\begin{pspicture}(0,0)(3,2)\showgrid
\psframe[framearc=0.25,
fillstyle=gradient,
gradangle=30,
gradbegin=yellow,
gradend=red]
(0.5,0.5)(2.5,1.5)
\end{pspicture}
```

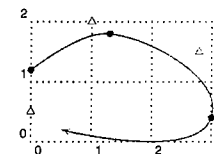
The next example shows more variety in filling, with varying angle and color of cross-hatching:



4-4-9

```
\usepackage {pstcol}
\begin{pspicture}(0,0)(4,2)\showgrid
\psellipse[linecolor=green,
fillstyle=vlines,
hatchangle=0]
(1,1)(1,0.5)
\pswedge[fillstyle=solid,
fillcolor=lightgray]
(2.5,1){1}{0}{120}
\end{pspicture}
```

There are a variety of commands for joining lists of coordinates. The next example illustrates two of them: `\psdots` simply places a chosen symbol at each point (the large choice of styles is shown in Table 4.4 on the following page), and `\pscurve` fits a curve through the points (here we use `showpoints` again to see the coordinates).



4-4-10

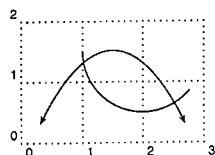
```
\usepackage {pstcol}
\begin{pspicture}(0,0)(3,2)\showgrid
\psdots[linecolor=blue,dotstyle=triangle,
dotscale=2](0,0.5)(1,2)(2.8,1.5)
\pscurve[linecolor=cyan,showpoints=true]
{->}%
(0,1.2)(1.3,1.8)(3,0.4)(0.5,0.2)
\end{pspicture}
```

`\parabola` draws a parabola that starts at one coordinate and whose maximum extent is the other, while `\psarc` does not use coordinates, but draws a portion of a circle for which

Table 4.4: PSTricks dot styles.

Style	Effect
*	• • • • • • •
+	+ + + + + + +
o	o o o o o o o
x	x x x x x x x
asterisk	* * * * * * *
diamond*	◊ ◊ ◊ ◊ ◊ ◊ ◊
diamond	◆ ◆ ◆ ◆ ◆ ◆ ◆
oplus	⊕ ⊕ ⊕ ⊕ ⊕ ⊕ ⊕
otimes	⊗ ⊗ ⊗ ⊗ ⊗ ⊗ ⊗
pentagon*	◐ ◐ ◐ ◐ ◐ ◐ ◐
pentagon	◑ ◑ ◑ ◑ ◑ ◑ ◑
square*	■ ■ ■ ■ ■ ■ ■
square	◻ ◻ ◻ ◻ ◻ ◻ ◻
triangle*	▲ ▲ ▲ ▲ ▲ ▲ ▲
triangle	△ △ △ △ △ △ △

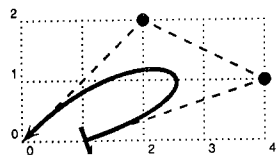
we supply the center point, radius, and the starting and ending angles.



```
\usepackage {pstcol}
\begin{pspicture}(0,0)(3,2)\showgrid
\parabola[linecolor=red]{<->%
(0.3,0.3)(1.5,1.5)
\psarc(2,1.5){1}{180}{320}
\end{pspicture}
```

4-4-11

For those who like fine control of their curves, the `\psbezier` macro directly harnesses the Bézier curve fitting in PostScript. We supply a set of four control points (using `showpoints` to display them), and PostScript fits a smooth curve.



```
\usepackage {pstcol}
\begin{pspicture}(0,0)(4,2)\showgrid
\psbezier[linewidth=0.8mm,linecolor=red,
showpoints=true]{|->%
(1,0)(4,1)(2,2)(0,0)
\end{pspicture}
```

4-4-12

4.5 Mixing text and graphics

With enclosed text the situation is rather different; now the size of objects is determined by the size of the text they enclose. Here T_EX is aware of the space used, so successive objects are placed as if they were letters, and coordinates are not used. Again, a number of the following pictures are reproduced in Color Plate II.

A commonly used command is `\psframebox`, which puts a frame around text:

4-5-1

The Buck Stops Here

```
\usepackage {pstricks}
\psframebox{The Buck Stops Here}
```

When using white text on a colored or black background in a frame, bold characters are usually necessary:

4-5-2

Beware of The Dog

```
\usepackage {pstricks}
\psframebox[fillstyle=solid,
fillcolor=black]
{\bfseries\color{white}%
\LARGE Beware of The Dog}
```

Nesting of `\psframebox` commands is possible in the normal way:

```
\usepackage {pstcol}
\psframebox{The dragon}
\psframebox{\psframebox[linecolor=green]{ate}
\psframebox[linecolor=blue]
{\psframebox[linecolor=red]{the women} and
\psframebox[linecolor=red]{children}%
}}
```

4-5-3

The dragon ate the women and children

Several variations on the `\psframebox` command provide commonly used effects. A nice one is `\psshadowbox`:

4-5-4

Chapter 1
We go to sea

```
\usepackage {pstcol}
\psshadowbox[fillstyle=solid,
fillcolor=yellow]
{\color{red}
\begin{tabular}{c}
Chapter 1\We go to sea
\end{tabular}}
```

Some features are provided by their own commands, but others use special parameters for `\psframebox`; a double circle around text is made using the `doubleline` parameter:



```
\usepackage {pstricks}
\pscirclebox[doubleline=true]
{\bfseries STOP!}
```

4-5-5

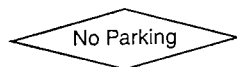
but a double box has its own `\psdblframebox`:



```
\usepackage {pstcol}
\psdblframebox[linecolor=green,
linestyle=dotted,linewidth=3pt,
dotstyle=**]
{\color{red}All Hail Caesar!}
```

4-5-6

Other shapes available for framing are diamonds (`\psdiabox`), triangles (`\pstribox`) and ovals (`\psovalbox`):



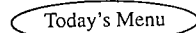
```
\usepackage {pstricks}
\psdiabox{\sfamily No Parking}
```

4-5-7



```
\usepackage {pstcol,pst-grad}
\pstribox[shadow=true,fillstyle=gradient,
gradbegin=green,gradend=red]
{\color{white}\Huge
 $\frac{\alpha}{\Omega}$ }
```

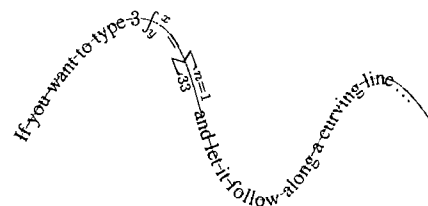
4-5-8



```
\usepackage {pstcol,pst-grad}
\psovalbox[linecolor=red]
{\color{blue}Today's Menu}
```

4-5-9

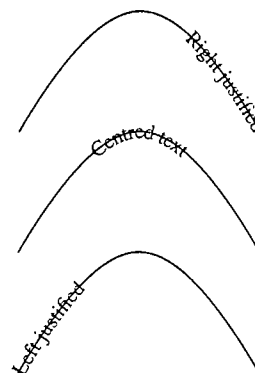
PostScript aficionados are aware that text in PostScript is treated just like any other graphical object (this was one of the great revolutions introduced by the language) and can be manipulated. The following examples demonstrate the `\pstextpath` macro, which has two parameters: a graphic object and some text. The text is made to fit the graphical object (here a `\pscurve`); only simple text is allowed, but that includes math.¹



4-5-10

```
\usepackage {pstcol,pst-text}
\begin{pspicture}{-4,-3.2}(3,0.2)
\psset{linecolor=lightgray}
\pstextpath
{\pscurve(-4,-2)(-2,0)(0,-3)(2,-1)(3,-2)}
{\color{blue}
If you want to type
 $\sum_{n=1}^3 x_n$ 
and let it follow along a
curving line \ldots}
\end{pspicture}
```

By using an optional argument to `\pstextpath`, the text can be shifted in relation to the ends of the path:



4-5-11

```
\usepackage {pstricks}
\begin{pspicture}(0,0)(4.2,6.2)
\pstextpath[l]
{\pscurve(0,0)(2,2)(4,0)}
{Left justified}
\pstextpath[c]
{\pscurve(0,2)(2,4)(4,2)}
{Centred text}
\pstextpath[r]
{\pscurve(0,4)(2,6)(4,4)}
{Right justified}
\end{pspicture}
```

In the examples above, we asked that the controlling curve be drawn; in the more complex example below, we make it invisible; we place text to fit the top and bottom of a circle (using the `\psarc` and `\psarcn` commands to specify two semicircles). A filled circle with no border (since `\psset` is used to turn off line drawing for subsequent objects) is placed

¹ Except for structures like radicals that use T_EX rules to construct the top line; these structures cannot be handled properly.

in the middle and \rput is used to typeset the banner text. It is true that Rincewind has not yet been made Arch Chancellor at the time of writing, but who knows?



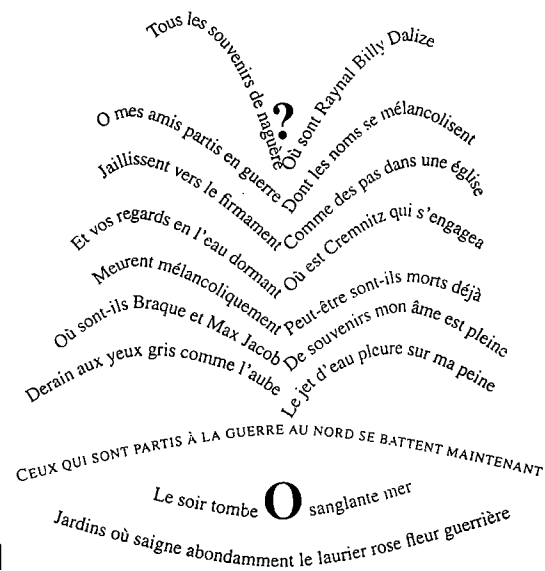
```
\usepackage{pstcol,pst-text}
\definecolor{pink}{rgb}{1,.75,.8}
\begin{pspicture}(-3,-2.2)(3,2.2)
\psset{linestyle=none}
\newcommand{\curly}[1]{\fontfamily{pzc}%
\fontsize{17}{17}\itshape#1}}
\pstextpath[c]{\psarcn(0,0){2}{180}{0}}
{\curly{The Unseen University}}
\pstextpath[c]{\psarc(0,0){2}{180}{0}}
{\curly{Ankh-Morpork,Discworld}}
\pscircle[fillstyle=gradient,
gradangle=45,gradbegin=pink,
gradend=yellow](0,0){1.7}
\rput[B](0,0){\Large\itshape\bfseries
Rincewind, Arch Chancellor}}
\end{pspicture}
```

4-5-12

A more extended example of text set along graphics paths is shown in a portion of the poem by Guillaume Apollinaire, *La colombe poignardée et le jet d'eau*, re-typeset by Denis Girou following the layout of original. The coordinates of each of the curves were obtained by placing a copy of the text on graph paper and reading off some points along each curve of the branch. To automate and parameterize this for an arbitrary text would be a nontrivial (though conceivable) task.

```
\usepackage[latin1]{inputenc}
\usepackage{pstricks,pst-text}
\begin{pspicture}(9,12)
\small\psset{linestyle=none}
\pstextpath[r]{\pscurve(1.5,8)(3,9)(4.45,6.5)}
{Tous les souvenirs de naguère}
\pstextpath[r]{\pscurve(1,7)(2,7.5)(4.45,6)}
{O mes amis partis en guerre}
\pstextpath[r]{\pscurve(1,6)(2,6.7)(4.45,6.2)}
{Jaillissent vers le firmament}
\pstextpath[r]{\pscurve(0.8,5)(2,5.8)(4.45,4.4)}
{Et vos regards en l'eau dormant}
\pstextpath[r]{\pscurve(1,4.3)(2,5)(4.45,3.8)}
{Meurent mélancoliquement}
\pstextpath[r]{\pscurve(0.3,3.2)(2,4.3)(4.45,3.3)}
{Où sont-ils Braque et Max Jacob}
\pstextpath[r]{\pscurve(0.2,2.7)(2,3.5)(3.8,3.2)(4.45,2.8)}
{Derain aux yeux gris comme l'aube}
\rput(4.5,7.3){\bfseries\Huge ?}
\pstextpath[l]{\pscurve(4.55,6.5)(6.5,8.7)(7.8,8)}
```

```
{Où sont Raynal Billy Dalize}
\pstextpath[l]{\pscurve(4.55,5.8)(6.5,7.5)(8.3,6.4)}
{Dont les noms se mélancolisent}
\pstextpath[l]{\pscurve(4.55,5.2)(6.5,6.6)(7.6,6.4)(8.2,5.2)(8.5,5.4)}
{Comme des pas dans une église}
\pstextpath[l]{\pscurve(4.55,4.5)(6.5,5.8)(8.5,4.7)}
{Où est Cremnitz qui s'engagea}
\pstextpath[l]{\pscurve(4.55,3.8)(6.5,4.7)(8.5,4)}
{Peut-être sont-ils morts déjà}
\pstextpath[l]{\pscurve(4.55,3.2)(6.5,4.2)(9,2.8)}
{De souvenirs mon âme est pleine}
\pstextpath[l]{\pscurve(4.55,2.4)(5,3)(6.5,3.5)(8.8,2.3)}
{Le jet d'eau pleure sur ma peine}
\pstextpath[c]{\pscurve(-0.1,1.4)(4.5,2.2)(9,1.4)}
{{\footnotesize\scshape Ceux qui sont partis à la guerre au nord se battent
maintenant}}
\pstextpath[c]{\pscurve(2.1,1.2)(4.5,0.8)(6.9,1.3)}
{Le soir tombe {\bfseries\Huge O} sanglante mer}
\pstextpath[c]{\pscurve(0,1)(4.5,0)(9,1)}
{Jardins où saigne abondamment le laurier rose fleur guerrière}
\end{pspicture}
```



4-5-13

It is also possible to set text as if it were a graphic, with the fill, color, and line properties. This is done with the `\pscharpath` macro; it has limitations (for instance, it cannot be mixed with `\pstextpath`), but is still a useful tool for special occasions. The following example is also printed as Color Plate IIIb:

```
\usepackage {pstcol,pst-char}
\definecolor{lightblue}{rgb}{.68, .85, .9}
\begin{pspicture}(0,-1)(8.7,2.5)
\fontfamily{ptm}\selectfont
\pscharpath[fillstyle=solid,linecolor=red,
            fillcolor=lightblue,linewidth=2pt]
{\fontsize{1.5in}{1.5in}\selectfont \LaTeX}
\end{pspicture}
```



4-5-14

Admirers of the distinctive style of Dorling Kindersley publications may like to try the following effect (also printed as Color Plate IIIc); `\DeclareFixedFont` is used to load letters quickly at a large size without worrying about making baseline changes and so on. The font family *zmn* is Adobe Multiple Master Minion, the main text font in this book.

```
\usepackage {pstcol,pst-char}
\begin{pspicture}(-5,-3)(5,1.2)
\psset{fillstyle=solid,shadow=true,shadowangle=0}
\DeclareFixedFont{\babyfont}{T1}{zmn}{m}{n}{3cm}
\DeclareFixedFont{\wordfont}{T1}{zmn}{m}{n}{2cm}
\newcommand{\Cc}[2]{\pscharpath[fillcolor=#1]{#2}}
\bfseries
\rput(0,0){\babyfont
\Cc{red}B\Cc{green}A\Cc{yellow}B%
\Cc{red}{Y'}\Cc{blue}S}
\rput(0,-2){\wordfont
\Cc{blue}{WORLD}}
\end{pspicture}
```



4-5-15

Setting each letter in a different color has the unfortunate side effect that kerning and ligatures are no longer available, but you are unlikely to use this effect except in very specialized circumstances.

The third of these tools that treat text like a graphic is `\pscharclip`; this takes some text as a parameter and is terminated by `\endpscharclip`. Any objects drawn inside this group are clipped to the shape of the letters (the example is also printed as Color Plate IIId).

```
\usepackage {pstcol,pst-char,ifthen}
\DeclareFixedFont{\bigsf}{T1}{phv}{b}{n}{1.75cm}
\DeclareFixedFont{\tinyrm}{T1}{ptm}{m}{n}{2mm}
\newcounter{myN}
\setcounter{myN}{300}
\begin{pspicture}(0,0)(11,1.4)
\pscharclip[linecolor=red,fillstyle=solid,
            fillcolor=yellow]
{\rput[bl](0,0){\bigsf CHOCOLATE}}
\rput[t]{90}(0,0)
{\begin{minipage}{4cm}
\offinterlineskip
\raggedright\tinyrm\color{black}%
\whiledo{\value{myN}>0}%
{%
\addtocounter{myN}{-1}
nuts and raisins
}%
\end{minipage}}%
\endpscharclip
\end{pspicture}
```

4-5-16

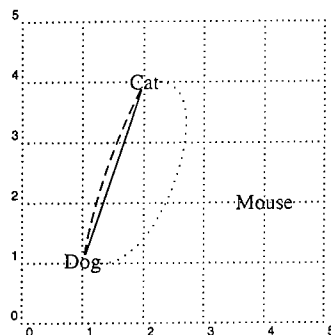


4.6 Nodes and their connections, and trees

PSTricks offers sophisticated macros for setting up named nodes and joining them together in complicated ways, complete with labels. The nodes can be created in three ways:

- By placing them at arbitrary coordinates;
- by placing them on a regular grid or matrix;
- by using higher-level tree macros.

The sort of effect we will create is like this:



```
\usepackage {pstricks,pst-node}
\begin{pspicture}(0,0)(5,5)\showgrid
\rput(1,1){\rnode{A}{Dog}}
\rput(2,4){\rnode{B}{Cat}}
\rput(4,2){Mouse}
\ncline{A}{B}
\nccurve[linestyle=dotted]{A}{B}
\ncarc[linestyle=dashed]{A}{B}
\end{pspicture}
```

4-6-1

Every node is given a symbolic name that is used for node connectors (lines, curves and so on) and node labels. The fact that nodes can occur *anywhere*, even in running text, makes it possible to use them for surprising effects like linking together two words. When considering connectors, we have to distinguish between the position of the node they point *towards* (the node reference point) and the actual extent of the connector. Most node creators have an invisible box around them, that determines the end of connector lines. Section 4.13.2 on p. 159 lists all the node creation and node connector commands, and Section 4.13.3 on p. 161 lists the commands to label connectors and nodes. Because labeling node connectors is a very common thing to do, a shortcut is provided for all the commands like `\naput`. If the key shortcut is set to nab, the `~` can be used instead of `\naput` and `_` instead of `\nbput`. If it is set to tab, `~` stands for `\taput`, `_` for `\tbput`, `<` for `\tlput` and `>` for `\trput`. Table 4.6 on p. 161 then lists the extra graphical parameters that apply to node connectors.

There are several important concepts we need to bear in mind when looking at the myriad of node and connector commands:

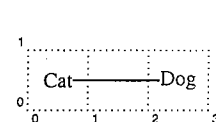
- When joining two nodes with something like a curve (as opposed to a simple line), the connectors are put by default on the right hand side of the object (at 0 degrees). If you have two boxes side by side, set the `angleB` parameter to 180 to make the connector

enter the second box on its left side. This may seem cumbersome at first, but it makes for a flexible system.

- the connector-labeling commands all place the label some proportion of the way along the connector, but they distinguish between the distance between nodes, and the length of the line (which depends on the size of the nodes it joins). The former is useful in constructions like matrices in which the label positions should be constant, regardless of the size of the nodes;
- the node connectors are not drawn directly in L^AT_EX but at the PostScript level; this means that L^AT_EX is not always quite sure how much space is taken up by the object. Curving connectors, in particular, may protrude outside the area allowed by L^AT_EX—adjust by hand.

We proceed by looking at simple examples of most node and node connector commands and then give a more complex example; the next section, describing the matrix and tree packages, provides more 'real' examples and in particular shows connector labeling.

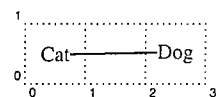
If you want text as the node, the `\rnode` command is simple to use; the connector `\ncline` is the simplest form of join.



```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,1)\showgrid
\rput(.5,.5){\rnode{A}{Cat}}
\rput(2.5,.5){\rnode{B}{Dog}}
\ncline{A}{B}
\end{pspicture}
```

4-6-2

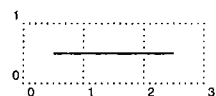
`\Rnode` is similar, but the connectors join the text baselines rather than the encompassing box:



```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,1)\showgrid
\rput(.5,.5){\Rnode{A}{Cat}}
\rput(2.5,.5){\Rnode{B}{Dog}}
\ncline{A}{B}
\end{pspicture}
```

4-6-3

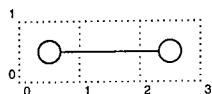
`\pnode` simply creates a point to anchor a connector:



```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,1)\showgrid
\pnode(.5,.5){A}
\pnode(2.5,.5){B}
\ncline{A}{B}
\end{pspicture}
```

4-6-4

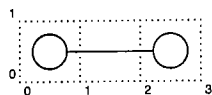
Simple circular nodes can be created with `\cnode`, which gets the radius as a parameter:



```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,1)\showgrid
\cnode(.5,.5){.2}{A}
\cnode(2.5,.5){.2}{B}
\ncline{A}{B}
\end{pspicture}
```

4-6-5

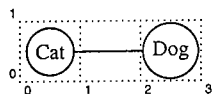
while `\Cnode` does not need a radius parameter but follows the general radius key:



```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,1)\showgrid
\psset{radius=.3}
\Cnode(.5,.5){A}
\Cnode(2.5,.5){B}
\ncline{A}{B}
\end{pspicture}
```

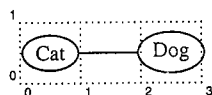
4-6-6

Text and a frame are combined as a node with `\circnnode`, `\ovalnode`, `\dianode`, and `\trinode`, used with `\rput`:



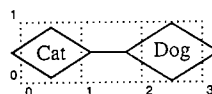
```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,1)\showgrid
\rput(.5,.5){\circnnode{A}{Cat}}
\rput(2.5,.5){\circnnode{B}{Dog}}
\ncline{A}{B}
\end{pspicture}
```

4-6-7



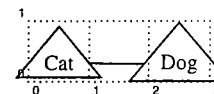
```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,1)\showgrid
\rput(.5,.5){\ovalnode{A}{Cat}}
\rput(2.5,.5){\ovalnode{B}{Dog}}
\ncline{A}{B}
\end{pspicture}
```

4-6-8



```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,1)\showgrid
\rput(.5,.5){\dianode{A}{Cat}}
\rput(2.5,.5){\dianode{B}{Dog}}
\ncline{A}{B}
\end{pspicture}
```

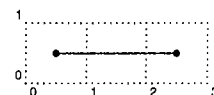
4-6-9



4-6-10

```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,1)\showgrid
\rput(.5,.5){\trinode{A}{Cat}}
\rput(2.5,.5){\trinode{B}{Dog}}
\ncline{A}{B}
\end{pspicture}
```

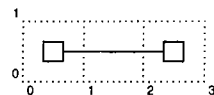
`\dotnode` is like `\pnode`, but is a filled small circle:



4-6-11

```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,1)\showgrid
\dotnode(.5,.5){A}
\dotnode(2.5,.5){B}
\ncline{A}{B}
\end{pspicture}
```

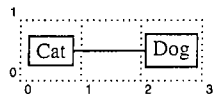
As you may be beginning to expect, many of the primitive commands introduced earlier in the chapter have a node equivalent. That for `\psframebox` is `\fnnode`:



4-6-12

```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,1)\showgrid
\fnode(.5,.5){A}
\fnode(2.5,.5){B}
\ncline{A}{B}
\end{pspicture}
```

but you can also work directly with `\psframebox` inside an `\rnode`:

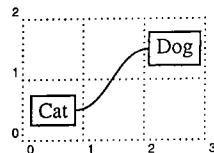


4-6-13

```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,1)\showgrid
\rput(.5,.5){\rnode{A}{\psframebox{Cat}}}
\rput(2.5,.5){\rnode{B}{\psframebox{Dog}}}
\ncline{A}{B}
\end{pspicture}
```

Let us now look at some of the connectors between nodes. So far, we have used simple lines, which behave in a straightforward manner, taking the shortest path between two nodes. When we use curves, we have to worry about the angle at which they join the node, as explained at the start of this section. Thus we have to set `angleB` to 180 in the following

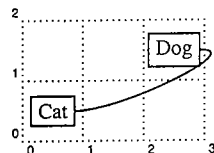
to have the curve (`\ncurve`) join the 'Dog' node on the left instead of the right:



```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,2)\showgrid
\rput(.5,.5){\rnode{A}{\psframebox{Cat}}}
\rput(2.5,1.5){\rnode{B}{\psframebox{Dog}}}
\ncurve[angleB=180]{A}{B}
\end{pspicture}
```

4-6-14

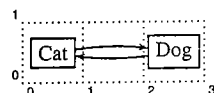
If we omit that parameter, the effect is probably not what we want:



```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,2)\showgrid
\rput(.5,.5){\rnode{A}{\psframebox{Cat}}}
\rput(2.5,1.5){\rnode{B}{\psframebox{Dog}}}
\ncurve{A}{B}
\end{pspicture}
```

4-6-15

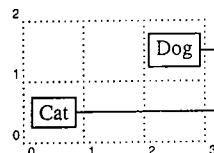
The `\ncarc` connector does not need this check, as it takes care of the situation by itself:



```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,1)\showgrid
\rput(.5,.5){\rnode{A}{\psframebox{Cat}}}
\rput(2.5,.5){\rnode{B}{\psframebox{Dog}}}
\ncarc->{A}{B}
\ncarc->{B}{A}
\end{pspicture}
```

4-6-16

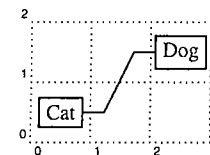
There are six multipart connectors, `\ncbar`, `\ncdiag`, `\ncdiagg`, `\ncangle`, `\ncangles`, and `\ncloop`, that are concerned about the `angleA` and `angleB` keys and join the nodes with a set of segments:



```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,2)\showgrid
\rput(.5,.5){\rnode{A}{\psframebox{Cat}}}
\rput(2.5,1.5){\rnode{B}{\psframebox{Dog}}}
\ncbar{A}{B}
\end{pspicture}
```

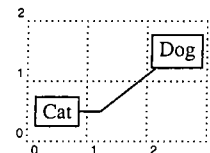
4-6-17

4-6-18



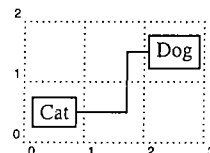
```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,2)\showgrid
\rput(.5,.5){\rnode{A}{\psframebox{Cat}}}
\rput(2.5,1.5){\rnode{B}{\psframebox{Dog}}}
\ncdiag[angleB=180]{A}{B}
\end{pspicture}
```

4-6-19



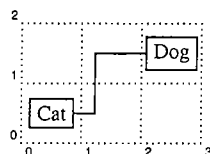
```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,2)\showgrid
\rput(.5,.5){\rnode{A}{\psframebox{Cat}}}
\rput(2.5,1.5){\rnode{B}{\psframebox{Dog}}}
\ncdiagg[angleB=180]{A}{B}
\end{pspicture}
```

4-6-20



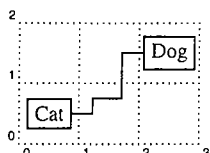
```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,2)\showgrid
\rput(.5,.5){\rnode{A}{\psframebox{Cat}}}
\rput(2.5,1.5){\rnode{B}{\psframebox{Dog}}}
\ncangle[angleB=180]{A}{B}
\end{pspicture}
```

4-6-21



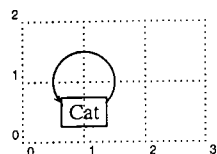
```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,2)\showgrid
\rput(.5,.5){\rnode{A}{\psframebox{Cat}}}
\rput(2.5,1.5){\rnode{B}{\psframebox{Dog}}}
\ncangles[angleB=180]{A}{B}
\end{pspicture}
```

4-6-22



```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,2)\showgrid
\rput(.5,.5){\rnode{A}{\psframebox{Cat}}}
\rput(2.5,1.5){\rnode{B}{\psframebox{Dog}}}
\ncloop[loopsize=.25,angleB=180]{A}{B}
\end{pspicture}
```

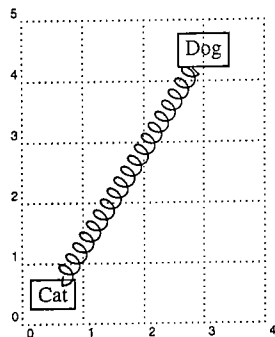
There is a special `\nccircle` command to make a node connect to itself:



```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3.2)\showgrid
\rput(1,.5){\rnode{A}{\psframebox{Cat}}}
\nccircle{->}{A}{.5}
\end{pspicture}
```

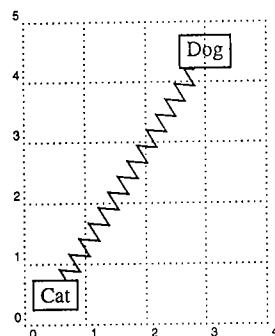
4-6-23

At a higher level, PSTricks offers two special connectors that join with a coil (`\nccoil`) or a zigzag (`\nczigzag`)—these require loading the `pst-coil` package:



```
\usepackage {pstricks,pst-node,pst-coil}
\begin{pspicture}(-.2,-.5)(3.5,5)\showgrid
\rput(.5,.5){\rnode{A}{\psframebox{Cat}}}
\rput(3.5,4.5){\rnode{B}{\psframebox{Dog}}}
\psset{coilarm=.01,coilwidth=.3}
\nccoil{A}{B}
\end{pspicture}
```

4-6-24

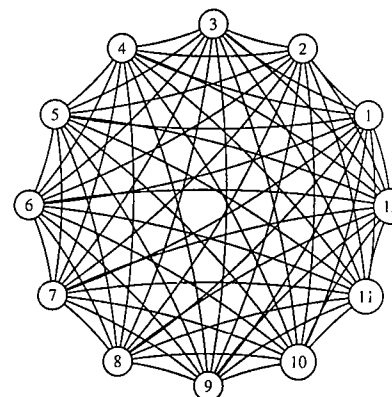
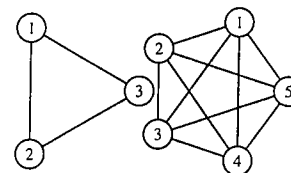


```
\usepackage {pstricks,pst-node,pst-coil}
\begin{pspicture}(-.2,-.5)(3.5,5)\showgrid
\rput(.5,.5){\rnode{A}{\psframebox{Cat}}}
\rput(3.5,4.5){\rnode{B}{\psframebox{Dog}}}
\psset{coilarm=.01,coilwidth=.3}
\nczigzag{A}{B}
\end{pspicture}
```

4-6-25

If all this is not enough, all the node connectors can also be used as ordinary drawing tools, by using the commands listed in Section 4.13.4 on p. 163, where the “pc” version corresponds to the “nc” node connector.

The effect of node connectors is demonstrated in the following, which uses the `\multido` macro (see Section 4.9 on p. 139) to place objects at regular intervals around a circle and join them up. Why in the definition of the `\Wheel` object do we explicitly set the color to black, and then set it to white before using it several times? This illustrates a technique that is sometimes useful to be sure that L^AT_EX generates the correct space in the output dvi file. We used the L^AT_EX `\fbox` command to put a frame around the whole thing, but set the color to white, so that the frame does not show, but just anchors the object in L^AT_EX. See p. 457 for more discussion of this technique.



4-6-26

```
\usepackage {pstcol,pst-node,multido}
\newcounter{CtA} \newcounter{CtB}
\newcommand{\Wheel}[3]{\{
\color{black} \psset{unit=#2}
\pspicture(-1,-1.2)(1,1.2)
\SpecialCoor \degrees[#1]
\multido{\ia=1+1}{#1}{
\setcounter{CtA}{\ia}
\stepcounter{CtA}
\setcounter{CtB}{#1}
\addtocounter{CtB}{-\ia}
\multido{\ib=\value{CtA}+1}
{\value{CtB}}{\#3(1;\ia)(1;\ib)}}
\multido{\i=1+1}{#1}{
\rput(1;\i){
\pscirclebox[fillstyle=solid,
fillcolor=white]{
\footnotesize\i}}}
\end{pspicture}}% end of newcommand
\color{white}
\fbox{\Wheel{3}{1.2}{\psline}
\Wheel{5}{1.2}{\psline}}

\psset{arcangle=10}
\fbox{\Wheel{12}{3}{\pcarc[linecolor=blue]}}
```

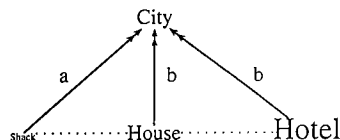
It should be clear that we could draw arbitrary diagrams and trees simply by working out the coordinates of each node; in practice, however, it is easier to use two higher-level environments for matrices and trees, discussed in the next section.

4.6.1 Matrices — grid-based nodes

The existing L^AT_EX `tabular` or `AMS-LATEX` `matrix` can be used to place nodes, but PSTricks provides its own environment, `psmatrix`. This is like an easy form of table, since the number of columns need not be specified—we simply separate column items

by & and rows by \\ as normal, and PSTricks makes each cell a node, named *rownumber, columnnumber*. Thus the first node in the first row is named 1, 1 and the third node in the fourth row is 4, 3; these names are used by the node connectors.

psmatrix has an optional parameter in which rowsep and colsep can be set to determine the gap between nodes. The parameter shortput (see page 114) is set to tab inside psmatrix by default, so we can adopt a quite succinct notation: the node connection labeling commands immediately follow the relevant connector.

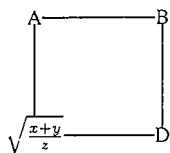


```
\usepackage {pstricks,pst-node}
\begin{pspicture}(0,0)(2.5,5)
\begin{psmatrix}[rowsep=1.5cm]
    & City & \\
    {\tiny Shack} & House & {\Large Hotel}
\psset{arrows=<-}
\ncline{1,2}{2,1}<{a}
\ncline{1,2}{2,2}>{b}
\ncline{1,2}{2,3}>{b}
\psset{arrows=-,linestyle=dotted}
\ncline{2,1}{2,2}
\ncline{2,2}{2,3}
\end{psmatrix}
\end{pspicture}
```

4-6-27

Notice that the shorthand '>' stands for \trput, which places labels according to the distance between node centres, not the connection length, so as to accommodate the difference in sizes of nodes in the second row. Nodes can be made to span multiple columns by using the \psspan command at the end of the cell, with a parameter of the number of columns to span.

Let us consider a simple square matrix:

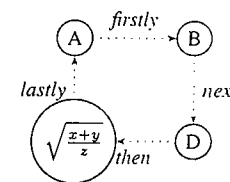


```
\usepackage {pstricks,pst-node}
\begin{psmatrix}
    A & & B \\
    {\sqrt{\frac{x+y}{z}}} & & D
\ncline{1,1}{1,2}
\ncline{1,2}{2,2}
\ncline{2,2}{2,1}
\ncline{2,1}{1,1}
\end{psmatrix}
```

4-6-28

By changing a few initial parameter settings, we can present a “fancier” result, with nodes encircled, arrows on connectors, and a better spacing. The connector labels are added

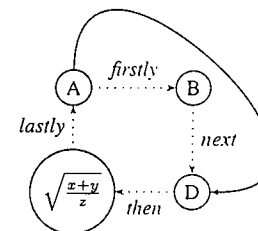
with the shorthand forms, those that are positioned in relation to node centers.



4-6-29

```
\usepackage {pstcol,pst-node}
\psset{arrows=->,labelsep=3pt,
linecolor=gray,mnode=circle}
\begin{psmatrix}[rowsep=20pt,colsep=28pt]
    A & B \\
    {\sqrt{\frac{x+y}{z}}} & D
\psset{linestyle=dotted}
\ncline{1,1}{1,2}^{\emph{firstly}}
\ncline{1,2}{2,2}>\emph{next}
\ncline{2,2}{2,1}_{\emph{then}}
\ncline{2,1}{1,1}<\emph{lastly}}
\end{psmatrix}
```

In the second version below, we redo the labels with the label types that relate to line length, which in this case gives a better result—compare the vertical positions of the *lastly* on the left-hand side. The outer looping connector is an example of a construct whose extent T_EX will probably guess incorrectly.



4-6-30

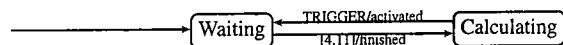
```
\usepackage {pstcol,pst-node}
\psset{arrows=->,labelsep=3pt,
linecolor=gray,mnode=circle}
\begin{psmatrix}[rowsep=20pt,colsep=28pt]
    A & B \\
    {\sqrt{\frac{x+y}{z}}} & D
\psset{linestyle=dotted}
\ncline{1,1}{1,2}\naput{\emph{firstly}}
\ncline{1,2}{2,2}\naput{\emph{next}}
\ncline{2,2}{2,1}\naput{\emph{then}}
\ncline{2,1}{1,1}\naput{\emph{lastly}}
\ncurve[ncurv=2,
linestyle=solid,angleA=90]{1,1}{2,2}
\end{psmatrix}
```

Matrices can be nested, and nodes from two different matrices can be linked, if the nodes are given explicit names. Each psmatrix wipes out the current set of *row, column* names.

In the following example we show the use of an empty node to ensure that a connector comes in from the side, and the method for connecting two nodes twice, with slightly different offsets for the connectors.

```
\usepackage {pstricks,pst-node}
\psset{framearc=.4}
\begin{psmatrix}[colsep=3]
    \pnode{A}&\psframebox{Waiting} & \psframebox{Calculating}
\ncline[arrows=->]{1,1}{1,2}
\psset{offset=-.1}
```

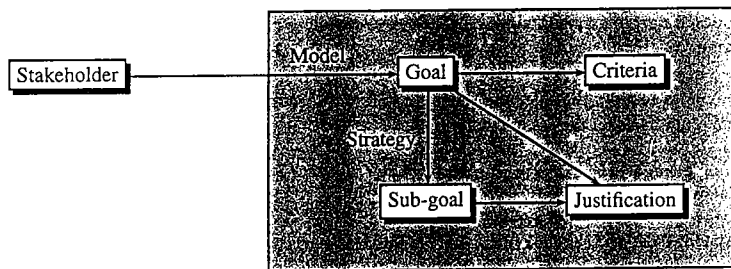
```
\ncline[arrows=->]{1,2}{1,3}\naput{\scriptsize TRIGGER/activated}
\psset{offset=.1}
\ncline[arrows=<-]{1,2}{1,3}\nbput{\scriptsize [4,11]/finished}
\end{psmatrix}
```



4-6-31

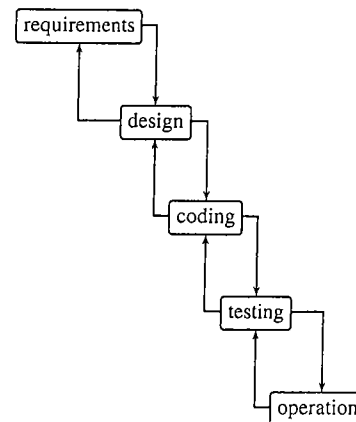
A normal low-level PSTricks command, like `\psframebox`, can be applied to a whole matrix. We have to take some care with alignment here (see Color Plate VIb) to make the connecting line horizontal, so we place the single node on the left in its own matrix.

```
\usepackage {pstcol,pst-node}
\psset{arrows=->,fillcolor=white,fillstyle=solid}
\newcommand{\Show}[1]{\psshadowbox{#1}}
\definecolor{pink}{rgb}{1, .75, .8}
\begin{psmatrix}[mnode=r,ref=t]
\psframebox[linestyle=none,framesep=.75]{%
\begin{psmatrix}
[name=A,ref=c]\Show{Stakeholder}
\end{psmatrix}} &
\psframebox[fillstyle=solid,fillcolor=pink,framesep=.75]{%
\rule{1cm}{0pt}
\begin{psmatrix}[ref=c]
[name=B]\Show{Goal} & \Show{Criteria}\\
\Show{Sub-goal} & \Show{Justification}
\ncline{1,1}{1,2}
\ncline{1,1}{2,2}
\ncline{1,1}{2,1}\tlput{Strategy}
\ncline{2,1}{2,2}
\end{psmatrix}}
\end{psmatrix}
\ncline[angleB=180]{A}{B}\naput[npos=.7]{Model}
```



4-6-32

A perhaps surprising effect is achieved with a *negative* column separation, which allows successive columns to overlap; the three-segment `\ncangle` connector is used below to join the boxes, but setting the `armA` key to 0 makes the first segment have no length. The `angleA` and `angleB` keys are set so that the first set of connectors leave the first node on the right (`angleA=0`) and join the second on the top (`angleB=90`), while the second set leave on the left (`angleA=180`) and join on the bottom (`angleB=-90`).



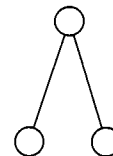
4-6-33

```
\usepackage {pstricks,pst-node}
\psset{framearc=.2}
\begin{psmatrix}[rowsep=1cm,colsep=-10pt]
[name=A]\psframebox{requirements}\\
& [name=B]\psframebox{design}\\
& & [name=C]\psframebox{coding}\\
& & & [name=D]\psframebox{testing}\\
& & & & [name=E]\psframebox{operations}
\psset{arrows=->,armA=0,angleB=90}
\ncangle{A}{B} \ncangle{B}{C}
\ncangle{C}{D} \ncangle{D}{E}
\psset{angleB=-90,angleA=180}
\ncangle{B}{A} \ncangle{C}{B}
\ncangle{D}{C} \ncangle{E}{D}
\end{psmatrix}
```

4.6.2 Tree diagrams

PSTricks has an extremely rich environment for drawing trees, that allows complex structures and presentation. The commands available are listed in Section 4.13.5 on p. 163 and the graphical parameters that apply especially to these are listed in Table 4.7 on p. 164. As one might expect, most other commands and parameters are also available, both the generalized drawing facilities, and the node connectors and labels. Each of the node types described earlier is turned into a “tree” node, and named by prefixing a “T” and removing the “node” suffix.

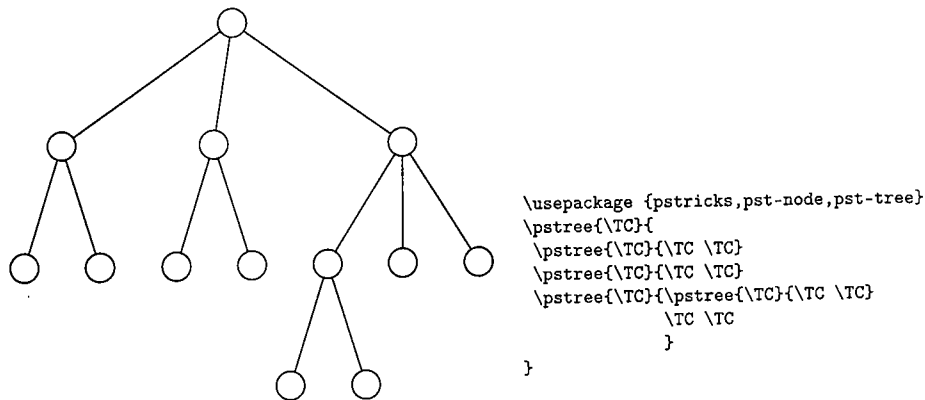
The fundamental concept in PSTricks trees is the *nesting* of trees; a tree (`\pstree`) consists of a root and one or more nodes:



4-6-34

```
\usepackage {pstricks,pst-node,pst-tree}
\pstree{\TC}{
\TC
\TC
}
```

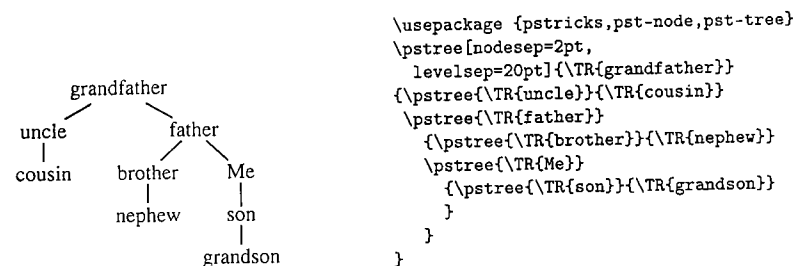

Each node can itself be a tree:



4-6-35

This construct allows arbitrarily complicated structures to be erected.

The following tree is a version of that shown in Goossens et al. (1994), section 10.5.2, which was drawn using the `edtree` package; the ease of notation is roughly similar. As often in L^AT_EX, the readability depends a great deal on how the code is laid out. The only change to the defaults is to lessen the vertical space between trees (`levelsep`) and add some extra space around nodes (`nodesep`).



4-6-36

The example of another tree drawing package described in Goossens et al. (1994) Section 10.2.3, Vanroose's trees, is a little harder to reproduce. The skeleton is trivial:



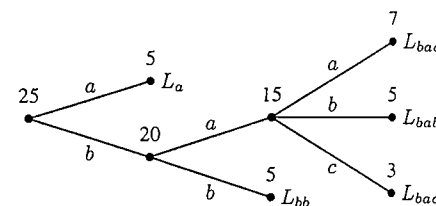
4-6-37

```

\usepackage {pstricks,pst-node,pst-tree}
\pstree{\Tdot}
{
  \Tdot
  \pstree{\Tdot}
  {
    \pstree{\Tdot}
    {
      \Tdot \Tdot \Tdot
    }
  }
}

```

but when we add all the node and connector labels and change a few parameters to make the result nicer, the markup becomes a little complex, though roughly similar in length to that of Vanroose:



4-6-38

```

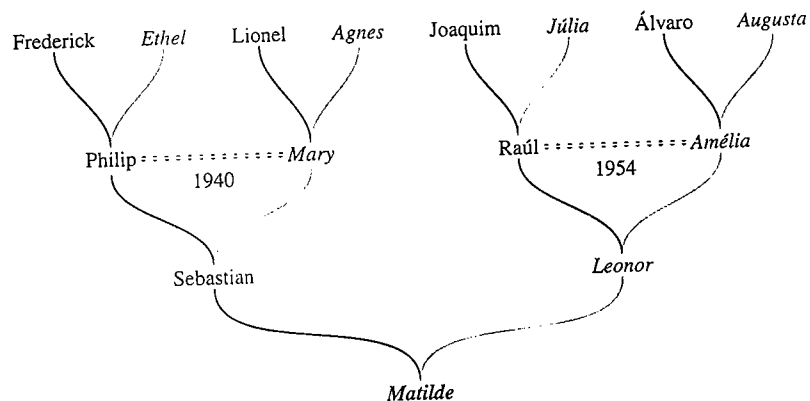
\usepackage {pstricks,pst-node,pst-tree}
\psset{labelsep=2pt,tnpos=a,radius=2pt}
\pstree[treemode=R]{\TC*~{25}}
{
  \TC*~{5}~[tnpos=r]{\L_a$}
  \taput{$a$}
  \pstree{\TC*~{20}\tbput{$b$}}
  {
    \pstree{\TC*~{15}\taput{$a$}}
    {
      \TC*~{7}~[tnpos=r]{\L_{baa}$}\taput{$a$}
      \TC*~{5}~[tnpos=r]{\L_{bab}$}\taput{$b$}
      \TC*~{3}~[tnpos=r]{\L_{bac}$}\tbput{$c$}
    }
    \TC*~{5}~[tnpos=r]{\L_{bb}$}
    \tbput{$b$}
  }
}

```

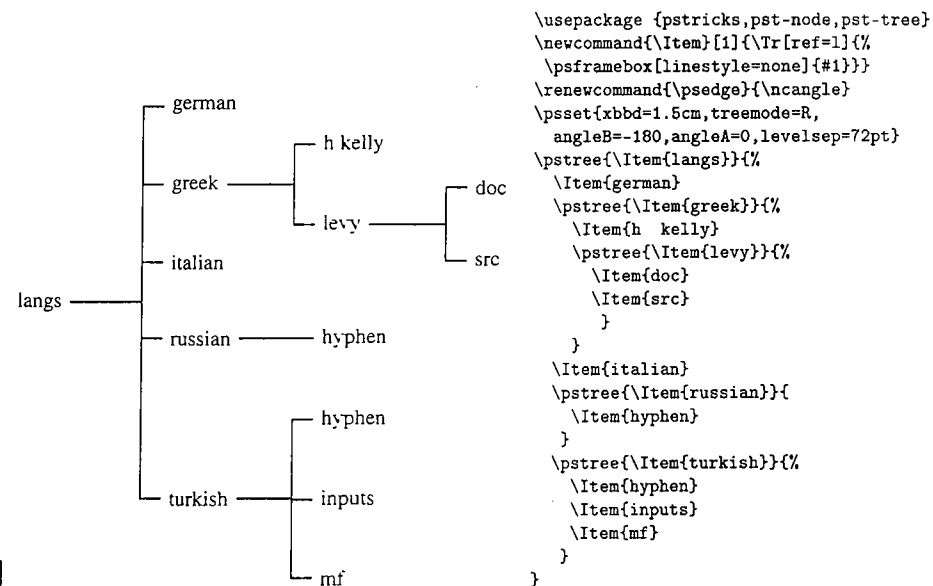
The node connectors in a tree are created by running the macro `\psedge` with the two nodes; the meaning of `\psedge` can be overridden explicitly by a redefinition or by using

the edge parameter. Here we redefine `\psedge` to be a curve, arrange the angles (bearing in mind that the tree is to grow upwards), and obtain a pleasing result. Note also the explicit links between named nodes as well as the regular connections.

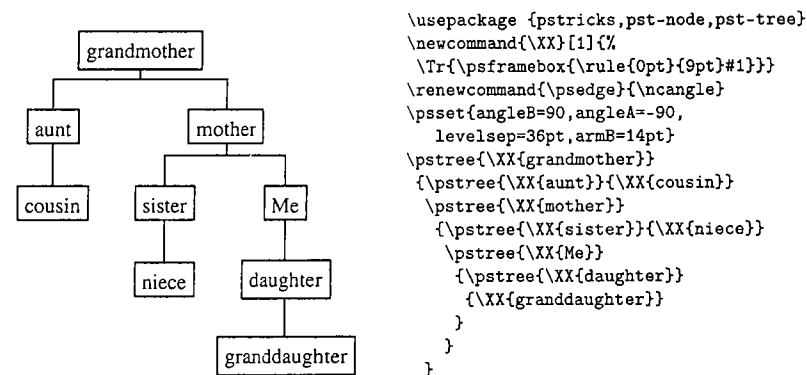
```
\usepackage {pstcol,pst-node,pst-tree}
\definecolor{pink}{rgb}{1, .75, .8}
\renewcommand\psedge{\ncurve}
\newcommand{\Female}[2][]{\psset{linecolor=pink}\TR[#1]{\emph{#2}}}}
\newcommand{\Male}[2][]{\psset{linecolor=blue}\TR[#1]{#2}}
\psset{nodesep=2pt,angleA=90,angleB=-90}
\pstree[treemode=U]{\Female{\bfseries Matilde}}{
  \pstree{\Male{Sebastian}}{
    \pstree{\Male[name=P]{Philip}}{\Male{Frederick}\Female{Ethel}}
    \pstree{\Female[name=W]{Mary}}{\Male{Lionel}\Female{Agnes}}
  }
  \pstree{\Female{Leonor}}{
    \pstree{\Male[name=R]{Ra'ul}}{\Male{Joaquim}\Female{J'ulia}}
    \pstree{\Female[name=A]{Am'elia}}{\Male{\Alvaro}\Female{Augusta}}
  }
}
\psset{linecolor=green,doubleline=true,linestyle=dotted}
\ncline{P}{W}\nbput{1940}
\ncline{R}{A}\nbput{1954}
```



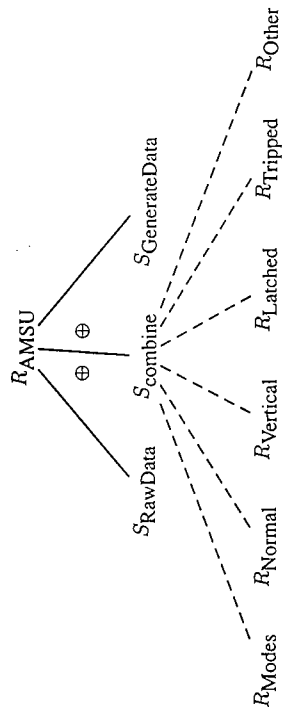
The technique of redefining edges is also necessary if we want right-angled joins rather than straight lines. Again, we need to take care of the `angleA` and `angleB` and ensure that in this left-right tree the nodes all line up on their left edges, using the `ref` parameter. If this is not done, the `\ncangle` edges produce strange results.



This technique allows us to remake the small family tree, this time with nicer connectors:



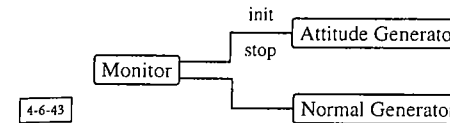
The node connectors are, of course, standard PSTricks objects, so they obey all the normal parameters; next we draw the lower part of the tree with dashed lines, and the contents of each node is set in math mode. The standard L^AT_EX graphics package command `\rotatebox` is used to turn the whole tree through 90 degrees, to demonstrate that the PSTricks constructs work normally with other packages.



```
\usepackage {pstcol,pst-node,pst-tree,graphics}
\rotatebox{90}{%
\psset{nodesep=2pt}
\newcommand{\XX}[2]{%
\TR{\ensuremath{#1_{\mbox{#2}}}}}%
}
\pstree[xbbr=1.5cm]{\XX{R}{AMSU}}
{
\XX{S}{RawData}
\pstree{\XX{S}{combine}}
{\trput{\ensuremath{\oplus}}}
{\tlput{\ensuremath{\oplus}}}
}
\psset{linestyle=dashed}
\XX{R}{Modes}
\XX{R}{Normal}
\XX{R}{Vertical}
\XX{R}{Latched}
\XX{R}{Tripped}
\XX{R}{Other}
}
\XX{S}{GenerateData}
}
```

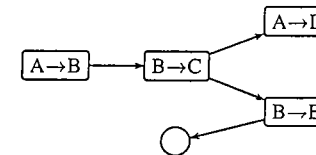
4-6-42

If we want to hang distinct-looking trees off one node, the `\ncangle` connector, with some offsets, produces the right result:



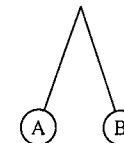
4-6-43

With simple connectors, we do not need to worry about the offsets or angles; what the next example shows is how nested trees can change direction:



4-6-44

Finally, let us not forget the simple empty node:



4-6-45

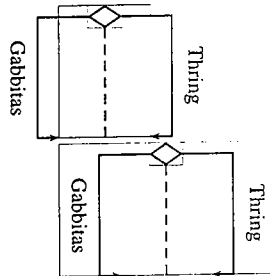
```
\usepackage {pstcol,pst-node,pst-tree}
\psset{framearc=.2,levelsep=4cm,
armB=1cm,angleB=-180}
\renewcommand{\psedge}{\ncangle}
\newcommand{\TreeBox}[1]{\Tr{\psframebox{#1}}}
\pstree[treemode=R]{\TreeBox{Monitor}}
{\psset{offsetA=4pt}
\TreeBox{Attitude Generator}
\naput[npos=2.5]{\small init}
\nbput[npos=2.5]{\small stop}
\psset{offsetA=-4pt}
\TreeBox{Normal Generator}
}
```

```
\usepackage {pstcol,pst-node,pst-tree}
\psset{arrows=->,framearc=.2}
\newcommand{\Treebox}[1]{%
\Tr{\psframebox{#1}}}
\pstree[treemode=R]
{\Treebox{A$\rightarrow$B}}{
\pstree{
\Treebox{B$\rightarrow$C}
}{
\Treebox{A$\rightarrow$D}
\pstree[treemode=L]
{\Treebox{B$\rightarrow$E}}
{\Tn\TC[arrows=<-]}
}
}
```

```
\usepackage {pstcol,pst-node,pst-tree}
\pstree{\Tp}{
\Tcircle{A}
\Tcircle{B}
}
```

4.6.3 Specifying tree bounding boxes

We said earlier that PSTricks does not always work out the extent of objects correctly, and this is illustrated by the connectors in the following example. Turning on `showbbox`, we see that without the use of `xbbl` and so on in the second incarnation, the bounding box is not correct:



```
\usepackage {pstcol,pst-node,pst-tree}
\psset{angleB=-90,arrows=->,nrot=:U}
\newcommand{\molesworth}[1]{%
\pstree[#1]{\Tdia{ }}
{
\Tp[arrows=->,edge={\ncbar[angleA=180]}}
\nbput{Gabbias}
{\psset{linestyle=dashed,arrows=->} \Tp }
\Tp[arrows=->,edge={\ncbar}]
\naput{Thring}
}
}
\psset{showbbox=true}
\begin{tabular}{l}
\molesworth{\\[1cm]}
\molesworth{xbbl=1cm,xbbr=1cm}
\end{tabular}
```

4-6-46

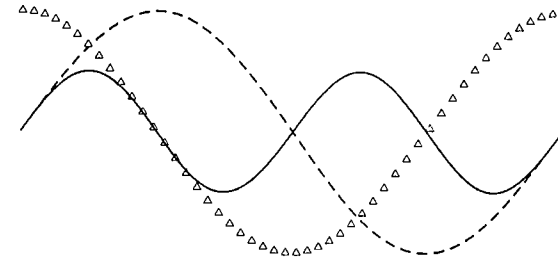
4.7 Data plotting

PSTricks has a set of high-level tools for common data-plotting functions that can read data from external files, in a variety of formats. The commands are listed in Section 4.13.6 on p. 165 and the extra parameters are listed in Table 4.8 on p. 166. The simplest form of data is a set of comma- or white-space-delimited numbers, but values can also be enclosed in braces ({}) or parentheses (()). If the data is enclosed in a single set of *square* brackets ([]) and the opening [is at the start of a line, it is read much faster; however, it runs T_EX out of memory sooner. Data can also be read once and then re-used with the `\readdata` and `\savedata` commands. There is an important distinction between `\fileplot` or `\dataplot`, which parse and validate the data in T_EX, and `\listplot`, which simply passes the data on to PostScript; the latter approach means that there is no check on PostScript memory requirements, but has the advantage that raw PostScript can be provided to generate or manipulate the data. Just to complicate matters, use of the `\PSTtoEPS` macro with the plotting commands can allow even bigger datasets. An example of this is given in Section 4.10.2 on p. 145.

Users must realize that it is up to them to check the data extents and scaling; unlike many other packages, PSTricks does not make the data fit a predefined plot area. Normally, judicious setting of `xunit` and `yunit` quickly produces nice results. Axes are generated separately, and no provision is made for supplying specific labels for axis tick points. It is, on the

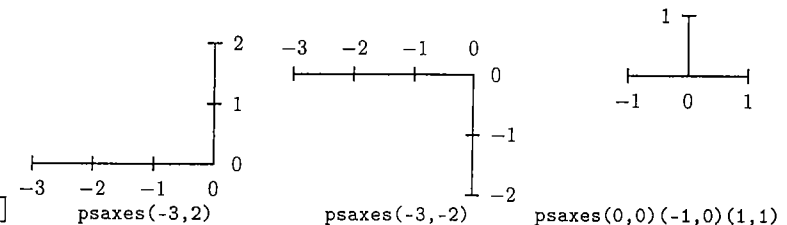
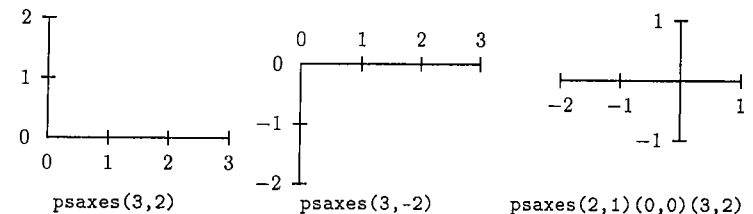
on the other hand, easy to superimpose multiple plots and use the PostScript language to calculate functions, as the following example shows, overlaying $\sin(x)$, $\sin(x) \cos(x)$, and $\cos(x)$.

```
\usepackage {pstcol,pst-plot}
\begin{pspicture}(-1,-2.5)(9,2.5)
\psset{xunit=.25mm,yunit=2cm}
\psset{plotpoints=50}
\psplot[linestyle=dashed,linecolor=blue]{0}{360}{x sin}
\psplot[plotstyle=dots,dotstyle=triangle]{0}{360}{x cos}
\psset{plotpoints=200}
\psplot[linecolor=red]{0}{360}{x dup sin exch cos mul}
\end{pspicture}
```



4-7-1

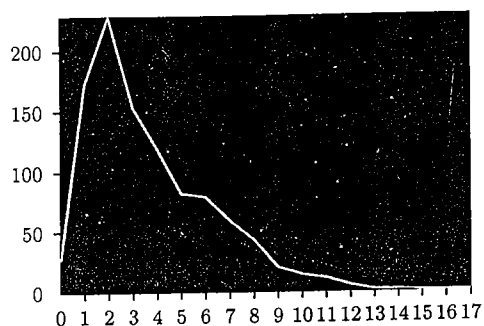
Axes or frames for plots can be created in a variety of ways; the default labeling tries to be intelligent, as these typical variations show:



4-7-2

The `\psaxes` command can also be used to create graph frames; in this mode, it obeys the graphical fill and color parameters. This example (showing word length against occurrences in a passage of Dickens) also shows how the axis labels can be manipulated.

```
\usepackage {pstcol,pst-plot}
\readdata{\foo}{words.dat}
\psset{yunit=.2mm,xunit=4mm}
\begin{pspicture}(-2,-50)(20,250)
\psaxes[axesstyle=frame,dy=50\psyunit,Dy=50,tickstyle=bottom,
fillcolor=gray,fillstyle=solid](1,1)(18,229)
\dataplot[plotstyle=line,linecolor=white]{\foo}
\end{pspicture}
```

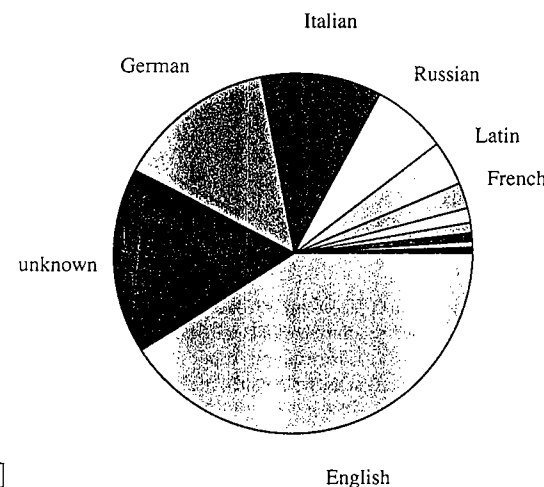


4-7-3

PSTricks is not designed to be a full-fledged business graphics package; its plotting functions are designed for simple scientific plots only. However, it should be clear that the lower-level PSTricks macros and T_EX's programmability make it a good basis for whatever graphing is needed. Below we show another version of the pie-chart we created with METAPOST on p. 80, with the macros and some of the data used to create it; note that only segments above a certain size are labeled—providing a sensible label for all elements would involve considerably more care. Denis Girou has produced a generalized pie-chart and bar-chart creation package for PSTricks that can produce very professional results (written in sh and awk, it is called `pie-chart.sh` and can be found in the PSTricks distribution.)

```
\usepackage{calc,pstcol}
\newcounter{lastval}
\newcounter{thishalf}
\newcounter{thisval}
\definecolor{lightblue}{rgb}{.68,.85,.9}
\newcommand{\lang}[4]{% name, value, proportion of 10000, color
\setcounter{thisval}{\value{lastval}}
\addtocounter{thisval}{#3}
\pswedge[fillcolor=#4]{1}{\thelastval}{\thethisval}%
```

```
\setcounter{thishalf}{((\value{thisval}-\value{lastval}))/2+\value{lastval}}
\ifnum#3>200\put(1.3;\thethisval){#1}\fi
\setcounter{lastval}{\value{thisval}}
}
\psset{fillstyle=solid}
\degrees[10000]
\SpecialCoor
\setcounter{lastval}{0}
\lang{Romanian}{1}{3}{green}
\lang{Czech}{2}{6}{blue}
...
\lang{German}{508}{1421}{lightblue}
\lang{unknown}{599}{1676}{red}
\lang{English}{1462}{4085}{yellow}
```



4-7-4

4.8 Working with a third dimension

Later versions of PSTricks offer some experimental facilities for viewing objects in three dimensions. Two-dimensional objects can be projected in a 3D coordinate system and arbitrary viewpoints established. At present PSTricks does not support true 3D solid objects, perspective projection, hidden-line removal, or lighting of objects, so the usefulness of this part of the package is limited. However, with some patience, pleasing effects can be obtained (well demonstrated, as usual, by Girou (1994)). Section 4.13.7 on p. 166 lists the new commands, and Table 4.9 on p. 166 describes the extra graphics parameter that apply to them.

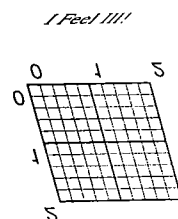
There are two high-level “3D” commands and one general tool. The first high-level command is `\psshadow`, which attaches to some text a shadow whose angle and size (as a proportion of the main text) can be set:

Words with a shadow

```
\usepackage {pstcol,pst-3d}
\psshadow[Tshadowangle=45,
  Tshadowsize=2.5]
{\LARGE\bfseries Words with a shadow}
```

4-8-1

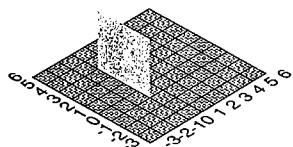
`\pstilt` angles an object into the third dimension:



```
\usepackage {pstcol,pst-3d}
\pstilt{45}{I Feel Ill!}
\begin{pspicture}(0,-2)(2,1)
  \pstilt{-75}{\psgrid(0,0)(2,2)}
\end{pspicture}
```

4-8-2

The general macro is `\ThreeDput`, which places any PSTricks object at a point in 3D space. This almost always uses the `normal` graphics parameter, setting the vector to be orthogonal to the plane of the object. To take a simple example, suppose we want to draw a cube with a reference origin at the left-hand end of the front panel; the left side panel would be drawn as follows:

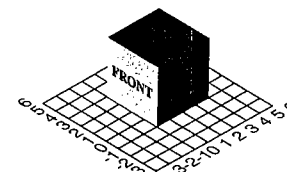


```
\usepackage {pstcol,pst-3d}
\begin{pspicture}(-5,-3)(4,3)
  \psset{unit=.3cm}
  \psset{viewpoint=-1 -1 1}
  \ThreeDput[normal=0 0 1]{\psgrid(-3,-3)(6,6)}
  \ThreeDput[normal=-1 0 0](0,4,0){
    \psframe*[linecolor=yellow](4,4)}
\end{pspicture}
```

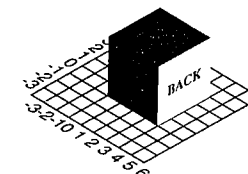
4-8-3

The viewpoint is from “up” (−1), “back” (−1) and “left” (1). The whole cube is

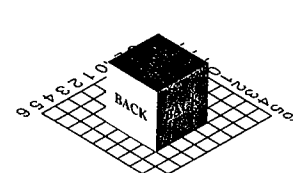
shown from the four “top” corners below:



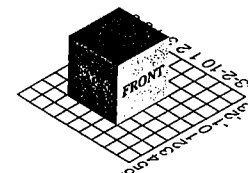
Viewpoint: -1 -1 1



Viewpoint: 1 -1 1



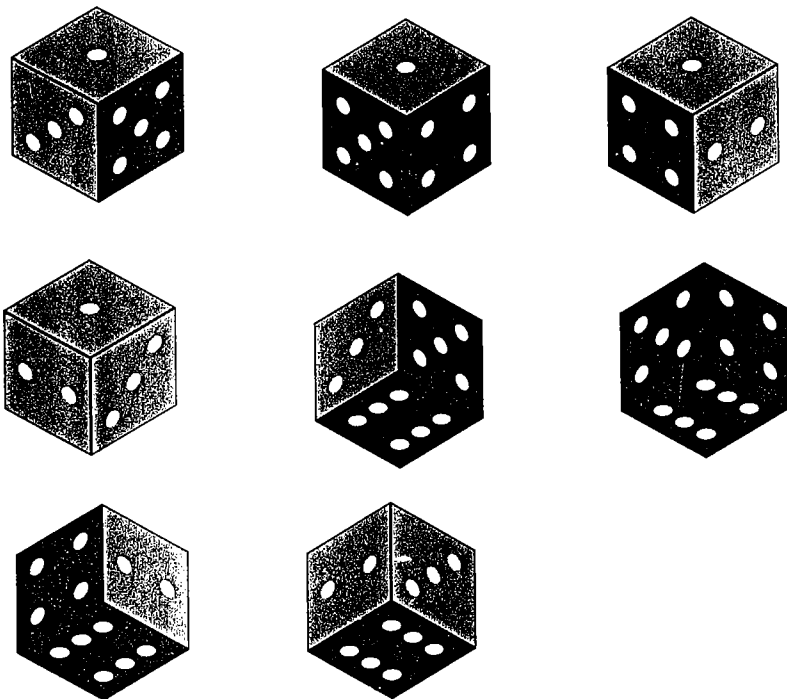
4-8-4 Viewpoint: 1 1 1



Viewpoint: -1 1 1

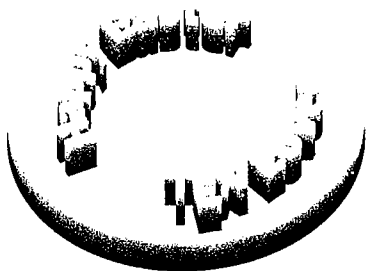
It is important to realize that PSTricks does not check which side hides which; the last side drawn wipes out those drawn earlier, even though it is behind them in 3D space. When drawing the different views, we have to give the sides in different orders. The following picture reinforces these issues; when we show the “underneath” well, we have to draw all six faces in the right order for the viewpoint. The crude “lighting” on the die is done simply by coloring three faces in a lighter color (to attempt formal ray-tracing from a light source is considerably beyond the scope of this package!) The values of the `normal` parameter for each of the die faces follow.

Face	normal			XYZ position
1	0	0	1	0,0,4
2	0	1	0	4,4,0
3	-1	0	0	0,4,0
4	1	0	0	4,0,0
5	0	-1	0	0,0,0
6	0	0	-1	0,4,0



4-8-5

Following the example of another picture by Denis Girou, we can use the `\ThreeDput` macro to give the appearance of raised text (also shown in Color Plate IVb):

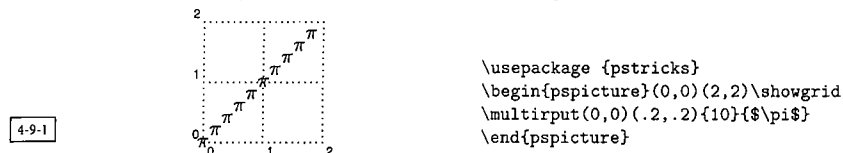


4.9 Iterating commands

To put a series of objects at regular intervals, you can use the repeating macro already used in this chapter:

```
\multirput*[refpoint][angle](x0,y0)(x1,y1){int}{stuff}
```

This acts like `\rput` but creates *int* copies, starting at (*x*₀,*y*₀) and advancing by *x*₁,*y*₁ each time. This works only with Cartesian coordinates. For example:



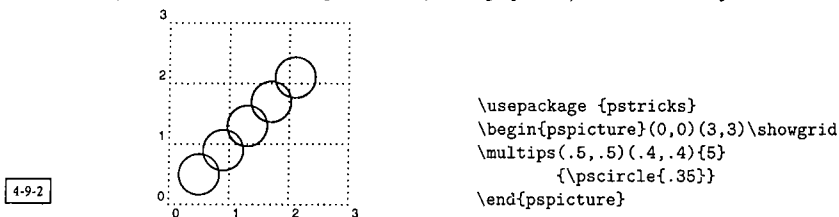
4-9-1

```
\usepackage {pstricks}
\begin{pspicture}(0,0)(2,2)\showgrid
\multirput(0,0)(.2,.2){10}{\pi}
\end{pspicture}
```

For regular repetition of normal graphic objects, there is another macro

```
\multips[angle][x0,y0](x1,y1){int}{graphics}
```

with syntax similar to `\multirput`; use only those graphic objects that take no space.



4-9-2

```
\usepackage {pstricks}
\begin{pspicture}(0,0)(3,3)\showgrid
\multips(.5,.5)(.4,.4){5}
{\pscircle{.35}}
\end{pspicture}
```

Note that (0, 0) for each copy of the graphic is shifted by the increment of `\multips`.

A more general repetition system is provided by the extra package `multido`, which is implicitly invoked when using the data-plotting commands (this package is in fact independent of PSTricks and can be used on its own). The syntax for `\multido` is:

```
\multido{variables}{repetitions}{actions}
```

where *actions* is something to be repeated and can be any valid TeX command. *repetitions* is the number of times *actions* is repeated. The first argument is a comma-separated list of variable declarations, each consisting of

variable = *initial value* + *increment*

The *variable* is set to *initial value* and is then incremented by *increment* at each repetition.

The first letter of the *variable* name determines the variable type. There are four variable types:

Dimension (d or D) The initial value and increment should be T_EX dimensions. The current value is represented in sp units (T_EX's scaled points). E.g., `\dx=4cm+5pt`.

Number (n or N) The initial value and increment should be integers or numbers with the same number of decimal digits (up to eight). Fixed-point addition is possible in incrementing. E.g., `\n=3+7.05`, `\Nx=5.30+-1.25`.

Integer (i or I) The initial value and increment should be integers. This gives the same result as using a number variable, but is faster. E.g., `\I=2+-1`.

Real (r or R) The initial value and increment should be integers or numbers with up to four digits on either side of the decimal. The substitution text is a number, but with floating point addition and occasional small errors. This gives a less satisfactory result than using a number variable but is faster. E.g., `\ry=4.2+1.05`.

The macro is documented with its source, which also describes some other features, but the main uses can be seen in the examples within this chapter. The sophisticated example (provided by Denis Girou) shown in Fig. 4.1 makes intense use of repeated, regular, actions.

4.10 Customizing and programming PSTricks

Although any graphics written in T_EX can use the macro programming features of T_EX itself, PSTricks provides extra facilities to describe pictures in more elegant ways than a simple list of commands. We suggest that this is an essential part of this type of picture drawing and is one of the main reasons for drawing within the T_EX environment rather than with an external drawing package. In this section we explore some of the features we can use and demonstrate styles of programming.

4.10.1 New PSTricks objects and styles

It is sometimes useful to avoid lengthy parameter lists and clumsy uses of `\psset` by creating special versions of graphic objects with their own default parameters. This is done with the `\newpsobject` command:

```
\newpsobject{name}{object}{par1=value1,...}
```

which creates a new object called *name*, based on the old *object*, with the various parameters preset. Thus:

```
\newpsobject{trafficsign}{pscirclebox}{linecolor=red,doubleline=true}
```

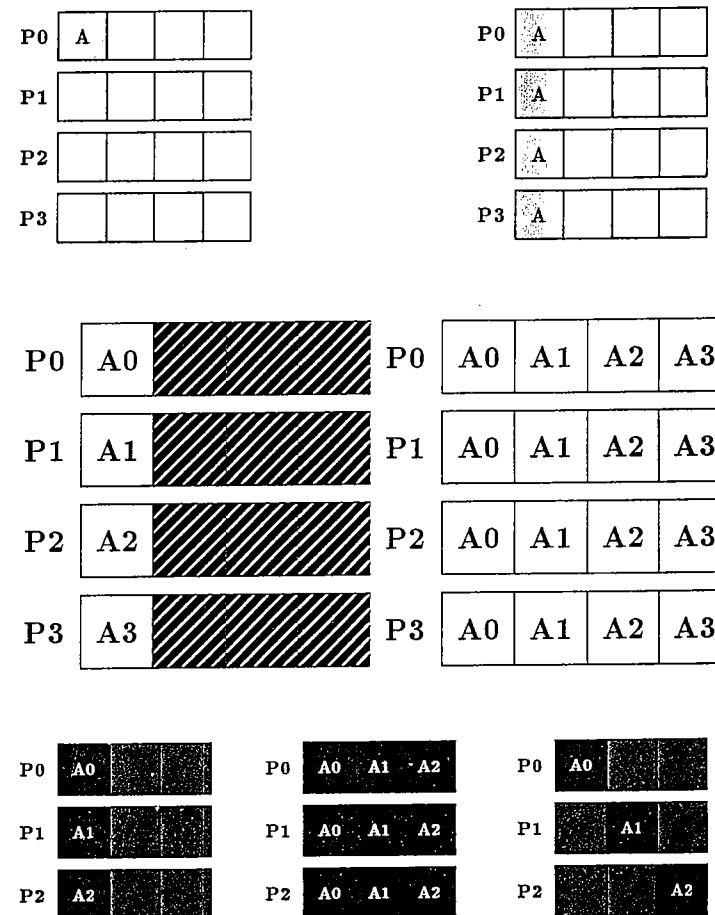


Figure 4.1: A processor grid, programmed by Denis Girou.

It is still possible to change parameters when using `\trafficsign`:

```
\trafficsign[linecolor=green, doublesep=20pt]{School}
```

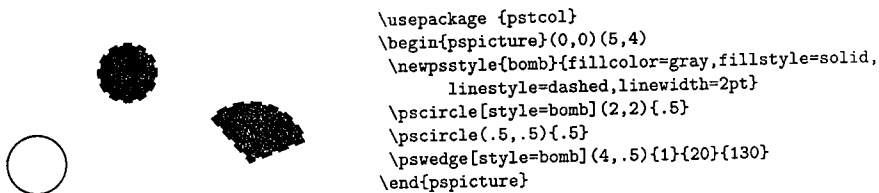

The macro `\showgrid` which we used in many examples earlier in this chapter was defined with the following command:

```
\newpsobject{showgrid}{psgrid}{subgriddiv=1,griddots=10,gridlabels=6pt}
```

Alternatively, one can associate a new style name with a set of graphical parameters which you want to use many times:

```
\newpsstyle{name}{par1=value1,...}
```

We can now use the new *name* as a parameter for any objects drawn subsequently, with the style key:



4-10-1

A considerably more powerful and complicated technique is to construct entirely new compound objects using the `\pscustom` command:

```
\pscustom[settings]{commands}
```

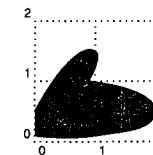
For a complete explanation of this system, see van Zandt (1993a); we give a brief overview here.

The effect of `\pscustom` is to create a new object that can be affected by fill commands or be used as a complete path in PostScript terms. Although most PSTricks objects can be used in `\pscustom`, only the simple ones that merely draw a path (like `\psline` and `\psarc`) are recommended. The extra graphic parameter `liftpen` determines whether or not the object being drawn is joined to the previous current point by a straight line, and whether the current point is taken as a first coordinate of a curve. The possible values of `liftpen` are

- 0 A line is drawn from the current point to the start of the next object; if that is a curve or line, the current point is taken as its first coordinate;
- 1 A straight line is drawn from the current point to the start of the next object;
- 2 The current point has no effect on the curve. This means that two consecutive curves are not joined and thus the space between is not filled.

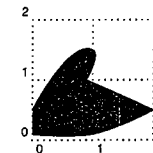
These three types of behavior are seen in the following:

4-10-2



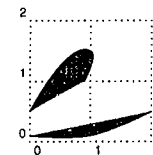
```
\usepackage {pstcol}
\begin{pspicture}(0,0)(2,2)\showgrid
\pscustom[fillstyle=solid,
fillcolor=gray]{%
\pscurve(0,.5)(1,1.5)(.9,1)
\pscurve[liftpen=0](2,.5)(1,.1)(0,.1)
}
\end{pspicture}
```

4-10-3



```
\usepackage {pstcol}
\begin{pspicture}(0,0)(2,2)\showgrid
\pscustom[fillstyle=solid,
fillcolor=gray]{%
\pscurve(0,.5)(1,1.5)(.9,1)
\pscurve[liftpen=1](2,.5)(1,.1)(0,.1)
}
\end{pspicture}
```

4-10-4



```
\usepackage {pstcol}
\begin{pspicture}(0,0)(2,2)\showgrid
\pscustom[fillstyle=solid,
fillcolor=gray]{%
\pscurve(0,.5)(1,1.5)(.9,1)
\pscurve[liftpen=2](2,.5)(1,.1)(0,.1)
}
\end{pspicture}
```

In addition to the PSTricks graphic objects, a new set of commands is available that are analogous to the PostScript operators of the same names; there are the setup commands `\newpath`, `\gsave`, `\grestore`, `\msave`, and `\mrestore`, the positioning command `\moveto(coordinates)`, the drawing commands `\stroke` and `\fill`, and the transformation commands `\translate(coordinates)`, `\scale(num1 num2)`, and `\rotate(angle)`. `\msave` and `\mrestore` save and restore *just* the coordinate system, while `\gsave` and `\grestore` are much more drastic. In Fig. 4.2, Denis Girou used custom commands to program a random walk (the random number generator is by Donald Arseneau).

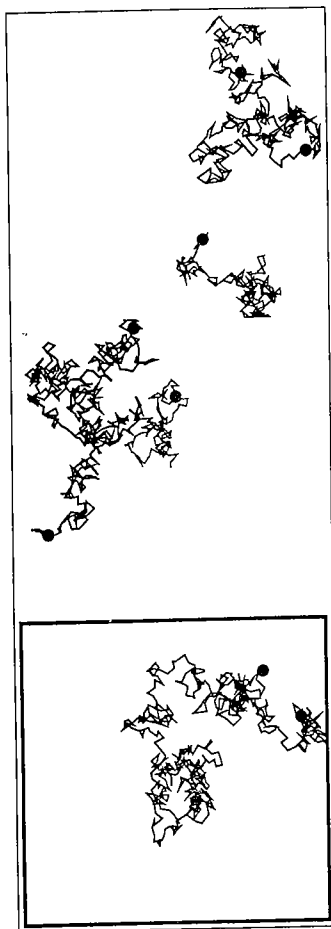


Figure 4.2: A random walk in PSTricks, programmed by Denis Girou.

```

\usepackage {pstcol,multido}
\input random.tex
\newdimen\X \newdimen\Y\newdimen\Coord
% #1 is number of steps
\newcommand{\RandomWalk}[1]{%
\setrandim\X{Opt}{1pt}
\setrandim\Y{Opt}{1pt}
\psdots[dotstyle=*,dotsize=6pt]
(\pointless\X,\pointless\Y)
\pscustom{%
\moveto(\pointless\X,\pointless\Y)
\multido{i=1+1}{#1}{%
\MoveCoordinate{\X}
\MoveCoordinate{\Y}
\lineto(\pointless\X,\pointless\Y)}}}
% Computation of new horizontal
% or vertical position of point
\newcommand{\MoveCoordinate}[1]{%
\setrandim\Coord{-0.03pt}{0.03pt}
\advance #1 by \Coord
\ifdim#1>1pt #1=1pt \fi
\ifdim#1<0pt #1=0pt \fi
}
\newcommand{\randomdemo}[2]{%
\psframe[linewidth=0.6mm](1,1)
\rput{#1,#2}{\RandomWalk{300}}
\psset{linecolor=red}
\rput{#1,#2}{\RandomWalk{200}}
\psset{linecolor=green}
\rput{#1,#2}{\RandomWalk{100}}
}
\psset{unit=5,dimen=middle}
\framebox{\begin{pspicture}(1,3)
\randomdemo{0}{0}
\randomdemo{0}{1}
\randomdemo{0}{2}
\end{pspicture}}

```

4-10-5

These extra tools allow us to fill the area between two curves without drawing a line or curve joining them:



4-10-6

```

\usepackage {pstcol}
\begin{pspicture}(0,0)(2,1)\showgrid
\pscustom{%
\pscurve(0,.6)(1,.9)(2,.6)
\gsave
\pscurve[liftpen=1](2,.4)(1,.1)(0,.4)
\fill[fillstyle=solid,fillcolor=lightgray]
\grestore
\pscurve[liftpen=2](2,.4)(1,.1)(0,.4)
\end{pspicture}

```

This trick works because `\gsave` and `\grestore` effectively stop the second `\pscurve` line from being drawn at all, by throwing its work away after it has been done. The `\fill` command, however, still works and obeys the path in existence when it was called. The `\pscurve` is then repeated to get the lower line actually drawn.

There are further PSTricks versions of analogous PostScript commands:

```
\lineto(coordinates)
```

```
\rlineto(coordinates)
```

```
\curveto(x1,y1)(x2,y2)(x3,y3)
```

```
\rcurveto(x1,y1)(x2,y2)(x3,y3)
```

but these require some care, since a current point must exist. Users who understand PostScript properly may also pass through raw PostScript code, with some tools provided to access PSTricks concepts; this is documented in van Zandt (1993a).

Although many drawings can be programmed using the safer tools of the main PSTricks package, the ability to build new layers of objects over PSTricks increases one's scope; Fig. 4.3 shows the use of a polygon language written in PSTricks by Denis Girou (the `pst-poly` package).

4.10.2 PSTricks programming examples

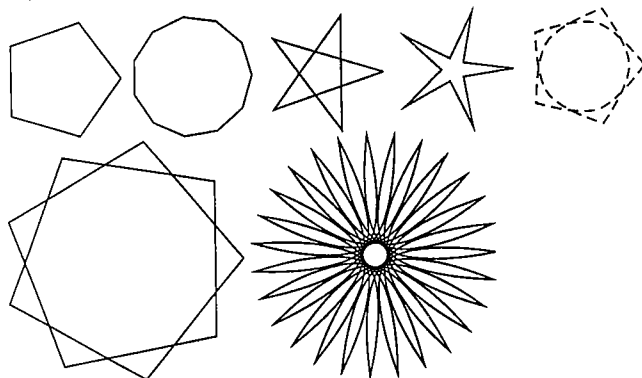
Here we show some of PSTricks' range, demonstrate the advantages of using a programming language for drawing pictures, and explore the various tools for simplifying and modularizing the code to make it more readable. We are especially grateful to Denis Girou for his input to this section, both in personal exchanges and in published examples. The lines of code in these pictures are numbered for ease of discussion.

```

\usepackage {pstcol,pst-poly}
\pstRegularPolygon
\pstRegularPolygon[RPolyNbSides=11]
\pstRegularPolygon[RPolyOffset=2]
\pstRegularPolygon[linecolor=red,RPolyIntermediatePoint=0.2]
\pstRegularPolygon[linestyle=dashed,RPolyCurves,RPolyOffset=2,
  RPolyIntermediatePoint=0.75]

\pstRegularPolygon[unit=2,RPolyNbSides=9,RPolyOffset=2]
\pstRegularPolygon[unit=2,RPolyNbSides=27,RPolyOffset=7,RPolyCurves,
  RPolyIntermediatePoint=0.1]

```



4-10-7

Figure 4.3: A polygon language written in PSTricks by Denis Girou.

In the first picture, a kite drawing from a child's book of colors and shapes, notice that the tail is drawn as a curved node connection between two points, and the bunting is added as labels on that connection. Use of the node feature means that calculating the line and positions along it is left entirely to PSTricks. It is also worth recalling the basic PostScript premise that objects are opaque unless otherwise stated; this means we can give the whole picture a blue background and then overlay solid blocks of color for the shapes. Some items in the picture are regular and repeating, and we take advantage of this to draw the rays of sun using the `\multido` macro; note that we use a T_EX group to localize the effect of the `\psset` that changes color and style.

```

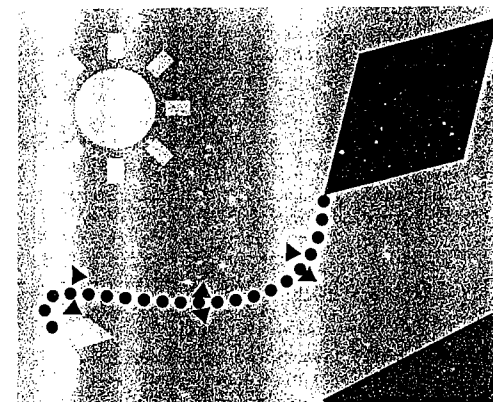
1 \usepackage {pstcol,multido,pst-node}
2 \definecolor{lightblue}{rgb}{.68, .85, .9}
3 \psset{unit=.825cm}\begin{pspicture}(10,8)
4 \psset{fillstyle=solid,linestyle=none,linewidth=0}

```

```

5 \psframe[fillcolor=lightblue](10,8)
6 \pscircle[fillcolor=yellow](2,6){.8} % Sun
7 {% Rays
8 \psset{linecolor=yellow,linestyle=solid,linewidth=.3}
9 \degrees[8]
10 \multido{i=1+1}{8}{\rput{i}(2,6){\psline(1,0)(1.5,0)}}
11}%
12 \pspolygon[fillcolor=green](6,0)(10,2)(10,0)% Grass
13 \psdiamond[fillcolor=red,gangle=-45](8,6)(1.5,2.5)% Kite
14 \rput{45}(8,6){\pnode(-2.5,0){Kitetail}}
15 \rput{-10}{(.8,1.5){\psdiamond[fillcolor=yellow](.6,.1)(.6,.3)}}
16 \rput{-80}{(.8,1.5){\psdiamond[fillcolor=yellow](.6,.1)(.6,.3)}}
17 \pnode(.8,1.5){Tailend}
18 \ncurve[fillstyle=none,angleA=270,angleB=125,ncurvB=.9,ncurvA=1.4,
19   linestyle=dotted,dotstyle=square,linewidth=.25]{Kitetail}{Tailend}
20 \newcommand{\bunting}{\pstriangle(.35,.35)}
21 \psset{fillcolor=red,labelsep=.01}
22 \naput[nrot=115,npos=.15]{\bunting}
23 \nbput[nrot=25,npos=.15]{\bunting}
24 \naput[nrot=75,npos=.4]{\bunting}
25 \nbput[nrot=115,npos=.4]{\bunting}
26 \naput[nrot=115,npos=.7]{\bunting}
27 \nbput[nrot=25,npos=.7]{\bunting}
28 \end{pspicture}

```



4-10-8

In the next child's picture, we again exploit the `strokeout` nature of PostScript solid color blocks to draw the cat's head as a circle, and then superimpose the wall (and on top of that the bricks), so that we do not worry about creating a precise wedge of just over a semicircle.

`\rput` is used extensively to place objects at an angle. The writing on the bricks demonstrates the importance of understanding the reference points of objects that are placed. Since the bricks and their legends are drawn *after* the graffiti, they partly obscure it. We group objects of similar characteristics together and use T_EX's standard grouping to set PSTricks values for the items in that group. We also break the picture into different elements, describe each in a separate macro, and group them into high-level objects. This technique allows us to build up a library of objects, and serves to make the final picture description considerably more readable. The cat sitting on its portion of wall sets its size according to a parameter, so that we can reproduce it several times at different sizes; note how the `\rput` command resets the coordinate system, so that the cat is drawn relative to the position of the `\rput`.

```

1 \usepackage {pstcol}
2 \DeclareFixedFont{\curly}{T1}{pzc}{m}{it}{30}
3 \definecolor{wheat}{rgb}{.96, .87, .7}
4 \definecolor{brown}{rgb}{.65, .16, .16}
5 \definecolor{lightblue}{rgb}{.68, .85, .9}
6 %
7 % The cat is designed to appear on a 10 x 10 grid
8 % cat head
9 \newcommand{\Cathead}{%
10 \pscircle[fillcolor=black](5,4.2){2.5}%
11 % ears
12 {%
13 \psset{linecolor=black,fillcolor=pink,linewidth=.05,linestyle=solid}
14 \rput{45}(5,4.2){\pspolygon(2.5,.5)(2.5,-.5)(3.5,0)}
15 \rput{135}(5,4.2){\pspolygon(2.5,.5)(2.5,-.5)(3.5,0)}
16 }%
17 }
18 % eyes, nose and whiskers
19 \newcommand{\Catface}{%
20 \pscircle[fillcolor=yellow](4,5.2){.5}
21 \psdiamond[fillcolor=gray](4,5.2){.2,.5}
22 \pscircle[fillcolor=yellow](6,5.2){.5}
23 \psdiamond[fillcolor=gray](6,5.2){.2,.5}
24 % nose
25 \rput{180}(5,4.6){\pstriangle[fillcolor=pink](.5,.5)}
26 % whiskers
27 {%
28 \psset{linecolor=white,linestyle=solid,linewidth=.1}
29 \rput{5}(5,4.2){\psline(.8,0)(1.8,0)}
30 \rput{15}(5,4.2){\psline(.8,0)(1.8,0)}
31 \rput{165}(5,4.2){\psline(.8,0)(1.8,0)}
32 \rput{175}(5,4.2){\psline(.8,0)(1.8,0)}
33 }%
34 }

```

```

35 % paws with claws
36 \newcommand{\Catpaws}{%
37 \pscircle[fillcolor=black](1,4.2){.8}
38 \pscircle[fillcolor=black](9,4.2){.8}
39 {%
40 \psset{fillcolor=yellow}
41 \newcommand{\clawsizes}{.4,.4}
42 \rput{180}(1,4.4){\pstriangle(\clawsizes)}
43 \rput{180}(1,4.4){\pstriangle(-.45,0)(\clawsizes)}
44 \rput{180}(1,4.4){\pstriangle(.45,0)(\clawsizes)}
45 \rput{180}(9,4.4){\pstriangle(\clawsizes)}
46 \rput{180}(9,4.4){\pstriangle(-.45,0)(\clawsizes)}
47 \rput{180}(9,4.4){\pstriangle(.45,0)(\clawsizes)}
48 }%
49 }
50 % wall
51 \newcommand{\Wall}{%
52 \psframe[fillcolor=brown](0,0)(10,4)
53 }
54 % The whole cat on its wall
55 \newcommand{\Cat}[1]{%
56 {\psset{unit=#1}
57 \Cathead\Catface\Catpaws\Wall}%
58 }
59 % bricks
60 \newcommand{\Bricks}{%
61 \bfseries\large
62 \psset{fillcolor=wheat}
63 \psframe(1,.4)(2.5,1.9)
64 \rput{bl}(1.1,1){\LaTeX}
65 \psframe(5,.4)(6.3,1.9)
66 \psframe(7,.4)(8.5,1.9)
67 \psframe(2,2.2)(3.2,3.7)
68 \rput{bl}{90}(2.6,2.4){\normalsize$e=mc^2$}
69 \psframe(5.3,2.2)(8,3.7)
70 \rput{bl}(5.4,2.8){\textsc{PostScript}}
71 }
72 \begin{pspicture}(10,8)
73 \psset{fillstyle=solid,linestyle=none,linewidth=0}
74 \psframe[fillcolor=lightblue](10,8)
75 \Cat{1}
76 \rput{bl}{5}(1,1){\curly\color{white}Don Knuth Rules OK}
77 \Bricks
78 \rput{7,1}{\Cat{.1}}
79 \rput{.1,2}{\Cat{.15}}
80 \end{pspicture}

```

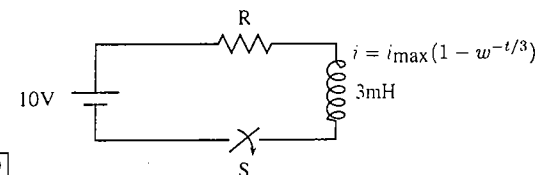


The third complete picture, a circuit diagram, is more complex. We draw it by programming a small language that implements the actions `\Battery`, `\Resistor`, `\Switch`, and `\Inductor`, with the movement commands `\Up`, `\Down`, `\Left`, and `\Right`, in the spirit of pic. Thus the final part of the input is quite simple, apart from the node connection and label commands, which are used in their normal way (some care is needed to handle the angles at which connection lines join and leave each node).

```
\Up{1}
\Battery{A}
\Up{1}
\Right{2}
\Resistor{B}
\Right{1}
\Down{.3}
\Inductor{C}
\Down{.5}
\Left{1}
\Switch{D}
\ncangle[angleA=90,angleB=180,armB=0]{A}{B}
\ncangle[angleA=0,angleB=90,armB=0]{B}{C}
\ncput[ref=1]{~~~~$i=i_{\mbox{max}}(1-w^{-t/3})$}
\ncangle[angleA=-90,armB=0,angleB=0]{C}{D}
\ncangle[angleA=180,armB=0,angleB=-90]{D}{A}
\nput{180}{A}{10V}
```

4-10-9

```
\input{90}{B}{R}
\input{0}{C}{3mH}
\input{270}{D}{S}
```



4-10-10

The new commands are implemented in terms of a current x and y coordinate, which is changed by the movement commands. Thus `\Down` is defined (simplified) in the following code:

```
\newcommand{\Cirdirection}{:}
\newcommand{\Down}[1]{%
  \setcounter{CurY}{-#1}\renewcommand{\Cirdirection}{3}%
}
```

After each movement the current direction of movement is recorded, since this affects how the new objects are drawn. These have a switch (simplified in the following code example²) to check direction. The plain T_EX `\ifcase` command is used to perform a four-way switch among directions. The definition of the resistor is:

```
\newcommand{\Resistor}[1]{%
  \ifcase\Cirdirection % right
    \rput[1]{\CurX,\CurY}{\rnode{#1}{%
      \pszigzag[coilarm=.01,coilwidth=.3](0,.15)(1,.15)%
      \MyBox{1}{.3}}}%
    \addtocounter{CurX}{1}%
  \or % left
    \addtocounter{CurX}{-1}%
    \rput[1]{\CurX,\CurY}{\rnode{#1}{%
      \pszigzag[coilarm=.01,coilwidth=.3](0,.15)(1,.15)%
      \MyBox{1}{.3}}}%
    \addtocounter{CurX}{-1}%
  \or % up
    \rput[b]{\CurX,\CurY}{\rnode{#1}{%
      \pszigzag[coilarm=.01,coilwidth=.3](.15,0)(.15,1)%
      \MyBox{.3}{1}}}%
    \addtocounter{CurY}{1}%
  \or % down
    \addtocounter{CurY}{-1}%
}
```

²Since we may want sizes to be real numbers, `\CurX` and `\CurY` are in fact T_EX dimensions, which we convert back to numbers before use.

```

\put[b](\CurX,\CurY){\rnode{#1}{%
\pszigzag[coilarm=.01,coilwidth=.3](.15,0)(.15,1)%
\MyBox{.3}{1}}}%
\fi
}

```

The macro `\MyBox` is very important; since by itself `\pszigzag` takes no space, it will create a node with no width or height, and connectors will go right to the middle. Therefore we put in some L^AT_EX struts with the `\rule` command to create an invisible box around the zigzag. The PSTricks units are converted to normal T_EX lengths using `\pssetlength`.

```

\newlength{\Cirtemp}
\newcommand{\MyBox}[2]{% width,height
\pssetlength{\Cirtemp}{#1}%
\rule{\Cirtemp}{Opt}%
\pssetlength{\Cirtemp}{#2}%
\rule{Opt}{\Cirtemp}%
}

```

Our final example takes PSTricks into another subject area, that of cartography. We created the following map from the same AutoCAD DXF file as Fig. 1.18; this time the 36 polygons, comprising 9619 separate line segments, were converted (using a simple *ad hoc* conversion program) to a separate coordinate data file for each polygon. The overall map description consists simply of 36 lines of the form:

```

\ProcessVector{moh174}
\ProcessVector{moh170}

```

Since the number of coordinates is so large, many of the PSTricks tools that can read the files (like `\fileplot`) run out of T_EX memory; however, for purely graphical objects like polygons, we have the more efficient and less memory-intensive option of writing Encapsulated PostScript files on the fly, so we instantiate the `\ProcessVector` lines with the definition:

```

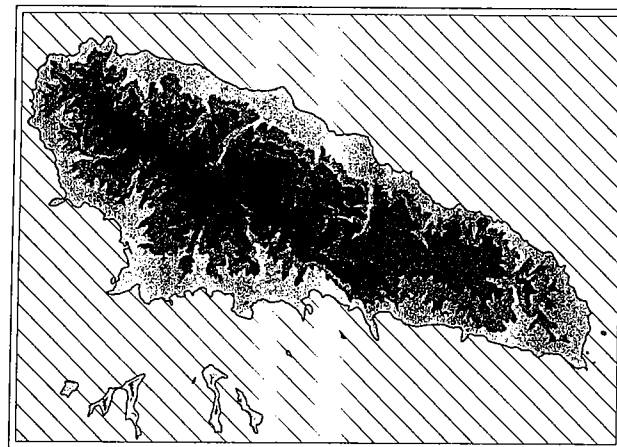
\newcommand{\ProcessVector}[1]{%
\PShtoEPS{#1.eps}{\fileplot{#1.dat}}
\put(0,0){\includegraphics{#1.eps}}%
}

```

The command `\PShtoEPS` takes two arguments: a filename and any pure PSTricks commands (i.e., not text). Instead of embedding the necessary PostScript as `\specials` in the T_EX output, a PostScript file is written directly.

Using the very simple top-level `\ProcessVector` command means that the master file is easily hand-edited; since the AutoCAD file identifies the polygons by their “layer” (the map

is a set of contour lines), we can set the fill color separately for each layer, and so produce the more traditional map (also printed as Color Plate VI).



4-10-11

4.11 Other PSTricks tools

We have not covered all the tricks of which this package is capable; for the sake of completeness, the remaining features are briefly listed here:

- Color definition macros; we have preferred to use the standard L^AT_EX color package, linking it to PSTricks with the `pstcol` package;
- Rotation and scaling macros; again, we have preferred to use the standard L^AT_EX graphics package. Users should be warned that PSTricks defines a macro called `\scalebox` that has different syntax from the standard command, so they should be careful about the order in which packages are loaded;
- Overlays; these macros are used in the seminar package described in Section 9.4.

4.12 Driver configuration for PSTricks

The main L^AT_EX package `pstricks`, and some of the subsidiary ones, need access to various PostScript header files; the user does not need to load them explicitly, but the driver must be capable of doing so. The method for loading header files, and other `\special` communication, is defined in a configuration file named `pstricks.con`. This file must define

the following T_EX macros; a more detailed description is given in the sample configuration provided with PSTricks:

`\pstdriver` Provides the name of the driver
`\pstunit` Defines the units used by the driver when doing verbatim PostScript inclusion.
`\pstverbscale` Writes PostScript code to scale the coordinate system from that when `\pstVerb` is used to that when `\pstverb` is used.
`\pstverb` Includes the argument verbatim PostScript, grouped by `save` and `restore`
`\pstVerb` Includes the argument as verbatim PostScript, *not* grouped by `gsave` and `grestore`.
`\pstheader` Includes the argument as a PostScript header file
`\pstrotate` Writes PostScript code to take a number off the stack and rotate the axes

Thus the configuration for dvips is as follows:

```
\def\pstdriver{Rokicki's dvips}
{\catcode'\="12\gdef\pstverb#1{\special{" #1}}}
\def\pstunit{1bp}%
\def\pstVerb#1{\special{ps: #1}}
\def\pstverbscale{\SDict begin normalscale end}
\def\pstheader#1{\special{header=#1}}
%% In the following, CP is short for currentpoint.
%% The neg is necessary because dvips scales the coordinates by 1 -1.
%% \tx@NET means neg exch neg exch translate.
\def\pstrotate{%
  CP CP translate 3 -1 roll neg rotate \tx@NET}
```

4.13 Summary of PSTricks commands and parameters

4.13.1 PSTricks basic drawing commands

To use the following commands, load the package `pstricks` (or `pstcol` if you need color support); in addition, load `pst-text` if you need the `\pstextpath` macro, `pst-char` if you need `\pscharclip` or `\pscharpath`, and `pst-coil` if you need `\pscoil`. Load `pst-grad` to use 'gradient' fill style.

`\parabola*[settings][arrows](x0,y0)(x1,y1)`
 Draw a parabola that starts at (*x*₀,*y*₀) and has maximum or minimum extent (*x*₁,*y*₁)
`\psarc*[settings][arrows](x0,y0){radius}{angleA}{angleB}`
 Draw a circle segment between *angleA* and *angleB* (counterclockwise)
`\psarcn*[settings][arrows](x0,y0){radius}{angleA}{angleB}`
 As `\psarc`, but the arc is drawn *clockwise*

4.13.1 PSTricks basic drawing commands

`\psbezier*[settings][arrows](x0,y0)(x1,y1)(x2,y2)(x3,y3)`
 Draw a Bézier curve with four control points
`\psccurve*[settings][arrows](x1,y1)...(xn,yn)`
 Draw a closed curve between the points
`\pscharclip*[settings]{text}... \endpscharclip`
 Set the clipping path to the character shapes
`\pscharpath*[settings]{text}`
 The *text* obeys the PSTricks *linestyle* and *fillstyle* parameters; this is effective only when using PostScript Type1 fonts
`\pscircle*[settings](x0,y0){radius}`
 Draw a circle with the center at (*x*₀,*y*₀)
`\pscirclebox*[settings]{text}`
 Draw a circle around the *text*
`\pscoil*[settings][arrows](x0,y0)(x1,y1)`
 Draw a 3D coil from (*x*₀,*y*₀) to (*x*₁,*y*₁)
`\psCoil*[settings]{angle1}{angle2}`
 Draw a coil horizontally from *angle1* to *angle2*
`\pscurve*[settings][arrows](x1,y1)...(xn,yn)`
 Draw an open curve through the points
`\psdblframebox*[settings]{text}`
 Draw a double box around the *text*
`\psdiabox*[settings]{text}`
 Draw a diamond around the *text*
`\psdiamond(x0,y0)(x1,y1)`
 Draw a diamond centered at (*x*₀,*y*₀) with half width (*x*₁,*y*₁) and height (*y*₁)
`\psdots*[settings](x1,y1)(x2,y2)...(xn,yn)`
 Draw dot at each coordinate
`\pscurve*[settings][arrows](x0,y0)...(xn,yn)`
 Draw an open curve but omit the last and first points
`\psellipse*[settings](x0,y0)(x1,y1)`
 Draw an ellipse with centre at (*x*₀,*y*₀), and horizontal and vertical radii of (*x*₁) and (*y*₁)
`\psframe*[settings](x0,y0)(x1,y1)`
 Draw a rectangular frame with corners at (*x*₀,*y*₀) and (*x*₁,*y*₁)
`\psframebox*[settings]{text}`
 Draw a box around the *text*
`\psgrid(x0,y0)(x1,y1)(x2,y2)`
 Superimpose a grid with corners at (*x*₁,*y*₁) and (*x*₂,*y*₂), labeled on the axes starting from (*x*₀,*y*₀); the default extent of the grid is the enclosing `pspicture`
`\psline*[settings][arrows](x0,y0)(x1,y1)...(xn,yn)`
 Draw a line through a series of coordinates
`\psovalbox*[settings]{text}`
 Draw an oval around the *text*

`\pspolygon` *[Settings]* *[Color]* $(x_1, y_1) (x_2, y_2) \dots$ *[Color]*
 Draw a line through the coordinates, and then close the path to make an object that can be filled

`\psshadowbox` *[Settings]* *{text}*
 Draw a box with a shadow around the *text*

`\pstextpath` *[Pos]* *[Style]* *{graphics object}* *{text}*
 Draw *text* along the line defined by the *graphics object*. *pos* determines how the text relates to the path; by default (1), it starts at the beginning of the path; *c* centers the text along the path and *x* makes it finish at the end of the path. *x, y* is the offset for the text in relation to the path. By default the text is offset by 0.7ex above the line. *This macro, and \pscharclip, are not guaranteed to work with every dvi-to-PostScript driver!*

`\pstriangle` *[Settings]* *[Color]* $(x_0, y_0) (x_1, y_1)$
 Draw an isosceles triangle with base centred at (x_0, y_0) , width (x_1, y_1) , and height (y_1)

`\pstribox` *[Settings]* *{text}*
 Draw a triangle around the *text*

`\pswedge` *[Settings]* *[Color]* *{radius}* *{angle1}* *{angle2}*
 Draw a wedge segment between *angle1* and *angle2* (counterclockwise)

`\pszigzag` *[Settings]* *[Arrow]* *[Color]* $(x_0, y_0) (x_1, y_1)$
 Draw a zigzag from (x_0, y_0) to (x_1, y_1)

`\qdisk` (x, y) *{radius}*
 A quick version of `\pscircle` with no optional parameters

`\qline` $(x_1, y_1) (x_2, y_2)$
 A quick version of `\psline`, with no optional parameters, to draw a single line segment

Table 4.5: PSTricks basic graphical parameters.

Parameter	Default	Explanation
<i>General</i>		
<code>unit=dim</code>	1cm	Size of basic PSTricks unit
<code>xunit=dim</code>	1cm	Size of PSTricks unit for <i>x</i> axis
<code>yunit=dim</code>	1cm	Size of PSTricks unit for <i>y</i> axis
<code>runit=dim</code>	1cm	Size of other PSTricks unit
<code>showpoints=true/false</code>	false	If set to <i>true</i> , this shows the control points of most objects; of special interest for those utilizing Bézier curves
<code>origin={coord}</code>	0pt,0pt	Origin of coordinate system for subsequent graphics objects
<code>dimen=where</code>	outer	Where distances are measured from in a closed object; possible values are <i>outer</i> , <i>inner</i> , and <i>middle</i>
<code>swapaxes=true</code>	false	<i>x</i> and <i>y</i> axes reversed
<code>linewidth=dim</code>	0.8pt	Width of lines
<code>linecolor=color</code>	black	Color of lines
<code>fillcolor=color</code>	white	Fill color for objects

PSTricks basic graphical parameters *cont.*

Parameter	Default	Explanation
<code>fillstyle=style</code>	none	Other styles are <i>solid</i> , <i>vlines</i> , <i>vlines*</i> , <i>hlines</i> , <i>hlines*</i> , <i>crosshatch</i> , and <i>crosshatch*</i> . The * versions also fill the background. The <i>pst-grad</i> package adds the extra gradient style for a graded fill and the following keys
<code>gradbegin=color</code>		Starting and ending color of a graded fill
<code>gradend=color</code>		Color at the midpoint
<code>gradlines=int</code>	500	Number of lines in the graded fill. More lines means finer gradation but slower printing
<code>gradmidpoint=num</code>	0.9	Position of the midpoint, as a fraction of the distance from top to bottom (<i>num</i> is between 0 and 1)
<code>gradangle=angle</code>	0	Gradation is rotated by <i>angle</i>
<code>hatchwidth=dim</code>	0.8pt	Width of fill lines
<code>hatchsep=dim</code>	4pt	Gap between fill lines
<code>hatchcolor=color</code>	black	Color of hatching lines
<code>hatchangle=angle</code>	45	Angle of hatching lines
<code>arrows=style</code>	none	Possibilities are listed in Table 4.3 on p. 103; any symbol, or none, can be put at either end of a line
<i>Lines, curves, and boxes</i>		
<code>linestyle=style</code>	solid	Other possibilities are dashed, dotted and none
<code>dash=dim1 dim2</code>	5pt 3pt	Black/white dash pattern for dashed lines
<code>dotsep=dim</code>	3pt	Gap between dots
<code>border=dim</code>	0pt	Draw border of width <i>dim</i> and color <i>bordercolor</i> (next parameter) on each side of a line or curve
<code>bordercolor=color</code>	white	Color of line border
<code>doubleline=true/false</code>	false	Draw lines as double line, separated by <i>doublesep</i> and with color <i>bordercolor</i> between the lines
<code>doublesep=dim</code>		Default is 1.25 × the current <i>linewidth</i>
<code>doublecolor=color</code>	white	Color of lines drawn if <i>doubleline</i> is true
<code>shadow=true/false</code>	false	A shadow is drawn at angle <i>shadowangle</i> , of depth <i>shadowsize</i> and color <i>shadowcolor</i>
<code>shadowsize=dim</code>	3pt	Depth of offset shadow
<code>shadowangle=angle</code>	-45	Angle at which shadow is drawn
<code>shadowcolor=color</code>	darkgray	Color of shadow
<code>linearc=dim</code>	0pt	Radius of arcs drawn at the corners of a box or set of line segments
<code>framearc=dim</code>	0pt	If <i>cornersize</i> is “relative”, then the radius of rounded corners of framing boxes is set to <i>num</i> times the width or height of the frame, whichever is less. <i>num</i> cannot be greater than 1. If <i>cornersize</i> is “absolute”, <i>num</i> sets the radius of arcs of rounded corners
<code>cornersize=type</code>	relative	possible values are “relative” and “absolute” (used in conjunction with <i>framearc</i> , see above)

PSTricks basic graphical parameters *cont.*

Parameter	Default	Explanation
Arcs		
arcsepA= <i>dim</i>	0pt	This and the following two parameters allow one to adjust the angles of <code>\psarc</code> so that the line just touches a line of angle <i>dim</i> that extends from the center in the direction of the angle
arcsepB= <i>dim</i>	0pt	(see previous explanation)
arcsep= <i>dim</i>	0pt	(see previous explanation)
Curves		
curvature= <i>a b c</i>	1.1 0	Lower values of <i>a</i> make the curve tighter; lower values of <i>b</i> tighten the curve if the angle of 3 points is greater than 45° and loosen it otherwise. <i>c</i> determines the slope at each point.
Text frames		
framesep= <i>dim</i>	3pt	Gap between a frame and the enclosed text
boxsep= <i>true/false</i>	true	Whether the T _E X produced includes the size of the frame itself or not
Text labels		
labelsep= <i>dim</i>	5pt	Offset of text in <code>\uput</code> command
Dots		
dotstyle= <i>style</i>	*	Possible styles are listed in Table 4.4 on p. 106
dotsize= <i>dim num</i>	2pt 2	Diameter of a circle or disc is <i>dim</i> plus <i>num</i> times the current linewidth
dotscale= <i>num1 num2</i>	1	Dots are scaled horizontally by <i>num1</i> and vertically by <i>num2</i>
dotangle= <i>angle</i>	0	Angle of rotation for each dot
Arrows		
arrowsize= <i>dim num</i>	1.5pt 2	Arrowhead width is <i>dim</i> plus <i>num</i> times the current linewidth
arrowlength= <i>num</i>		Arrowhead length is this <i>arrowlength</i> times width
arrowinset= <i>num</i>		Inset of the arrow is <i>arrowinset</i> fraction of arrow length
tbarsize= <i>dim num</i>		Width of a t-bar, square bracket or rounded bracket is <i>num</i> × linewidth, plus <i>dim</i>
bracketlength= <i>num</i>		Set height of a square bracket is <i>num</i> × its width
rbracketlength= <i>num</i>		Set height of a round bracket is to <i>num</i> × its width
arrowscale= <i>num1 num2</i>		Scale width of arrows by <i>num1</i> and length by <i>num2</i> . If <i>num2</i> is omitted arrows are scaled the same in both directions
Diamonds and triangles		
gangle= <i>angle</i>	0	Rotation of diamonds and triangles
Grids		
gridwidth= <i>dim</i>	0.8pt	Width of grid lines
gridcolor= <i>color</i>	black	Color of grid lines

PSTricks basic graphical parameters *cont.*

Parameter	Default	Explanation
griddots= <i>num</i>	0	If <i>num</i> is positive, the grid lines are dotted, with <i>num</i> dots per division
gridlabels= <i>dim</i>	10pt	Size of the numbers used to mark the grid
gridlabelcolor= <i>color</i>	black	Color of label numbers
subgriddiv= <i>int</i>	5	Number of grid subdivisions
subgridwidth= <i>dim</i>	0.4pt	Width of lines of subgrid
subgridcolor= <i>color</i>	gray	Color of lines of subgrid
subgriddots= <i>num</i>	0	Number of dots per subgrid division
Coils and zigzags		
coilwidth= <i>dim</i>	1cm	Diameter of coil or zigzag
coilheight= <i>num</i>	1	Length of each coil or zag is <i>coilwidth</i> × <i>coilheight</i>
coilarm= <i>dim</i>	0.5cm	Length of the straight segment at the ends of the coil or zigzag; they can be set separately with the following two parameters
coilarmA= <i>dim</i>	0.5cm	(see previous explanation)
coilarmB= <i>dim</i>	0.5cm	(see previous explanation)
coilaspect= <i>angle</i>	45	Angle of view on 3D object
coilinc= <i>angle</i>	10	Smoothness of coil curves

4.13.2 PSTricks node-drawing commands

Node creators

The following commands, and those in the next section, are loaded with the `pst-node` package.

```

\pnode[repoint](name){text}
  create a node called name, consisting of text; connectors point to the repoint

\Rnode[xy](name){text}
  the same as \pnode, but the reference point is the middle of the box's baseline, plus (x, y)

\pnode[xy](name)
  create a node at (x, y) that takes up no space

\dotnode[settings](x, y)(name){text}
  like \psdots, but makes a node

\fnode[settings](x, y)(name){text}
  like \psframe, but makes a node

\cnode[settings](r)(name)
  create a node consisting of a circle of radius r

\Cnode[settings](r)(name)
  create a node consisting of a circle, using the radius set by the radius graphical parameter (it is
  sometimes useful to set the radius for many circles)

\cnodeput[settings](angle)(x, y)(name){text}
  create a node consisting of text in a circle, like \cput

```

```
\circnode*[settings]{name}{text}
  like \pscirclebox, but makes a node
\ovalnode*[settings]{name}{text}
  like \psovalbox, but makes a node
\dianode*[settings]{name}{text}
  like \psdiabox, but makes a node
\trinode*[settings]{name}{text}
  like \pstribox, but makes a node
```

Node connectors

```
\ncline*[settings]{arrows}{firstnode}{secondnode}
  straight line between nodes
\ncLine*[settings]{arrows}{firstnode}{secondnode}
  straight line between nodes, but labels are placed as if the line went right to the center of the nodes
\ncarc*[settings]{arrows}{firstnode}{secondnode}
  arc between nodes; uses parameter arcangle
\ncdiag*[settings]{arrows}{firstnode}{secondnode}
  using the arm and angle parameters, “arms” start out from each node then are joined by a line;
  corner shape is controlled by the linearc parameter
\ncdiagg*[settings]{arrows}{firstnode}{secondnode}
  as \ncdiag, but the second arm is not drawn
\ncbar*[settings]{arrows}{firstnode}{secondnode}
  draws a line with arms coming off at a right angle to the nodes, at an angle of angleA; the arm length
  is adjusted if necessary
\ncangle*[settings]{arrows}{firstnode}{secondnode}
  draws a connection like \ncdiag, but the angle between arm A and the connector line is forced to
  a right angle
\ncangles*[settings]{arrows}{firstnode}{secondnode}
  like \ncangle, but arm A is joined to arm B by two line segments that meet at a right angle
\ncloop*[settings]{arrows}{firstnode}{secondnode}
  like \ncangles but five line segments are used, the second and forth being loopsize long
\ncurve*[settings]{arrows}{firstnode}{secondnode}
  draws a Bézier curve between nodes, using the ncurv parameters to determine the control point
  positions
\ncircle*[settings]{arrows}{node}{radius}
  draws a circle or part circle of radius radius connecting the node to itself
```

Coil and zigzag node connectors

To use these extra node connections, load the `pst-coil` package.

```
\nccoil*[settings]{arrows}{firstnode}{secondnode}
  connects two nodes with a coil
```

```
\nczigzag*[settings]{arrows}{firstnode}{secondnode}
  connects two nodes with a zigzag
```

4.13.3 PSTricks node connection labeling commands

Labeling based on connector length

```
\ncput*[settings]{something}
  place something on the connector line
\naput*[settings]{something}
  place something above the connector line
\nbput*[settings]{something}
  place something under the connector line
```

Labeling based on distance between nodes

```
\tvput*[settings]{something}
  working on the vertical distance between nodes, place something in the middle of the line
\tlput*[settings]{something}
  working on the vertical distance between nodes, place something to the left of the line
\trput*[settings]{something}
  working on the vertical distance between nodes, place something to the right of the line
\thput*[settings]{something}
  working on the horizontal distance between nodes, place something in the middle of the line
\taput*[settings]{something}
  working on the horizontal distance between nodes, place something above the line
\tbput*[settings]{something}
  working on the horizontal distance between nodes, place something below the line
```

Labeling nodes

```
\nput*[settings]{angle}{name}{something}
  place something next to the node name, at a distance of nodesep, in the direction angle from the
  centre of the node
```

Table 4.6: PSTricks graphical parameters for node connectors

Parameter	Default	Explanation
<code>offset=dim</code>	0pt	Offset of connection point to a node
<code>nodesep=dim</code>	0pt	Border around nodes at which connectors stop
<code>nodesepA=dim</code>	0pt	Border around first node
<code>nodesepB=dim</code>	0pt	Border around second node
<code>arcangle=angle</code>	8	In \ncarc, angle between the arc and a straight line drawn between nodes
<code>angle=angle</code>	0	Angle at which connectors hit the nodes

PSTricks graphical parameters for node connectors *cont.*

Parameter	Default	Explanation
angleA= <i>angle</i>	0	Angle at which a connector hits the first node
angleB= <i>angle</i>	0	Angle at which a connector hits the second node
arm= <i>dim</i>	10pt	Length of the line segment where the connector joins the nodes
armA= <i>dim</i>	10pt	Length of the line segment where the connector joins the first node
armB= <i>dim</i>	10pt	Length of the line segment where the connector joins the second node
loopsize= <i>dim</i>	1cm	Length of line segments for \ncloop
ncurv= <i>num</i>	0.67	Distance to Bézier control points in \nccurve; lower values give tighter curves; the distance from the node to the first control point is half <i>ncurv</i> × the distance between the two end points
ncurvA= <i>num</i>	0.67	as <i>ncurv</i> but for first node only
ncurvB= <i>num</i>	0.67	as <i>ncurv</i> but for second node only
boxsize= <i>dim</i>	0.4cm	half the width of the enclosing box of \ncbox and \ncarcbox

Parameters for node labels

ref= <i>ref</i>	c	Sets the reference point for labels
nrot= <i>rot</i>	0	Rotation of label text; if the angle is preceded by :, it is measured with respect to the connector line; the letter abbreviations in Table 4.2 on p. 102 are available, so :U is commonly used to align text on the connector line
npos= <i>num</i>		Position along the length of the connector line where a label is placed; each connector line has one or more segments, and the value of <i>npos</i> +1 determines the segment on which the label is set; see the PSTricks manual for the default values
shortput= <i>type</i>	none	Determines whether short codes are available for labeling connectors; possible values are none, nab, tabl or tab; see p. 114
tpos= <i>num</i>	0.5	Proportion of the distance between nodes at which labels are placed on a connector
mnode= <i>type</i>	R	(for matrices) the default node type; possibilities are R (\Rnode), r (\rnode), C (\Cnode), f (\fnode), p (\pnode), circle (\circnode), oval (\ovalnode), dia (\dianode), tri (\trinode), dot (\dotnode), and none
emnode= <i>type</i>	none	(For matrices) the type of node created for empty cells in a matrix
name= <i>name</i>		(For matrices) the name of a node; parameters like this can be set in square brackets in the cell
nodealign= <i>true/false</i>	false	(For matrices) whether baselines of nodes pass through the centre of nodes
mcol= <i>l/r/c</i>	c	(For matrices) alignment of a node within a matrix cell

PSTricks graphical parameters for node connectors *cont.*

Parameter	Default	Explanation
mnodesize= <i>dim</i>	-1pt	(For matrices) if positive, nodes are forced to be this size
rowsep= <i>dim</i>	1.5cm	(For matrices) gap between rows
colsep= <i>dim</i>	1.5cm	(For matrices) gap between columns

4.13.4 PSTricks drawing commands comparable to node connectors

```
\pcline*[settings][arrows](x1,y1)(x2,y2)
\pccurve*[settings][arrows](x1,y1)(x2,y2)
\pcarc*[settings][arrows](x1,y1)(x2,y2)
\pcbar*[settings][arrows](x1,y1)(x2,y2)
\pcdiag*[settings][arrows](x1,y1)(x2,y2)
\pcangle*[settings][arrows](x1,y1)(x2,y2)
\pcloop*[settings][arrows](x1,y1)(x2,y2)
\pczigzag*[settings][arrows](x1,y1)(x2,y2)
\pccoil*[settings][arrows](x1,y1)(x2,y2)
```

4.13.5 PSTricks tree-drawing commands

Load the package pst-tree if you want to use these commands.

```
\pstree{node}{subtrees}
  draws a node and subtrees connected to it
\psTree{rootnode}{subtrees}\endpsTree
  an “environment” form of \pstree
\Tn
  null tree node
\Tspace{dim}
  leaves gap of dim before next level
\TC*[settings]
  tree node like \Cnode node
\TR*[settings]{something}
  tree node like \Rnode node
\Tcircle*[settings]{something}
  tree node like \circnode node
\Tc*[settings]{dim}
  tree node like \cnode node
\Tdia*[settings]{something}
  tree node like \dianode node
\Tdot*[settings]
  tree node like \dotnode node
```

`\Tf*[Settings]`
tree node like `\fnode` node

`\Tfan*[Settings]`
draws a triangle with a top corner of the predecessor node

`\Toval*[Settings]{something}`
tree node like `\ovalnode` node

`\Tp*[Settings]`
tree node like `\pnode` node

`\Tr*[Settings]{something}`
tree node like `\rnode` node

`\Ttri*[Settings]{something}`
tree node like `\trinode` node

`\skiplevel*[Settings]{n}` nodes or subtrees
misses out entire levels in a particular subtree

`\skipleve*[Settings]{n}` nodes or subtrees
skips *n* levels

Table 4.7: PSTricks graphical parameters for trees

Parameter	Default	Explanation
<code>bbd=<i>dim</i></code>		Set lower bounding box to <i>dim</i>
<code>bbh=<i>dim</i></code>		Set upper bounding box to <i>dim</i>
<code>bbL=<i>dim</i></code>		Set left bounding box to <i>dim</i>
<code>bbr=<i>dim</i></code>		Set right bounding box to <i>dim</i>
<code>edge=command</code>		Node connector used to join tree nodes
<code>fansize=<i>dim</i></code>	1cm	Size of base for <code>\Tfan</code> tree node
<code>levelsep=<i>dim</i></code>	2cm	Distance between successive levels in a tree; the * makes the dimension be in addition to the size of the nodes (levels are normally a fixed distance apart)
<code>showbbox=true/false</code>	false	Draw a dotted frame showing the enclosing rectangle of trees
<code>thislevelsep=<i>dim</i></code>		Like <code>levelsep</code> but applies only to current tree
<code>thisreefit=tight/loose</code>		Like <code>treefit</code> but applies only to current tree
<code>thisreenodesize=<i>dim</i></code>		Like <code>treenodesize</code> but applies only to current tree
<code>thisreesep=<i>dim</i></code>		Like <code>reesep</code> but applies only to current tree
<code>tndepth=<i>dim</i></code>		Minimum depth of tree node labels
<code>tnheight=<i>dim</i></code>		Minimum height of tree node labels
<code>tnpos=l/r/a/b</code>	b	Position of tree node labels relative to the node (left, right, above, below)
<code>tnsep=<i>dim</i></code>		Gap between tree node labels and the node (by default the same as <code>labelsep</code>)
<code>treefit=tight/loose</code>	tight	If “tight”, <code>reesep</code> is the minimum distance between nodes on any level; if “loose”, <code>reesep</code> is the distance between the enclosing bounding boxes of subtrees
<code>treeflip=true/false</code>	false	Does a mirror image of the tree, flipping the nodes

PSTricks graphical parameters for trees *cont.*

Parameter	Default	Explanation
<code>treemode=R/L/U/D</code>	D	Direction of tree growth (right, left, up, and down)
<code>treenodes=dim</code>	-1pt	If positive, sets a fixed size for tree nodes, regardless of content
<code>treesep=dim</code>	0.75cm	Distance between successive nodes in a tree
<code>xbbd=dim</code>		Increase lower bounding box by <i>dim</i>
<code>xbbh=dim</code>		Increase upper bounding box by <i>dim</i>
<code>xdbl=dim</code>		Increase left bounding box by <i>dim</i>
<code>xbbr=dim</code>		Increase right bounding box by <i>dim</i>

4.13.6 PSTricks plotting commands

Load the package `pst-plot` if you want to use these commands.

`\fileplot*[Settings]{file}`
Reads data from *file* and plots it. Supports only line, polygon, and dot styles

`\dataplot*[Settings]{command}`
Plots data previously saved with `\readdata` or `\savedata`

`\savedata{command}[data]`
Saves the *data* in `\command` for processing by `\dataplot`; the data must be enclosed in square brackets

`\readdata{command}{file}`
Reads data from *file* and saves it in `\command` for processing by `\dataplot`

`\listplot*[Settings]{file}`
Reads data from *file* and plots it, in any style, but does less checking to generate valid PostScript

`\psplot*[Settings]{xmin}{xmax}{function}`
Plots a function *f*(*x*); this is written in the PostScript language, using *x* as the dependent variable

`\parametricplot*[Settings]{tmin}{tmax}{function}`
Plots a function (*x*(*t*); *y*(*t*))

Axes

`\psaxes*[Settings]{arrows}{x0,y0}{x1,y1}(x2,y2)`

Draws a set of graph axes, with an intersection at (*x₀*,*y₀*); (*x₀*,*y₀*) and (*x₁*,*y₁*) default to (0,0)

`\psxlabel`
Macro with one parameter that can be redefined to set the font and size of labels on the *x* axis; thus if we say `\renewcommand{\psxlabel}[1]{\tiny #1}` sets the labels in the “tiny” font

`\psylabel`
Macro to redefine to set the font and size of labels on the *y* axis

Table 4.8: PSTricks graphical parameters for plots.

Parameter	Default	Explanation
plotstyle=style	line	Valid values are dots, line, polygon, curve, ecurve, ccurve
plotpoints=int	50	Number of points used in a plot
<i>Axes</i>		
axesstyle=axes/frame/none	axes	
labels=all/x/y/none	all	On which axes are labels set?
showorigin=true/false	true	Are labels set at the origin?
tickstyle=full/top/bottom	full	Whether ticks extend above and below the axis line
ticks=all/x/y/none	all	On which axes are ticks drawn?
ticksiz=dim	3pt	
Ox=num	0	Value of label at x origin
Oy=num	0	Value of label at y origin
Dx=num	1	Increment of labels on x axis
Dy=num	1	Increment of labels on y axis
dx=dim	0pt	Distance between labels on x axis (if the value is 0pt, the current x unit is used)
dy=dim	0pt	Distance between labels on y axis (if the value is 0pt, the current y unit is used)

4.13.7 PSTricks 3D commands

Load the package pst-3d if you want to use these commands.

`\psshadow*[settings]{text}`
 Draws a shadow on the *text*

`\pstilt{degrees}{text}`
 Places *text* tilted

`\ThreeDput( $x_0, y_0, z_0$ ){object}`
 Places *object* at coordinate (x_0, y_0, z_0) , displayed according to the current viewpoint

Table 4.9: PSTricks 3D graphical parameters.

Parameter	Default	Explanation
Tshadowsize=size	1	Length of shadow
Tshadowcolor=color	lightgray	Color of shadow
Tshadowangle=angle	60	Angle of shadow
viewpoint=x y z	1 -1 1	Position of observer looking at the object origin
normal=x y z	0 0 1	Vector orthogonal to the plane of the 2D object, that specifies its position in 3D space
embedangle=angle	0	Rotation around the axis through the reference point of the object in the direction of the positioning vector

CHAPTER 5

The Xy-pic package

Xy-pic is a general-purpose drawing package based on T_EX. It works smoothly with most formats, including L^AT_EX, A^MS-L^AT_EX, A^MS-T_EX, and plain T_EX. It has been used to type-set complicated diagrams from numerous application areas, including category theory, automata, algebra, geometry, neural networks, and knot theory. Xy-pic's generic syntax lets you use a consistent mnemonic notation system that is based upon the *logical* construction of diagrams by the combination of various elementary *visual* components. Macros can also be written that combine these basic elements consistently to form higher-level structures specific to the intended application.

Xy-pic was originally written by Kristoffer Høgsbro Rose (Rose, 1992). Over the past few years extensive revisions and extensions have been made in collaboration with Ross Moore (Rose, 1995; Rose and Moore, 1995).

5.1 Introducing Xy-pic

The Xy-pic system is built around an object-oriented drawing language called the *kernel*: this is a notation for composing “objects” with “methods” that correspond to the meaningful drawing operations on the object.

The kernel supports the following basic graphic notions (see Section 5.3):

- *Positions* can be specified in various formats. In particular, user-defined coordinates can be absolute or relative to previous positions, objects, object edges, or points on connections.
- *Objects* can have several forms, e.g., circular, elliptic, and rectangular, and can be adjusted in several ways, even depending on the *direction* of other objects. In particular, an object can be used to *connect* two other objects.

Enhancements to the kernel, called “options”, have two main varieties: *extensions* (see Section 5.4) add more objects and methods to the repertoire (such as “curving” and “framing”), while *features* (see Section 5.5) provide notations for particular application areas (like “arrows,” “matrices,” “polygons,” “lattices,” “knots”). In general, extensions provide visual components, whereas features add domain-specific notations for their logical composition.

This chapter gives examples of Xy-pic’s use in various application areas. In this “teach by-example” approach, it serves as a complement to the *Xy-pic User’s Guide* (Rose, 1995), which introduces the most used features, and the *Xy-pic Reference Manual* (Rose and Moore, 1995), which describes the syntax of all Xy-pic commands and their parameters. A study of our examples should put you in an excellent position to start drawing your own diagrams; we hope it will also convince you of the beauty, power, and flexibility of the Xy-pic package.

5.2 A first example of Xy-pic code

Xy-pic consists of various modules. If you are not sure which ones to load, it is probably best to load “a large set”, as follows:¹

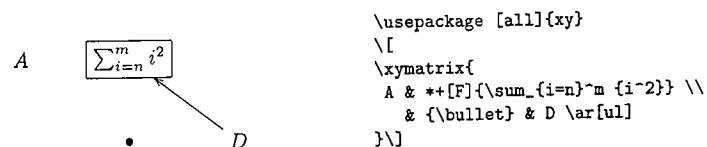
```
\usepackage[all]{xy}
```

On the other hand, once you know enough about Xy-pic to identify which functions you want to use, then you can specify only the extensions or features that are actually needed, for instance:

```
\usepackage[curve,arrow,cmatrix]{xy}
```

The example above uses the *curve* extension and *arrow* feature, tuned to produce \special commands understood by Thomas Kiffe’s CMacTeX Macintosh port of T_EX programs.

To get an idea of the philosophy on which Xy-pic is based, let us first look at how we “construct” an Xy-picture. To make things relatively easy, we consider a matrix-like diagram. As explained in more detail in Section 5.5.2, the principal way to create a diagram is with the command `\xymatrix{spec}`, where *spec* is the specification of the *matrix entries*, which, in general, are aligned in *rows* and *columns*. Just as in a `tabular` environment, entries inside a row are separated by ampersands and successive rows are separated by `\\`.



```
\usepackage [all]{xy}
\l
\xymatrix{
A & **[F]{\sum_{i=1}^m {i^2}} \ll
& {\bullet} & D \ar[u]
}\l
```

5-2-1

¹For formats other than L^AT_EX, use the command `\input xy` followed by `\xyoption{all}`. The `all` option loads the *curve*, *frame*, *tips*, *line*, *rotate*, and *color* extensions as well as the *matrix*, *arrow*, and *graph* features. Any other features or extensions needed must be loaded separately.

This example has two rows of three columns and shows a good deal about how Xy-pic interprets commands.

- By default, entries inside Xy-pic environments are typeset in mathematics mode, using “text style”, and are centered.
- In many cases you may not start entries with a bare macro name—such names must be enclosed in braces or be otherwise “protected.”
- As in a `tabular` environment, empty entries at the end of rows can be omitted if not referred to.
- Elements can be addressed by their *relative* (“logical”) position in the diagram; thus `\ar[ul]` draws an arrow from the “current” position to the matrix cell “one up and one to the left.”
- The *format* and *shape* of an element can be customized by specifying an “entry modifier” (e.g., “[F]” tells Xy-pic to frame the entry).

5.3 Basic constructs

A thorough knowledge of how Xy-pic interprets the various commands will let you exploit its many functions fully. It will also help you understand the subtleties of the various extensions and features introduced in later sections.

A kernel Xy-picture is enclosed in an `xy` environment:²

```
\begin{xy}...\end{xy}
```

The location at which an Xy-pic object is being “dropped” is called its “position.” In fact, in most cases only the coordinates or shape of the “current position” are set.

5.3.1 Initial positions

The simplest form of Xy-pic position is called *absolute*, written `<X,Y>`. The coordinates *X* and *Y* are the offsets *right* and *above* the origin of the picture, which thus lies at `<0cm,0cm>`. Simple arithmetic operators can be used to position the current point. A comma is used to separate one position from another:

```
UL UR
5,5
DL DR
```

5-3-1

```
\usepackage {xy}
\l[\begin{xy}
0*(DL) ,+/r1cm/*{DR}
,<0cm,1cm>*{UL} ,<1cm,1cm>*{UR}
,(5,5)*{5,5}
\end{xy}\l
```

²When using Xy-pic with formats other than L^AT_EX, use `\xy... \endxy`.

The above example illustrates various ways to specify coordinates.³ In particular, 0 (zero) is a shorthand for the origin, and $+/\!r\,1\text{cm}/$ moves right by 1 cm. The next two points $\langle 0\text{cm}, 1\text{cm} \rangle$ and $\langle 1\text{cm}, 1\text{cm} \rangle$ are explicit X,Y coordinates. Finally, the middle point (5,5) uses the *default coordinate system* with units of 1 mm for usual Cartesian x - and y - axes starting from (0,0). We say more about the “*” operator later.

In the next example the units of the coordinate system are defined explicitly by setting them to 5 mm in X and Y using the `:` operator. This means that all further dimensionless coordinate pairs refer to multiples of 5 mm. You can add or subtract lengths from a given position. In particular, for the right-hand part of the diagram below—starting on the third line of code—we first offset the coordinate with respect to the origin 0 by moving four units (i.e., 2 cm) to the right. Call this new location “NO” (for “new origin”—the quotation marks indicate that this is to be a name) using the “=” operator. We use this name to calculate the location at which we want to drop the text object UR and name it “SUR” (for “saved upper right”). Finally, we use this “SUR” twice, each time subtracting a coordinate specification, to obtain locations at which the text UL and DR are to be dropped.

```

\usepackage {xy}
\[\begin{xy} 0;<5mm,0mm>:
0*[DL] , (2,0)*[DR] , <0cm,1cm>*[UL] , (2,2)*[UR]
,0+(4,0)="NO"*{\mathtt{DL}}
,"NO"+(2,2)="SUR"*{\mathtt{UR}}
,"SUR"-<1cm,0cm>*\mathtt{UL}}
,"SUR"- (0,2)*{\mathtt{DR}}
\end{xy}\]

```

UL UR UL UR

DL DR DL DR

5-3-2

5.3.2 Making connections

The effects of the *connect* operator, `**`, can be quite complex. To a first approximation, it “connects” the *current* and *previous* positions (c and p respectively). As a simple example, let us connect some of the locations in an earlier diagram.

```

\usepackage {xy}
\[\begin{xy}
0*[DL];<1cm,1cm>*[UR]**@{-}
,<0cm,1cm>*[UL];<1cm,0cm>*[DR]**@{=}
\end{xy}\]

```

UL UR

DL DR

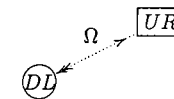
5-3-3

Here the connection operation, `**@{-}`, typesets a connection using the special `@{-}` kernel object, which connects as a solid line. The Xy-pic documentation lists the connections initially provided; new ones can be created with the command `\newdir`. Most have mnemonic names: `@{.}` connects with a dotted line, `@{--}` with a dashed line, `@{=}` with a double line, etc. Note the use of the semicolon operator, which *swaps* the positions p and

³Note that the comma delimiter appears *before* the following item rather than *after* the preceding one. This is a particularly useful device in developing a picture, since whole lines can be easily “commented out” while retaining valid syntax overall.

c . Here it has the effect of moving the position in c , namely 0, into p , then c is immediately overridden with a new position for the next object (remember that connectors with the `**` operator are drawn from p to c). The example also reveals that the drop operator `*` does not place a default margin around objects. Rather, such space is created implicitly, by inserting one or more `+` modifiers between the `*` and the brace opening the object; more on this later.

You can combine several drop operations and the question mark (?) operator lets you specify the location where something is to be drawn “along” a just typeset connection in a coordinate-independent way:



5-3-4

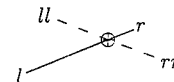
```

\usepackage [frame]{xy}
\[\begin{xy}
0**[o][F]{DL};<2cm,1cm>**[F]{UR} **@{.}
?*<@{<<}>?(0.5)*!/_3mm/{\Omega} ?>>*@{>}
\end{xy}\]

```

The two objects are enlarged with the `+` modifier, and the `[o]` makes the *shape* of the first object round instead of the default rectangular (we have added the `[F]` modifier, defined by the frame extension, to highlight this fact). On the second line, the `?` operator lets us position objects at a given place along the last “connection” (here the dotted line between circle and frame). First, a left double arrow tip is placed at the starting end (`?<`), i.e., the end given by the position p when the connection was made. Next, near the middle (`?(0.5)`) and 3 mm above the line, we write an Ω (by shifting, as described later). Some length units away from the finishing end of the line, i.e., at the position c when the connection was made, a right arrow tip is output (positioned by `?>>`). The `>` and `<` notations accumulate, somewhat analogously to the `+`, `++`, etc. operators that alter the object margin.

Finally, we calculate the intersection of two lines with the `?!@` operator:



5-3-5

```

\usepackage {xy}
\[\begin{xy}
0**+{l}="l" ;<2cm,8mm>***+{r}="r" **@{-}
,<4mm,10mm>+{ll} ;<25mm,2mm>+{rr} **@{--}
?!{ "r"; "l" }*{\oplus}
\end{xy}\]

```

First the two lines (l , r) and (ll , rr) are typeset and the end points are stored with names “ l ” and “ r ” respectively. Using the syntax `?!{<XY1>;<XY2>}` (where $\langle XY1 \rangle$, $\langle XY2 \rangle$ are two positions), the intersection with the last connection is computed as the place to drop the $\text{\LaTeX}$ $\oplus$ symbol.

5.3.3 Dropping objects

We have already used *objects* in most of the examples in the previous section. More precisely, objects are the arguments of the *drop* `*` and *connection* `**` operators; they are usually the elements that are actually output into the picture. As seen above, objects always include a

brace pair { . . . } specifying what is to be dropped. The part preceding the opening brace, called the *modifier*, allows fine adjustments to specify exactly how the object is to be placed.

The “!” *shift* modifier lets you move the reference point of an object from its initial central position to somewhere else within the object’s bounding box:

```
\usepackage [frame]{xy}
\[\begin{xy}
  (0,0)*@{o}*!UL{Box1}*!frm{-}*@{x}
  ,(20,0)*@{o}*!RD{Box2}*!frm{-}*@{x}
\end{xy}\]
```

Box1

Box2

5-3-6

The first box has its upper left corner (!UL) positioned at the current position (0,0), while the second has its lower right corner (!RD) at current position (20,0). In either case, the location of *c* remains unchanged, since the times (x, typeset with @{x}) and open circle (o, typeset with @{o}) overlap in the picture. This example also illustrates the use of the modifier @ to request a “directional” object from the kernel library, using the contents of the braces as a mnemonic abbreviation (a fact we have already used silently on several occasions).

Shifting an object can be compared to the *skew* position operator, which also uses the “!” symbol but occurs after a dropped object or position specification:

```
\usepackage [frame,arrow,curve]{xy}
\[\begin{xy}
  (0,0)*@{*}*!F{Box1}!UL="a",*@{x}
  ,(20,0)*@{*}*!F{Box2}!RD*@{x}
  ,\ar@/^20pt/"a", \POS"a"!DR(.8)*@{+}
\end{xy}\]
```

Box1

Box2

5-3-7

Here the current point *c* is translated to the location specified on the box (as can be seen by comparing the positions of *c* before (•) and after (x) applying the ! operator). However, the “extents” of the position remain those of the box itself, as can be seen from where the curved arrow places its arrow-tip. The difference between shifting and skewing highlights the essential difference between position operators and modifiers: only the former can change the current position, while the latter can be used to change the extents, adjustment, etc., of the rendered object relative to the reference point at which the object is “anchored.” Modifiers have no effect on the current position after the object has been dropped.

Several more commands are available to specify the vector amount of the shift or skew. We have already seen how to obtain a position based on the current object by using the vectors R right, L left, U up, D down, and combinations thereof. These denote the offsets to the corners of the current (rectangular) object from the reference point; C is the offset to the center. These can be further refined using “factors”; e.g., the !DR(.8) in the previous example specifies where to put the + sign (@{+}). Also useful is a specific distance in the *current direction*, as set by the most recent connection using **. The notation /3mm/ refers to the vector that is 3 mm long and oriented along the current direction. Similarly, the notation /_3mm/ has the same length but is oriented 90° clockwise of the current direction

(this is how we positioned the Ω in an earlier example), while /_3mm/ is the vector in the counterclockwise direction.

Among the object modifiers are some that change the size and shape of an object’s bounding box (called the *edge* in Xy-pic jargon). Initially this is the typeset size of the object itself, but it can be changed using the *grow* and *shrink* modifiers, “+” and “-”. These add or subtract a fixed amount, called the *object margin*, to effectively create or reduce a margin of space around the object. The default value for the object margin is 3 pt, but this can easily be changed. The *set size* modifier, “=*wid,ht*”, lets one set the width and height to any specified values. Without such a specific value, “=” is used to square the shape, i.e., equalize the width and height to the smaller of their existing values; similarly, “+=” equalizes to the larger. Modifiers are always interpreted sequentially, from left to right.

```
\usepackage [frame]{xy}
\[\begin{xy}
  ,(0,0)*+=[o][F]\txt{Box}
  ,(12,0)*+=[o]=<9mm>[F]\txt{Box}
  ,(0,12)*+=[o]+[F]\txt{Box}
  ,(12,12)*+=[o]+[F]\txt{Box}
  ,(0,24)*+=[o]-[F]\txt{Box}
  ,(12,24)*+=[o]--[F]\txt{Box}
\end{xy}\]
```

5-3-8

The bottom line above shows at the left the default equalized circle drawn around the object, and at the right a circle with radius set to 9 mm (more precisely, the height and width are both set to 9 mm, since <9mm> abbreviates <9mm,9mm>). On the middle line we increase the circle’s radius by the object margin, then by twice the object margin. Finally, on the top line we shrink the circle’s radius by the same amounts.

5.3.4 Entering text in your pictures

The \txt object command facilitates entering text strings in Xy-pic pictures. It typesets text in centered paragraph mode, i.e., line breaks can be controlled with the \\ command. The syntax is:

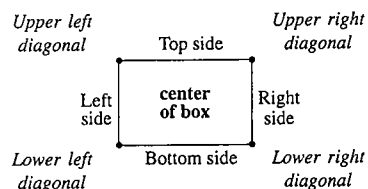
```
\txt <wid>sty{text strings}
```

Both <wid>, the declared width, and sty, the style to be used for typesetting, may be absent. Various possibilities are seen below; note the use of *opposite* vectors to obtain the same positioning by shifting and skewing⁴.

```
\usepackage [frame]{xy}
\[\newcommand{\smbf}{\small\bfseries} \newcommand{\smit}{\small\itshape}
```

⁴The example also illustrates how \newcommand definitions should be made before the xy-environment, but within the outer math display, in order to ensure that those definitions have no effect in later diagrams.


```
\begin{xy}
0*=(22,14)!UR!(-10,-8)[F]\txt\smbf{center\\of box}="box";
"box"+L**!R\txt\small{Left\\side}
,"box"+R**!L\txt\small{Right\\side}
,"box"+D**!U\txt\small{Bottom side}
,"box"+U**!D\txt\small{Top side}
,"box"+LD*Q{*}*!RU\txt<2cm>\smit{Lower left diagonal}
,"box"+RD*Q{*}*!LU\txt<2cm>\smit{Lower right diagonal}
,"box"+LU*Q{*}*!RD\txt<2cm>\smit{Upper left diagonal}
,"box"+RU*Q{*}*!LD\txt<2cm>\smit{Upper right diagonal}
\end{xy}\]
```



5-3-9

5.4 Extensions

We have already used the frame extension, since that is where the [F] modifier is defined. By activating further extension options, other sophisticated graphics functions become available:

- The curve extension lets you draw *curves* and *splines* using quadratic or cubic Bézier curves and B-splines (see Section 5.4.1).
- The frame extension provides a convenient way to draw *frames*, *brackets*, and *filled regions* (see Section 5.4.2).
- The tips extension lets you choose the type of *arrow tip* (Computer Modern or Euler, in addition to the Xy-pic default “technical” style) while *line styles* are controlled by specifying the line extension.
- Both *rotation* and *scaling* are possible with the rotate extension. *Color* and *patterns* and *tiling* effects can be obtained using the color and tile extensions. Graphics images can be imported using the import extension.

Several of these work fully only if your dvi driver supports them. However, it should be emphasized that, by default, the Xy-pic package uses only standard T_EX and METAPOST. Thus it produces completely standard dvi files containing references to the Xy-pic fonts. Output for a specific driver (like dvips) can be generated by loading an appropriate *back end* option that “tunes” the output produced in the dvi file to the indicated driver (by using

\special commands). This does *not* extend the Xy-pic language. When no back end is available, Xy-pic tries to approximate what is requested by using, among other means, the special Xy-pic fonts. In particular, the picture size is identical, so that the choice of back end can never affect the page count, etc.

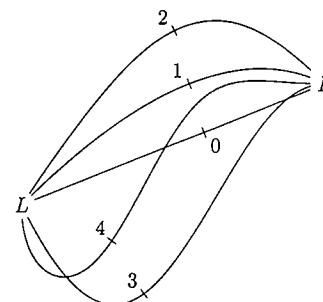
As an example, consider the ps extension, which permits inclusion of PostScript code in Xy-pictures whether or not the driver supports it (the included PostScript will work only if the driver does indeed support it, of course). If the same file is to be L^AT_EX'd on a different system, then the source file need not be changed (except to insert the declaration of which back end to use). With a PostScript-based back end, the Xy-pic fonts are not used, since native PostScript is generated to draw all arrows, tips, etc.⁵ This improves the quality of the printed output, especially for dotted or dashed lines and curves; in addition, L^AT_EX processing time is reduced. We do not discuss back ends further in this book; refer to the Xy-pic documentation for details, including the current list of supported drivers.

5.4.1 Curves and splines

The curve extension makes it possible to draw spline curves. It allows calculation of curved connections along which objects may be dropped; when directional, the objects are aligned with the tangent direction. The basic syntax is

```
\crv setup{control-points}
```

where the argument *control-points* is a list of positions separated by & signs. The object really makes sense only with the connection operator **; as always, the previous and current points *p* and *c* define the end points of the connection.



5-4-1

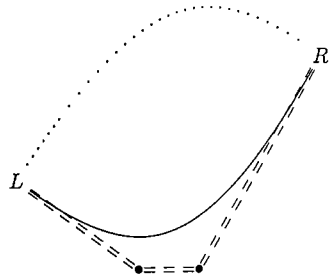
```
\usepackage [curve,frame]{xy}
\begin{xy}
0*[o]+{L};(50,20)*[o]+{R}="R"
**\crv{ }?(.6)*Q{*}*~!UL{0}
,"R" **\crv{(30,30)}?Q{*}*~!DR{1}
,"R" **\crv{(20,30)&(30,40)}?Q{*}*~!DR{2}
,"R" **\crv{(10,-20)&(25,-20)&(40,20)}
?(.4)*Q{*}*~!DR{3}
,"R" **\crv{(0,-10)&(12,-20)&(28,25)&(40,20)}
?(.4)*Q{*}*~!DR{4}
\end{xy}\]
```

This example shows five curves, labeled 0 to 4 to indicate the number of supplementary control points used. This number determines the type of Bézier curve used to connect the start (*L*) to the end (*R*) point. With one or two control points a pure quadratic or cubic

⁵The package also comes with a Type 1 version of the various Xy-pic-specific fonts: these are used only when a PostScript driver is used on a dvi file produced *without* activating a PostScript back end!

Bézier curve is drawn, with the tangents at L and R pointing along the lines connecting these points with the adjacent control point. When three or more control points are specified, then a cubic B-spline is constructed. Note the use of the ? operator to locate places along the connection; in addition to finding the correct location, it also sets the “current direction” to be the tangent direction of the curve *at that point*. The small crosses, set with $*@{+}$, indicate that this directional object has been aligned appropriately.

It is quite simple to visualize the control points by using options along with the $\backslash\text{crv}$ objects.



```
\usepackage {curve}{xy}
\[\begin{xy}
0*+{L}; (50,20)*+{R}
**\crv{~*=<2mm>{.}}(20,30)&(30,35)}
**\crv~Lc{~**@{=}}~*{\bullet}%
(20,-15)&(30,-15)}
\end{xy}\]
```

5-4-2

The first $\backslash\text{crv}$ has no optional part, but uses an object specifier $\sim*$ describing which objects should be drawn along the path. The second curve has the Lc option, specifying that the curve should be drawn together with the control points and the lines connecting them. Furthermore, the positional coordinates are preceded by connector ($\sim**$) and drop ($\sim*$) object specifiers; these determine how the control points are marked and what style to use for the connecting lines.

If you don't have a back end with built-in support for expressing curves, drawing them can consume quite a lot of memory. In such cases it can be wise to use the command $\backslash\text{SloppyCurves}$ or to adjust the tolerance for typesetting the curves with the command $\backslash\text{splinetolerance}{tol}$, whose only parameter is a length tol . A curve is constructed as a series of closely spaced points. The value of tol is the minimum separation of points between which an intermediate point is not calculated—it is *not* the separation of the points themselves. This helps to explain the non-uniform spacing of objects on curves, especially dotted or dashed curves.

The minimum tolerance allowed is 0.2 pt, which is what is used for fine “solid” curves, and is what is actually used if 0 pt is requested. Specifying $\backslash\text{SloppyCurves}$ changes tol to 0.8 pt.

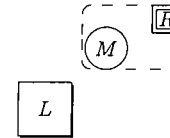
5.4.2 Frames and brackets

The frame extension introduces commands of the type

```
\frm{dim}{spec}
```

where $spec$ is a frame specification and $\langle dim \rangle$ an optional dimension. When the frame is “dropped” (with the $*$ operator), then the object at the current point c is framed, and when it is “connected” connector ($**$ operator), then the rectangle defined by the previous and current objects together is framed. A complete list of possible frames is given in the Xy-pic documentation.

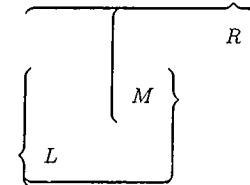
5-4-3



```
\usepackage {frame}{xy}
\[\begin{xy}
0*+{L}*+{\frm<2pt>{-,}}
,(10,10)*+{M}*+{\frm{o}};
(20,15)*+{R}*+{\frm{=}}**\frm{o-}
\end{xy}\]
```

In the picture above we drop the letter L , surrounding it with a rectangular shadow box of size increased by three times the object margin. At the coordinate point $(10, 10)$ we drop the letter M and surround it with a circular frame increased by twice the object margin. Then we transfer the current position to the *previous* position p (; operator) and set the current position c as the new coordinate $(20, 15)$. Here we drop the letter R , surrounded by a double rectangular frame increased by the object margin. Finally, with the connection operator $**$, we draw a rectangular dashed frame with rounded corners surrounding the covering rectangle defined by the objects at c and p .

5-4-4



```
\usepackage {frame}{xy}
\[\begin{xy}
(0,0)*+{L}="1";
p+(15,10)*+{M}*+{\frm{}}% 1
**\frm{\{ } % 2
**\frm{\} } % 3
p+(15,10)*+{R}*+{\frm{ } } % 4
**\frm{\} } % 5
;"1"*+{\frm{-}} % 6
\end{xy}\]
```

When constructing braces using the $frame$ extension, we must track the previous p and current c positions carefully. In the above example we drop the letter L , name the corresponding position “1”, and store it as p (; operator). Then we use p to calculate the coordinates of a new object M . Next we link the objects c and p by a lower bracket (at “1”) and a left-hand brace (at “2”). We now exchange p and c so that the M becomes p and the L becomes c . Then we draw the right-hand brace. Note in particular how the nibble of the brace is aligned for the cases “2” and “3”. Next, we move up to point R , which now becomes c , and draw a left-hand parenthesis (at “4”), then a right-hand brace (at “5”) linking M and R . Once again using the ; operator, we make object R the “previous” p and retrieve the saved position “1” of L , which becomes “current” c . This lets us finally draw the top brace (at “6”).

As a special convenience, rather than separately dropping a frame after an object via $*\text{frm}\langle dim \rangle\{spec\}$, you can obtain the same effect using the object modifier

[F spec: <dim>], or just [F spec] when no <dim> is specified. The simplest variant [F] corresponds to `*\frm{-}`, since this is the one most frequently desired.

5.5 Features

The Xy-pic package comes with an interesting and quite complete set of add-ons that extend Xy-pic for particular application domains. These so-called “features” must be loaded as needed. The present version of Xy-pic provides functionality in the following areas:

- *Arrows.* Draw simple and segmented arrows with configurable marks and labels (Section 5.5.1).
- *Matrices.* Construct two-dimensional matrix-like layouts, in which an object may be addressed by its row and column identifier or by “hops” along the grid from another object (Section 5.5.2).
- *Graphs.* Draw directed graphs, flowcharts, trees, etc. (Section 5.5.3).
- *Two-cells.* Typeset “categorical two-cell” diagrams containing pairs of (labeled) curved arrows (Section 5.5.4).
- *Polygons.* Specify the positions of the vertices of regular polygons (Section 5.5.5).
- *Circles, ellipses, and arcs.* Construct (parts of) circles and ellipses with their minor and major axes aligned in any direction (Section 5.5.6).
- *Lattices and webs.* Draw objects on the regular arrangement of a two-dimensional lattice (Section 5.5.7).
- *Knots and links.* Draw and label knot-like and link-like structures (Section 5.5.8).

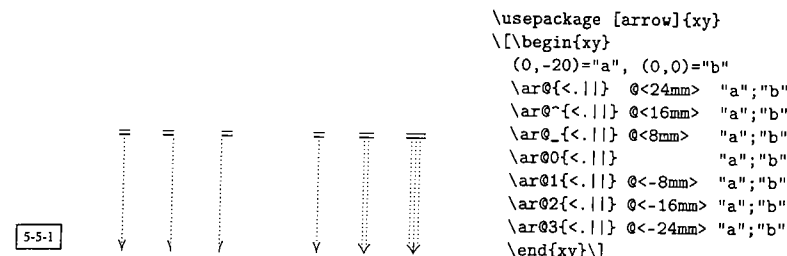
5.5.1 Arrows

Constructing “pretty” arrows was at the very heart of Xy-pic’s original development, as its author wrote (Rose, 1992): “Our first task [with Xy-pic] is to design an arrow such that it looks nice even when very long...”

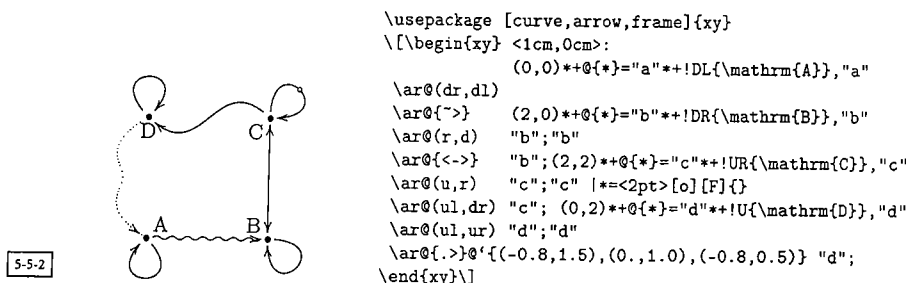
The arrow feature is automatically loaded with the `all` option. In fact, many other extensions depend upon this extension, so it is automatically loaded by those also. Arrows are implemented as an extension of connections to permit an explicit *tail*, *stem*, and *head*. As arrows occur in many places, a simple and convenient syntax is available to typeset them, that is initiated by the `\ar` command. We limit ourselves here to a few simple examples. The versatility of the arrow feature will become more apparent later when we describe the matrix and knot features.

Generally speaking, the style of an arrow is customized with the help of the `@` character. The braced part specifies the tail, stem, and head to be used. Preceding this, the characters `^`, `_`, `0`, `1`, `2`, and `3` stand, respectively, for the above, below, invisible, single, doubled, and

tripled variants, as shown in the example below (note the absence in the center of the `\ar@0` instance):



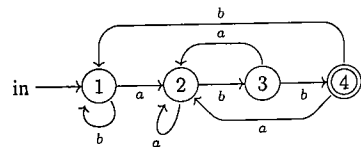
The following example, combining syntax already introduced, shows several useful variations (the curve and frame extensions and the arrow feature must be loaded.)



After setting the coordinate system base unit to 1 cm and dropping the vertex A (named “a” using the `=` operator), we typeset several arrows. Like connections, arrows are typeset from *p* to *c* as determined by the position following the arrow. Both default to the current position *c* before the arrow: this is exploited by the first `\ar` command to draw a loop, i.e., an arrow leaving and reentering A. The outgoing and incoming directions of a loop are indicated using `@(<dir>, <dir>)`, where `<dir>` represents a *direction*, combining at most one of *d* and *u* with at most one of *l* and *r* (other forms are described in the Xy-pic documentation). Next we draw a “variant” arrow from A to a bullet at (2, 0) with label B and name “b”. Notice how the typesetting of the label relative to B is “hidden” with the comma operator so as explicitly to make the final *c* be “b” and not the label. We make a second loop arrow, this time from the explicitly specified “b” to itself, followed by a double arrow to C with yet another loop, this one “broken” in the middle by a 2 pt circle (typeset with `*+<2pt>[o][F]{ }`). The next arrow, linking C to D at (0, 2), is a “curved arrow” with specified tangent directions at either end. To finish, we draw another curved arrow, this time specifying control points explicitly, with the notation `@' {cp, cp, ...}`, where each *cp* is the position of a control point.

First we add with `\newdir` a variant of the `@{>}` tip that is moved a bit so as to make a better arrow tail. The rest is just a matrix designed to have entries at all “interesting” locations. Note how the quote character ‘ is used to specify that an arrow should go via certain cells, passing “below” another arrow (without inserting “turns”, as with the backquote character ‘). For instance, to produce the arrow pointing down from node Σ^L we use that notation for the relative number of “hops” to refer to the nodes one (‘[d]) and three (‘[ddd]) rows below the current one. We use similar code to make the arrow to the right of Σ^G pass below the arrows at cells one (‘[x]) and three (‘[rrr]) hops to the right. The `<<<<` and `(...)` operators fine-tune the placement of the text associated with the arrows by specifying how far from the start point of the arrow the material should appear. The latter form is especially useful, since it allows you to take into account small differences in the lengths of the arrows due to slightly wider elements, so that the typeset material can be aligned with great precision. Also notice the three *setup* parameters: `@=1pc` forces the spacing between columns to be 1 pc; `@*[r]` imposes the adjustment [r] on all entries, which is aesthetically pleasing since all the entries contain one upper-case letter with varying sub- and superscripts; and `@*[o]` gives all entries with a round shape that lets diagonal arrows come just a bit closer.

Finite-state diagrams can also be typeset in a straightforward way:



```
\usepackage [matrix,curve,arrow,tips,frame]{xy}
\[\UseTips
\entrymodifiers={++[o][F]}
\xymatrix @-1mm {
  *+txt[in] \ar[r]
  & 1 \ar@{dr,d1}[]^b \ar[r]_a
  & 2 \ar@{d,d1}[]^a \ar[r]_b
  & 3 \ar[u,l]{}^d \ar[l]_a \ar[r]_b
  & *+o[F=]{4}
  \ar[d1,l]{11mm}/d6mm/'l_u1[11]^a [11]
  \ar[u1,l]{11mm}/u1cm/'l^d[11]_b [111]
}
```

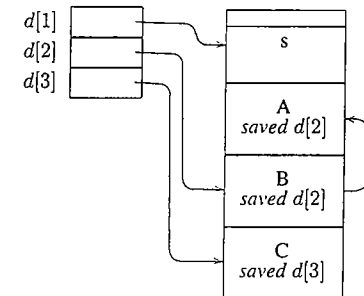
5-5-5

In this kind of diagram,⁷ all states (elements) are enclosed in circles; here we use the `\entrymodifiers` command to specify the default modifier to realize this. To get nice arrowheads on the end of curves, we use Computer Modern tips, and to keep the diagram a little more compact, we reduce the interelement spacing by 1 mm (`@-1mm` before the opening brace of the `\xymatrix` command). Starting an entry with an asterisk (i.e., using the form `*(object)`) overrides the default settings from `\entrymodifiers`; this feature is used in the leftmost cell to eliminate the frame and in the rightmost to typeset a double circle. Note that in the latter case the complete modifier specification had to be given. The only other tricky bit is the use of displacements towards the exterior, adding 6 mm (for *a*) and 1 cm (for *b*) in establishing the locations of the turns.

⁷We based our example on the deterministic finite automaton diagram in Aho et al. (1986), p. 136; another representation of the same diagram can be found in Rose (1995) Section 3.4, and we also used it for example 3-3-6 on p. 73.

As a final example, we can draw stack diagrams with pointers as follows:

```
\usepackage [matrix,arrow,frame]{xy}
\newcommand{\topbar}{\vrule height 0.4pt width 20mm}
\newcommand{\previous}{% turning-width of 15mm
  \save\ar'r[u]/r15mm/'[u][u]\restore }
\newcommand{\saved}[2]{\txt{#1\\emph{saved} } $d[#2]$\}
\newcommand{\bendto}[2]{% creates a bendy arrow, offset 5mm
  \save c!C+/r5mm/\ar 'r#1!C+/l#2/ 'r#1!C #1!C\restore}
\newcommand{\dinout}[1]{% label-offset 11mm
  \save +/l11mm/*{d[#1]}\restore}
\[\begin{xy}
\xymatrix "R" @M=0mm @H=12mm @W=20mm @R=0mm @*[F] {%
  {\txt{\topbar}\s\} } %1,1
  \\\saved{A}{2} %2,1
  \\\saved{B}{2}\previous %3,1
  \\\saved{C}{3} %4,1
  }\turnradius{2mm}\POS(-30,+4)
\xymatrix @M=0mm @H=5mm @W=12mm @R=0mm @*[F] {%
  \dinout{1}\bendto{"R1,1"}{15mm} %1,1
  \\\dinout{2}\bendto{"R3,1"}{17mm} %2,1
  \\\dinout{3}\bendto{"R4,1"}{19mm} %3,1
  }
\end{xy}\]
```



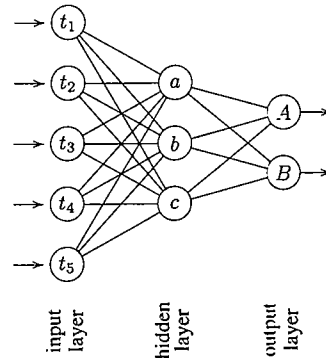
5-5-6

We build up the picture using two `\xymatrix` commands, with text or labels on some entries. Arrows between entries are specified as *excursions*, that use previously defined positions and are enclosed within `\save... \restore` pairs so as not to affect the layout of the subsequent matrix entries. Placing the right-hand stack first and assigning it the name “R” lets us access its cells as positions while the left-hand stack is being built; indeed, some of those cells are used as targets for arrows starting from the left-hand stack. For precise control on object positioning within each `\xymatrix`, one first kills the object margin (`@M=0mm`) and row separation (`@R=0mm`) as part of the matrix *setup*. Furthermore, exact


```

[d]!H{b}      [d]!H{c}
"T" [l] *{left}!U[F]\txt<12mm>{hidden\\layer}
"a" {ul}!I{t_1}      [d]!I{t_2}      [d]!I{t_3}
      [d]!I{t_4}      [d]!I{t_5}
"T" [ll] *{left}!U[F]\txt<12mm>{input\\layer}
}\}

```



S-5-9

First some macros are defined, using names reminiscent of those for the actual neural networks being described. Use of these names in subsequent `\xygraph` allows the logical structure of the components in the diagram to be specified while hiding the details of the lower-level Xy-pic implementation. The command `\newgraphescape` permits new types of nodes to be recognized, following the `!` character. These "node-macros" can even take arguments, as in the above example.

After defining the unit vectors for the two base directions, we place the elements of the neural network. Moving from right to left, we first encounter object *A* on top of *B* (created by `!O`), both with a right arrow pointing away from the main part of the diagram. This is called the *output layer*. Then we have three objects *a*, *b*, *c* (created by `!H`), all connected to *A* and *B*, in an intermediate layer called the *hidden layer*. Finally, we arrive at the *input layer* (using `!I`) with labels of type *t_i*.

When executing the code above, the two uses of the node macro `!O` define the labels "A" and "B", which are referenced by the three `!H`-nodes, which themselves define the labels "a", "b", and "c". These in turn are referenced by the five `!I` commands to draw links among all elements of the input and hidden layers. To typeset the text vertically, we used the `[left]` specifier, which is part of the `rotate` extension and requires a PostScript driver.

It should be evident from this description that these or similar commands can be used in a generic way to typeset all kinds of diagrams featuring neural circuits.

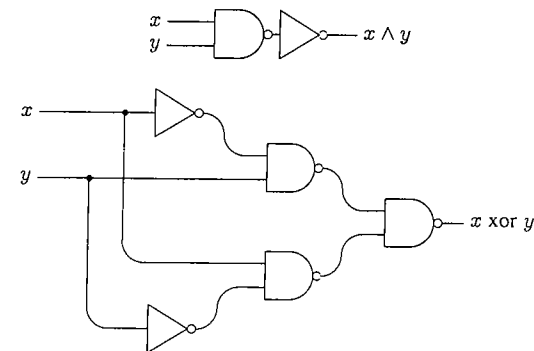
Our second mini-package is a command, `\circuit`, for typesetting logical circuit diagrams. The components we consider are nand (negated-and) and inverter gates, denoted `!N` and `!I` respectively. These are defined in the Xy-pic spirit of being *independent of the*

current direction; however, only one direction, `!R` (for "right") is set up and used. A gate is placed with its output at the current location. It must have a name, which is used for later reference; in addition, the inputs are given names with suffixes *a* and *b*. Notice in the two sample circuits below how few of the gates are placed absolutely—this kind of specification is very modular.

```

\usepackage [graph,curve,arc]{xy}
\newgraphescape{N}[1]{!\save-4mm/-4pt;p+/4mm/:
  (-1,1);(-1,-1)**@{-} ?(.25)="N#1a" ?(.75)="N#1b",
  (-1,1);(0,1)**@{-} ; (0,-1),{\ellipse{}} ; (-1,-1)**@{-},
  (0,0);(1,0)**@{-}!E\cir<2pt>{!C-E="N#1"\restore \POS"N#1"}}
\newgraphescape{I}[1]{!\save-4mm/-2pt;p+/4mm/:
  (-1,1);(-1,-1)**@{-} ?="I#1a", (-1,1);(.667,0)**@{-};(-1,-1)**@{-},
  (0,0)**@{-}!E\cir<2pt>{!C-E="I#1"\restore \POS"I#1"}}
\newgraphescape{B}{!{*O@{*}}
\newgraphescape{R}{!{p+/r4mm/**{}};}}
\newgraphescape{p}[2]{!{#1!{"#2";p+/-/}}
\newcommand{\circuit}[1]{\xygraph{~{0;<10mm,0mm>:<0mm,9mm>::0}#1}}
\[\begin{array}{c}
\circuit{ []!R!N1 ("N1a"([l]x - ?), "N1b"([l]y - ?))
[r]!I2 ("N1" - "I2a") - [r]{x\land y}}\
\circuit{ []x
[rrr]!R!I1 ("I1a"("x" - ?))
[dr]!R!N1 ("I1" -'r[d]"N1a""N1a", "N1b"!p1x y - "N1b")
"I1" [ddd]!R!I2 ("I2a"([l]!puy!B - 'd"I2a""I2a")
"N1" [dd]!R!N2 ("I1a"([l(.5)]!pux!B - 'd"N2a""N2a", "I2" -'r[u]"N2b""N2b")
[urr]!R!N3 ("N1" -'r[d]"N3a""N3a", "N2" -'r[u]"N3b""N3b")
- [r]{x\mathrel{\textrm{xor}}y}}
\end{array}\]

```

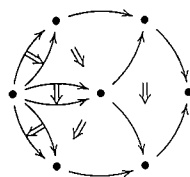


S-5-10

Note how we use a kernel code protected within `\save... \restore` to set up a local coordinate system reflecting the current direction when the gate is placed; we use the `\ellipse`

to the right) and next comes any displacement with respect to the center of the target cell (as this argument is empty, here we point to the center of cell "3,5" itself). The next token `\omit` signifies that we do not want the curved arrows, while the last argument specifies a negative nudge factor and a label *a*. This allows us to position the upper double arrow in a convenient way. The same applies to the rightmost double arrow: it is drawn with the `\rrtwocell` command with `\omit` as first argument, thus also suppressing the curved arrows.

The two-cell feature can be combined with curved arrows to obtain the following interesting layout:



```
\usepackage [all,2cell]{xy}
\UseTwoCells
\l[
\ymatrix @R=2.0pc @C=.8pc {
& \bullet \ar @/_1ex/ [rr]
&& \bullet \ar @/_1ex/ [dr]
\\ \bullet \urtwocell<2> \rrtwocell<2>
\drtwocell<2> \xtwocell[1,3]{\omit
\xtwocell[-1,3]{\omit
&& \bullet \ar @/_1ex/ [ur]
\ar @/_1ex/ [dr]
\rrtwocell\omit
&& \bullet
\\ & \bullet \ar @/_1ex/ [rr]
&& \bullet \ar @/_1ex/ [ur]
}\l]
```

The two `\xtwocell` commands produce the double arrows in the middle of the diagram, pointing towards and away from the central bullet. The rightmost double arrow is the only thing typeset by the final `\rrtwocell` command. The `\omit` on all three of these commands prevents typesetting of the curved arrows.

5.5.5 Polygons

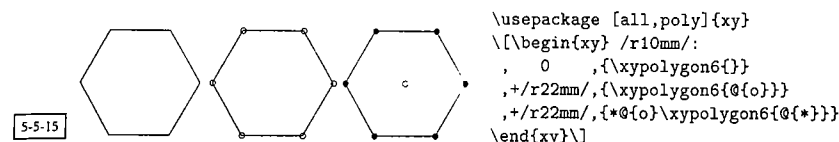
Regular polygons are easy to construct with the `poly` feature. Moreover, one can draw several kinds of non-regular polygons by using a non-square coordinate system. This feature depends on the `arrow` feature and loads it automatically.

The general form of the command for drawing polygons is as follows.

```
\xypolygon<nb>"<pref>"{<swit>...}
```

The mandatory argument *nb* is the number of sides of the polygon. The other mandatory part, represented by the ... above, contains a description of the objects to be deposited at the vertices. The optional prefix "*pref*" provides an explicit name for the polygon so that

you can address it from anywhere inside the `xy` environment. If the first character inside the curly braces is not the tilde character, then the material inside the braces is interpreted as an object to be dropped at each vertex. An argument starting with `~` signifies the presence of one or more switches that modify the form of vertices, sides, or spokes. A few examples will make this clearer.

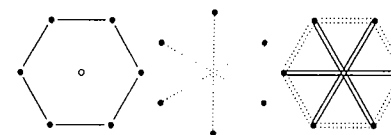


Above we show three forms of a hexagon, the default (no ornaments), then with a simple object at the vertices, and finally with a small open circle in the middle (indicating that the reference point of the polygon lies at its center.)

It is not very difficult to make a few variants on the above by exploiting the switches. First, for the *vertices* we can use `~*{obj}` to drop an object, and `~= {ang}` to indicate the angle of the first drawn vertex. On the other hand, in typesetting the *spokes (sides)*, directionals are specified with `~<{...}` (`~>{...}`), arrow styles with `~<<{arr}` (`~>>{arr}`), and labels and breaks with `~<>{...}` (`~><{...}`).

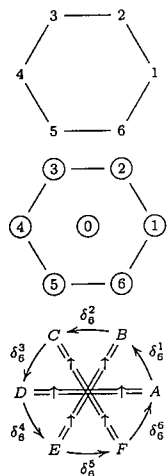
Let us look at all this in practice.

```
\usepackage [all,poly]{xy}
\l[\begin{xy} /r10mm/:
, 0 ,{*@{o}\xypolygon6{~*{\dir{*}}}}
,+/r22mm/,{\xypolygon6{~<{.}~>{}}~= {30}{\dir{*}}}}
,+/r22mm/,{\xypolygon6{~<{=}~>{:}{\dir{*}}}}
\end{xy}\l]
```



By default the object on a vertex is typeset in a box of zero size (see the third command of the previous example where we wrote `\xypolygon6{*}`). However, the first line above shows that, in the case of the switch `~*{...}`, the object at each vertex takes into account the `\objectmargin`. On the second and third lines above we specify both the directionals for the spokes and the sides of the polygons; with an empty declaration (`~>{}`) no sides are drawn. Furthermore, on the second line the declaration `~= {30}` rotates the hexagon by thirty degrees. On the third line we specify the directionals for the spokes and the sides as `~<{=}` and `~>{:}`, respectively.

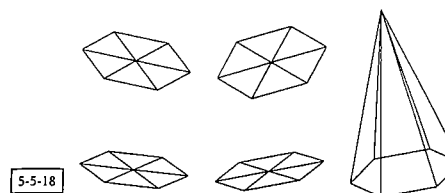
The vertices are automatically named "1", "2", ..., "nb", with the center being identified as "0". For typesetting labels, the command `\xypolynode` corresponds to the actual number of a side, spoke, or vertex at the moment the command is executed. Moreover, the command `\xypolynum` typesets the number of sides. Let us return to our hexagon and see what we can achieve with this knowledge.



```
\usepackage [all,poly]{xy}
\newcounter{node}
%
\l[
\renewcommand{\objectstyle}{\scriptstyle}
\newcommand{\Letter}%
  {\setcounter{node}%
   {\xypolynode}\Alph{node}}}
%
\begin{xy} /r11mm/:
  (2,4.8), {\xypolygon6{*{\xypolynode}}}}
  ,(2,2.5), {*{0}*\cir<5pt>{}}
    \xypolygon6{*{\xybox{}}
  *{\xypolynode}*\cir<2mm>{}}}}
  ,(2,0) , {\xypolygon6{}}
    ~><{\@/_ .5ex/}
    ~>>{\delta^{\xypolynode}_{\xypolynum}}
    ~<<{\@{=}}
    ~<>{\uparrow}
    ~*{\Letter}}
\end{xy}
\l
```

After declaring that objects should be typeset in script style, we first define a command `\Letter` that translates a node number (one through six) into an upper-case letter. The top hexagon shows how node numbers can be typeset at the vertices. In the middle hexagon we circle the vertex numbers, but to achieve this we must specify the compound command sequence as an argument of an `\xybox` command. Finally, at the bottom we construct a rather more complex setup in which sides are curved arrows labeled with the Greek letter δ , with the node number and total number of sides as super- and subscript. The spokes have a double line with an up-arrow in the middle; the vertices have the upper-case letter corresponding to their number, as determined by the `\Letter` macro.

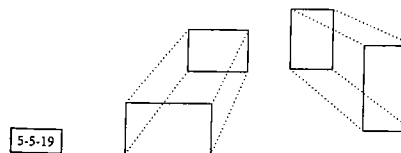
There is one more switch not yet described, namely `~{...}`. This notation allows scaling of the coordinate axes to build non-regular polygons. Perhaps more interestingly, this lets three-dimensional or perspective drawings be simulated, as in the following example.



```
\usepackage [all,poly]{xy}
\l[\begin{xy}/r9mm/:
  (0,0),{\xypolygon6{}}
    ~:~{(1,-.1):(0,.33):~<{-}}}}
  ,(0,2),{\xypolygon6{}}
    ~:~{(1,-.2):(0,.5):~<{-}}}}
  ,(2.5,0),{\xypolygon6{}}
    ~:~{(1,.2):(0,-.3):~<{-}}}}
  ,(2.5,2),{\xypolygon6{}}
    ~:~{(1,.3):(0,-.6):~<{-}}}}
  ,(5,0)="0", +(-.5,3)="T", "0"
  ,{\xypolygon6{~:~{(1,0.2):(0,.4):~<{-}}}}
    ~<>~;"T"***@{-}}}}
\end{xy}\l
```

At the left we show the effect of four different coordinate transformations on a hexagon. To understand the "colon" notation in the argument of the `\xypolygon` commands consider the first hexagon, in which the new base's x axis is set to the coordinate $(1, -1)$ while the y axis is set to $(0, .33)$ so as to generate the skewing effect. The drawing at the right, at position $(5, 0)$, defines an "origin" "0", and an apex "T", at position $(-5, 3)$ with respect to that point. In drawing the hexagon, extra lines to "T" are drawn as a result of the `~<>{...}` switch. This is achieved by the semicolon operator which first loads the vertex position as "previous" object p , and then sets the "current" object c to the top "T", before finally drawing a connector `**@{-}` between them.

Another application is constructing a perspective drawing. The next example also shows how to use the prefixes to identify different polygons.

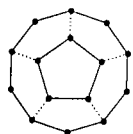
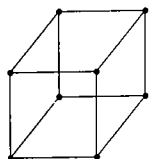


```
\usepackage [all,poly]{xy}
\l[\begin{xy} /r1cm/:
  {\xypolygon4"F"~:~{(0,.6):~}}+{(.8,1.3)}
  ,{\xypolygon4"B"~:~{(0,.7):~}}
  , "F1"; "B1"***@{.} , "F2"; "B2"***@{.}
  , "F3"; "B3"***@{.} , "F4"; "B4"***@{.}
\end{xy}
\qqquad
\begin{xy} /1cm/:
  {\xypolygon4"F"~:~{(0,.6):~}}+{(.8,1.3)}
  ,{\xypolygon4"B"~:~{(0,.7):~}}
  , "F1"; "B1"***@{.} , "F2"; "B2"***@{.}
  , "F3"; "B3"***@{.} , "F4"; "B4"***@{.}
\end{xy}\l
```

In these two drawings, we first make the base asymmetric. For the front rectangle, the y axis is scaled 60% with respect to the x axis, while for the back plane the x axis is scaled 70% and the y axis a further 70%, to give an impression of perspective. Since we labeled the polygons

"F" (front) and "B" (back), we can identify the various vertices with this prefix and their number. It is then easy to connect corresponding vertices on the rectangles to create the visual effect of a three-dimensional box.

A variant of the above allows the transition to nested polygons:

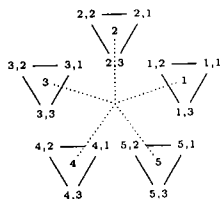


```
\usepackage [all,poly]{xy}
\[\begin{xy} /r10mm/:
  {\xypolygon4"F"{\dir{*}}}
  ,+ (.8,1.0) ,{\xypolygon4"B"{\dir{*}}}
  ,"F1";"B1"*@{-}, "F2";"B2"*@{-}
  ,"F3";"B3"*@{-}, "F4";"B4"*@{-}
  , - (1.0,2.8)
  ,{\xypolygon10"O"{"~"={16}\dir{*}}}
  ,{\xypolygon5"I"{"~"={0.55,0}:}
    ~"={16}\dir{*}}}
  ,"O1";"I1"*@{.}, "O3";"I2"*@{.}
  ,"O5";"I3"*@{.}, "O7";"I4"*@{.}
  ,"O9";"I5"*@{.}
\end{xy}\]
```

5-5-20

Having learned above how to displace rectangles to give the impression of depth, it is now self-evident how to construct a cubical form: we just place two squares a certain distance from each other and then connect corresponding foreground ("F") and background ("B") vertices. In the lower part of the picture we show that it is equally simple to draw a pentagon inside a decagon by merely positioning them at the same point. As we want the top vertices of both polygons to point up, we rotate them by 16°. The inner polygon is scaled down (to 55% in our case) for aesthetics, and finally the vertices of the inner ("I") pentagon are connected to the appropriate vertices of the outer ("O") decagon.

We close this section on polygons by showing how to nest them. Here we draw a triangle at each of the vertices of a pentagon.



```
\usepackage [all,poly]{xy}
\[\renewcommand{\objectstyle}{\scriptscriptstyle}
\begin{xy}
\xypolygon5{"~"={/r12mm/:}
  ~<{\xypolygon3{"~"={0.45,0}:}
    ~*{\xypolynode\xypolynode}
    ~"={30}}
  ~>{}
  ~<{.}
  =<3mm>[o]{\xypolynode}}
\end{xy}\]
```

5-5-21

Inside the "outer" pentagon are triangles at each of the vertices, scaled down 45% with respect to the dimensions of the pentagon. Each vertex of the triangle is labeled with its number but prefixed with \xypolynode, which corresponds to the vertex identifier of the enclosing polygon: labels of the type "4,2" identify the second vertex of the triangle

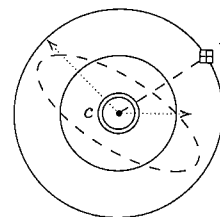
positioned on the fourth vertex of the pentagon. All triangles are oriented at the same 30° angle (~={30}). Furthermore, we have turned off the drawing of the sides and connect the triangles to the center of the pentagon by a dotted line. At the center of each of the pentagon nodes we typeset its number inside an empty disk of radius 3 mm.

5.5.6 Arcs, circles, and ellipses

Circles and ellipses in many forms and styles are available with the arc feature. More generally, circular arcs can be specified that join two points or have a specified tangent at a given point. This feature is based on the curve extension discussed in Section 5.4.1, so you should be sure that it is also loaded. As explained there, drawing curves can overload T_EX's memory quite rapidly, so you are advised to use a back end that directly supports the curve paradigm.

Simple circles and ellipses

Circles of arbitrary radius and line style, including curved arrows, are available with the \ellipse command. In its basic form \ellipse draws a "circle" at the current point. It has a variety of forms, as the next example shows.

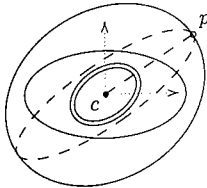


5-5-22

```
\usepackage [all,arc,dvips]{xy}
\[\begin{xy} /r12mm/:
  (0,0)*@{*}="c", *++R{c}
  , (1.2,.8)*[F]@{+}="p"
  , *++LD{p}, "p"; "c"*@{-}
  , {\ellipse<>{}}
  , {\ellipse(0.8){}}
  , 0; (-1,1) ::
  , {\ar@{.}> 0; (1,0)}
  , {\ar@{.}> 0; (0,1)}
  , "p"; "c", {\ellipse(.8){--}}
  , {\ellipse<3mm>{=}}
\end{xy}\]
```

We start with a rectangular coordinate base of 12 mm in both directions. We choose a point *p* (for "previous") and a point *c* (for "current") and position the latter at the origin for convenience. The simplest way to draw a circle is the first \ellipse command, which produces in a circle with its center at the current point and radius $\overline{cp}$. As the argument between the curly braces is empty, the default style (a full line) is used to draw the curve. Note that the circle is not drawn through the object at *p*. The second \ellipse command draws a circle with radius expressed as a fraction of the current coordinate units (80% of 12 mm in our case). We now skew the coordinate system and draw the dotted coordinate axes. Then, with a command similar to the previous one but with a dashed line style, we draw an ellipse. While this shows clearly the effect of the introduction of the skewed base, this base has no effect on the drawing resulting from the last \ellipse command, where the radius is specified in angle brackets.

Since a non-square coordinate system is not always convenient, or even desirable, ellipses can also be specified with other forms of the `\ellipse` command. For these you must in general specify the lengths of the minor and major axes and their alignment. As above, the current point c lies at the center of the ellipse and the vector connecting the previous point p with c provides the alignment of one of the axes.



```
\usepackage [all,arc,dvips]{xy}
\[\begin{xy} 0;/r12mm/:
  {\ar@{.} 0;(1,0)}
  ,{\ar@{.} 0;(0,1)}
  ,(0,0)*@{*}="c",**!RU{c}
  ,(1.2,0.8)*@{o}="p",**!LD{p}
  ,"p";"c"**@{--}
  ,{\ellipse(1.1,.6){}}
  ,{\ellipse(.0.8){}}
  ,{\ellipse<6mm,4mm>{=}}
  ,{\ellipse<.5mm>{--}}
\end{xy}\]
```

5-5-23

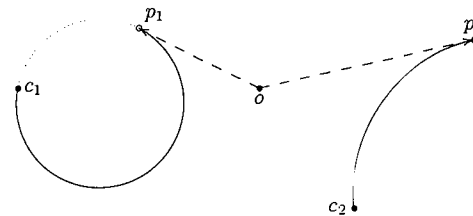
We begin with the same coordinate system as previously and also choose the same points c and p , now of zero size. The syntax with parentheses before the argument uses the coordinate basis as unit length. When numbers are explicitly given, both axes of the ellipse are aligned with the coordinate axes and their lengths are given as a fraction of the base vectors. When the number in front of the comma is absent, one of the axes of the ellipse is aligned with the line $\overline{cp}$ and the perpendicular axis is scaled by the number specified after the comma. Hence, in the examples above, the first ellipse is aligned with the coordinate system and has horizontal axis equal to 1.1 and vertical axis to 0.6 base units. The second ellipse has an axis perpendicular to $\overline{cp}$ with a length of 0.8 base units. The second basic syntax uses angle brackets in front of the curly braces and, in this case, the actual dimensions of the axes are exactly specified as L^AT_EX lengths. With this syntax the ellipse is always aligned with the direction $\overline{cp}$. If the first dimension is absent, then (as in the parenthesized case), $\overline{cp}$ becomes one of the axes while the length specified after the comma is used for the perpendicular axis.

Constructing arcs

Often you are not interested merely in typesetting full circles or ellipses, but also in using circular or elliptical arcs. Generally speaking, two kinds of situations arise: (i) the end points are given but the radius is not determined; (ii) the radius is known but the end points are to be determined.

In fact, in most practical cases the end points are known. Yet, since an infinite number of circular (elliptical) arcs can be drawn through two points, more information is needed. For instance, to determine the arc uniquely, you can supply the tangent of the curve at one of its end points. This is implemented by taking the current direction at the point p , i.e., the direction determined by the latest connection or "up" when there has been no connection.

Let us look at an example.



5-5-24

```
\usepackage [all,arc,dvips]{xy}
\[\begin{xy} 0;/r20mm/:
  (0,0)="o",*@{*},**!U{o}
  ,(-1.0,0.5)="p"*@{o},**!LD{p_1}
  ,{\ar @{->} "o";"p"};
  (-2.,0.)="c",*@{*},**!L{c_1}
  ,"c",{\ellipse_{}} ,{\ellipse^{-}{}}
  ,(1.8,0.4)="p",*@{o},**!D{p_2}
  ,{\ar @{->} "o";"p"};
  (0.8,-1)="c",*@{*},**!R{c_2}
  ,"c",{\ellipse{}}
\end{xy}\]
```

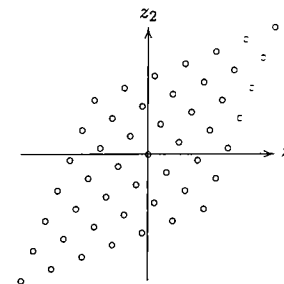
After defining the points "o" and "p" (at p_1), we draw an arrow, at the same time setting the direction $\overline{op}$, to be used as tangent for the circular arc we are going to draw. We then define the current point (c_1), and draw the circle segment from p_1 to c_1 with the `\ellipse` command. Note that an underscore-tagged command draws the segment in the clockwise direction, while the form with the caret (^) draws it counterclockwise (the dotted circle segment). Similarly, we define points p_2 and c_2 and draw the circle segment to the right of the picture. Without further tags, the segments are drawn traversing the arc counterclockwise.

More generally, one can also base the drawing on the tangent at the end points or specify alternate types of curve, such as parabolic or cubic segments, "interpolating" Bézier splines, or "cuspidal" cubics (see Rose and Moore (1995) for more details).

5.5.7 Lattices and web structures

Two-dimensional lattices and other web-like structures can be handled with the web feature. At present its facilities are limited to dropping objects at the intersection points of an integer lattice. This lattice can be skew, i.e., its basis need not be rectangular; any coordinate basis setting defined with Xy-pic can be used.

The simplest command is `\xylattice`, whose four arguments are integers specifying which part of the lattice is to be drawn (in fact they define the positions of the lattice points at the lower left and upper right corners as multiples of the base vectors).

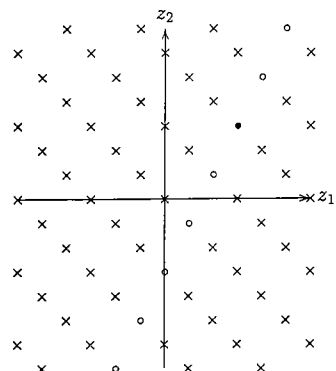


5-5-25

```
\usepackage [all,web]{xy}
\[\renewcommand{\latticebody}{\drop@{o}}
\begin{xy}
  *\xybox{0;<5mm,2mm>;<2mm,5mm>::
  ,0,{\xylattice{-3}{3}{-3}{3}}="S"
  ,{"S"+L \ar "S"+R**!L{z_1}}
  ,{"S"+D \ar "S"+U**!D{z_2}}
\end{xy}\]
```

The command `\latticebody` is expanded at each point of the lattice, so that it can be used to drop objects (`\drop` is like `*` but is a stand-alone command). In the example above we choose an open circle. The command `\xybox` isolates the coordinate base change needed to construct the lattice from the rest of the picture. We define the base vectors of the lattice as $\langle 5\text{mm}, 2\text{mm} \rangle$ and $\langle 2\text{mm}, 5\text{mm} \rangle$, and then draw all lattice points from -3 to $+3$ in the directions of both unit vectors. The resulting box is stored as "S" so that we can reference its dimensions to draw the coordinate axes z_1 and z_2 .

Sometimes one wants to drop objects at specific lattice points. To find the position of a lattice point, the commands `\latticeA` and `\latticeB` give its "coordinate," while the commands `\latticeX` and `\latticeY` return the x and y offsets (in points) with respect to the lattice origin. In addition, the size of the picture can be limited with the `\croplattice` variant, which has four further parameters for specifying a cropping rectangle outside of which no lattice points are shown.



```
\usepackage [all,web]{xy}\usepackage {ifthen}
\[\renewcommand{\latticebody}{%
\ifthenelse{\latticeA=1}{%
\ifthenelse{\latticeB=1}{\drop{\dir{*}}}%
{\drop{\dir{o}}}%}
{\drop{\dir{x}}}%}
\begin{xy}
*\xybox{0;<8mm,4mm>:<4mm,8mm>::
,0,{\croplattice{-6}{-6}{6}%
{-3}{3}{-3.5}{3.5}}%
}="S"
,{"S"+L \ar "S"+R+!L{z_1}}
,{"S"+D \ar "S"+U+!D{z_2}}
\end{xy}\]
```

The four numbers defining the clipping rectangle need not be integers (as seen in the above code): they define the x and y ranges between which lattice points should be typeset as multiples of the lattice's unit vectors. When the object to be typeset at a given grid position depends on its x and y coordinates, then (as remarked above) it is probably more convenient to use the commands `\latticeX` and `\latticeY`. Note that you need to load the `ifthen` package in order to use the `\ifthenelse` construction.

5.5.8 Links and knots

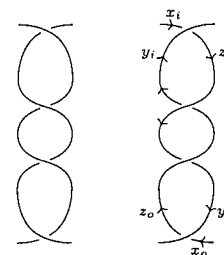
Research about strings has become very popular in many fields of physics and mathematics, and Xy-pic offers an interesting toolkit for constructing arrangements of different kinds of knots, string crossings, and links.

The knot feature provides two kinds of basic building blocks, "crossings," to pass one string above or below another one, and "joins," to connect strings at their end-points. The

knot feature uses the curve extension and arrow feature, so all three should be loaded together. Also, the processing of knot diagrams is the most time-consuming and memory-greedy application discussed so far, so that such diagrams must often be output on individual output pages or as separate Encapsulated PostScript files. These can then be used on subsequent L^AT_EX runs so that one saves both time and memory (see Section 11.6 on p. 455 for a discussion of this technique).

Constructing crossings

Strings "cross" when they come close to one another without actually meeting. Therefore three types of crossings exist: a string can pass *above*, *below*, or *alongside* another string. These possibilities, in various configurations, are systematically summarized in the Xy-pic reference manual; take care to separate the "h" and "v" categories of commands which are building blocks for stacking in the *horizontal* (the current point moves to the top right) and *vertical* (the current point moves to the bottom left) directions.



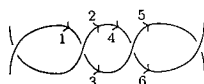
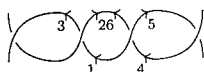
```
\usepackage [all,poly,knot,dvips]{xy}
\[\UseTips
\renewcommand{\objectstyle}{\scriptstyle}
\renewcommand{\labelstyle}{\scriptstyle}
\begin{xy} /r9mm/:
\vover\vcross\vcross\vover-
\end{xy}
\hspace{1cm}
\begin{xy} /r9mm/:
\under<<{x_i}|>{y_i}>>{z_i}%
\vtwist|><<\vtwist
\under-<>{x_o}|>{y_o}>>{z_o}%
\end{xy}\]
```

Some of the combinations of vertical crossings are drawn in this example. On the left we see the effect of the bare commands, while on the right we show how to add labels and arrow tips. Note that the `\(c)twist` and `\(c)cross` variants twist the strings in opposite directions. Note also that either end of the string can be the source or target of a curved arrow, a distinction that becomes important when arrowheads and labels are to be placed. Use of a hyphen (or minus sign) as the first character immediately following the name of any knot piece produces a mirror image of that piece. It may appear identical to another piece, but string orientations will be different (see next example).

Positioning of labels and arrows is controlled by operators `<` and `>`, which should precede the object to be put on the initial and final portion of the crossed string. The operator `|` is used to specify material to be added to the crossing string. When the first character following the `<`, `>`, or `|` is another `>` (or `<`), then an arrow tip pointing in (against) the "natural" direction is typeset at a predetermined position. These placing operators can be repeated as many times as needed. Examine the code for the above example to see how each tip and

label has been placed on the knot crossings, and see Rose and Moore (1995) for a complete listing of the pieces, their orientations, and the default label positions on each part.

Horizontal rows of knot pieces can be combined in complete analogy with the “vertical” knot-building commands above. In the next example we show that the same knot configuration can be built by exchanging commands of the `..cross..` and `..twist..` series. However, one sees from the labelled arrows that these visually identical curves actually have different orientations and label positions.



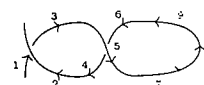
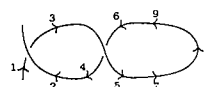
```
\usepackage [all,poly,knot,dvips]{xy}
\[\UseTips
\renewcommand{\labelstyle}{\scriptstyle}
\begin{xy} 0;/r8mm/:
  (0,0)
  ,{\hover\hcross<<<{1}>|>{2}>>>{3}%
    \hcross<<<{4}>|>{5}>>>{6}\hover-}
  ,(0,-3)
  ,{\hover\htwistneg<<<{1}>|>{2}>>>{3}%
    \htwistneg<<<{4}>|>{5}>>>{6}\hover-}
\end{xy}\]
```

5-5-28

Adding joins

Ends of crossing strings can be connected by “joins,” in particular *loops* and *caps*. The next two examples illustrate the use of joins with horizontal or vertical crossing commands of the types `..cross..` or `..twist..`.

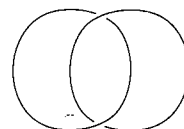
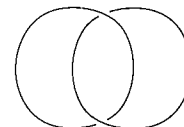
As with the crossing commands, labels, and arrow tips can be placed on the joins. Now there is only one segment, so the `|` operator refers to the middle of the curve while `<` and `>` refer to places *before* and *after* the midpoint. A scale factor, given between square brackets immediately following the command name, can be introduced for each string segment. Moreover, the position of label and tip can be fine-tuned by specifying a value between 0 and 1 between the operator and the object to be typeset, or by adding/subtracting a small amount.



```
\usepackage [all,poly,knot,dvips]{xy}
\[\UseTips
\renewcommand{\labelstyle}{\scriptscriptstyle}
\begin{xy} /r9mm/:
  (0,0)
  ,{\hunder<<<{1}>|>{2}>>>{3}%
    \htwist<<<{4}>|>{5}>>>{6}%
    \hloop<<<{7}>|>{8}>>>{9}}
  ,(0,-3)
  ,{\hunder<<(+0.1)><{1}>|>{2}>>>{3}%
    \htwist<<<{4}>|<(-.2)>|<{5}>>>{6}%
    \hloop<<(+.1)><{7}>|<{8}>>>{9}}
\end{xy}\]
```

5-5-29

Here we have combined two horizontal crossings and one join command. The top drawing shows the default positions for labels and tips; the lower one uses the fine-tuning parameters to position tips and labels. Positive values move along the curve in the “natural” direction. Note the use of the `~` character in the first position of the label “5”, which places the label “above” the arrow while the (default) `_` character places it “below.”

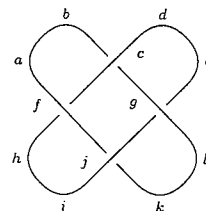


```
\usepackage [curve,knot,graph,dvips]{xy}
\[\xygraph{!{0;/r10mm/:}
  !{\vover}
  [u] !{\hcap[-2]}
  [d] !{\vover-}
  [ruu] !{\hcap[2]}
}
\begin{xy} 0;/r10mm/:
  ,\hcap[-2]\vunder\vunder-
  ,+(1,2),\hcap[2]
\end{xy}\]
```

5-5-30

Since all knot crossings are, by default, bounded by a rectangle of one coordinate unit, and since loop and cap commands do not change the current point, it is convenient to use the graph feature to put together the various pieces of knot crossings and joins. This is shown in the top part of the example above, where the `\vover` and `\hcap` commands position the elements by using “turtle” movements (up, down, left, right). The bottom part presents a variant diagram in which an explicit coordinate move was used to place the final `\hcap`. Note the use of the scaling factors `[2]` or `[-2]`.

Commands are also available to combine pieces in which the strings are basically at angles of 45°, as in this next example.

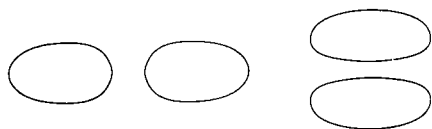


```
\usepackage [curve,knot,arrow,dvips]{xy}
\[\renewcommand{\labelstyle}{\scriptstyle}
\begin{xy} 0;/r8mm/:
  ,{\xcapv-|{a}}
  ,+(0,1),{\xcaph|{b}\xunderh|{c}}%
    \xcaph|{d}\xcapv|{e}}
  ,-(3,0),{\xoverh|{f}}
  ,+(1,0),{\xoverh|{g}}
  ,-(3,1),{\xcapv-|{h}\xcaph-|{i}}
  ,+(0,1),{\xunderh-|{j}}
  ,+(0,-1),{\xcaph-|{k}}
  ,+(0,1),{\xcapv|{l}}
\end{xy}\]
```

5-5-31

The placement of the various pieces in this construction is easy to follow by looking at the labels.

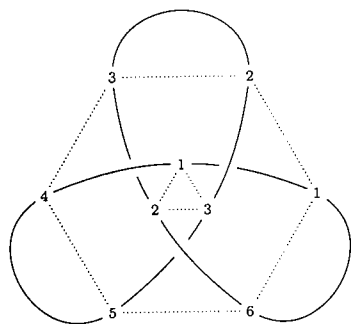
There are also some “bendy” pieces that allow easy connection of these 45° pieces with the vertical and horizontal ones. However, even more general effects are obtained by using a non-square coordinate base.



```
\usepackage [curve,knot,dvips]{xy}
\[\begin{xy} /r2cm/: (0,.5)::
, {\hcap-\huncross\hcap}
, + (1,0), {\vcap\vuncross\vcap-}
\end{xy}\]
```

S-5-32

The greatest variety in the shape of knot pieces is obtained by setting the coordinate base for each piece individually, using the `~` switch. The remaining examples illustrate this technique in conjunction with the `\xypolygon` command from the `poly` option (in the form of the `!P` standout macro of the `graph` feature).

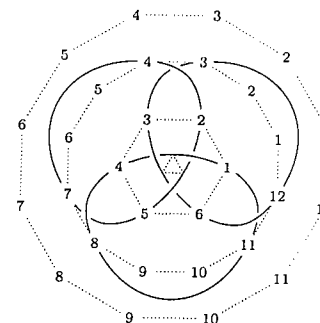


```
\usepackage [all,knot,poly,dvips]{xy}
\[\begin{xy}
\renewcommand{\labelstyle}{\scriptstyle}
\renewcommand{\objectstyle}{\scriptstyle}
\knotholesize{3mm}
\renewcommand{\vcap}[2]{\save 0;#2-#1:#1,
\vcap~{#1+(0,1)}{#2+(0,1)}{#1}{#2}\restore}
\xygraph{!{0;/r5mm/:}
!P3"t"~{>{.}}~*{\xypolynode}}
!P6"h"~{>{.}}~*{\xypolynode}}
!{\xunderv~{"h2"}{"h1"}{"t1"}{"t3"}@(.62)}
!{\xunderv~{"h4"}{"h3"}{"t2"}{"t1"}@(.62)}
!{\xunderv~{"h6"}{"h5"}{"t3"}{"t2"}@(.62)}
!{\vcap{"h3"}{"h2"}\vcap{"h5"}{"h4"}%
\vcap{"h1"}{"h6"}}
\]
```

S-5-33

This three-leaf figure is drawn with the help of the vertices defined by the inner (dotted) triangle (identified by “t”) and the outer (dotted) hexagon (identified by “h”). To make explicit the different steps in the construction, we also show the number of each vertex. Three knot crossings of type `\xunderv` are used, and the `~` syntax permits their precise positioning between pairs of vertices of the triangle and the hexagon (see Section 5.5.5 above for more details). To construct the outer caps we have to renormalize the coordinate base vector, since the `\vcap` command bridges one coordinate unit in the *x*-direction. That is the reason for the base change inside the `\vcap` command, which is isolated from the rest of the diagram with the `\save... \restore` pair. Note the scaling factor of 4.5 inside the hexagon specification and the fine-tuning of the position of the hole for the crossing with

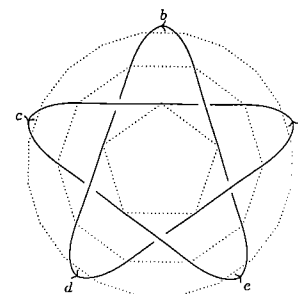
the `@(.62)` syntax and of its size with the `\knotholesize` command.



S-5-34

```
\usepackage [all,knot,poly,dvips]{xy}
\renewcommand{\labelstyle}{\scriptstyle}
\renewcommand{\objectstyle}{\scriptstyle}
\[\xygraph{!{0;/r2mm/:}
!P3"t"~{>{.}}~*{\xypolynode}}
!P6"h"~{>{.}}~*{\xypolynode}}
!P12"d"~{>{(9,0):}}~*{\xypolynode}}
!P12"D"~{>{(13,0):}}~*{\xypolynode}}
!{\xoverv~{"h2"}{"h1"}{"t1"}{"t3"}@(.62)}
!{\xoverv~{"h4"}{"h3"}{"t2"}{"t1"}@(.62)}
!{\xoverv~{"h6"}{"h5"}{"t3"}{"t2"}@(.62)}
!{\vover~{"d4"}{"d3"}{"h3"}{"h2"}}
, "d4"-@{"D5","D6"}"d7",
!{\vover~{"d8"}{"d7"}{"h5"}{"h4"}}
, "d8"-@{"D9","D10"}"d11",
!{\vover~{"d12"}{"d11"}{"h1"}{"h6"}}
, "d12"-@{"D1","D2"}"d3"
\]
```

The drawing above is a little more complex since it involves four polygons. The central triangle and hexagon are similar to those discussed earlier. But we then add a second level of crossings defined by pairs of vertices of the hexagon and the inner dodecagon (identifier “d”). To close the ends of the open strings we draw curves from the relevant vertices using control points on the *outer* dodecagon (identifier “D”) by means of the `@’` syntax discussed in Section 5.5.1.



S-5-35

```
\usepackage [all,knot,poly,dvips]{xy}
\[\begin{xy}
\UseTips
\knotholesize{2mm}
\xygraph{!{0;/r1cm/:}
!P5"p"~{>{.}}~*{\xypolynode}}
!P10"d"~{>{(1.7,0):}}~*{\xypolynode}}
!P20"D"~{>{(2.2,0):}}~*{\xypolynode}}
!{\xunderv~{"d3"}{"d2"}{"p2"}{"p1"}}
!{\xunderv~{"d5"}{"d4"}{"p3"}{"p2"}}
!{\xunderv~{"d7"}{"d6"}{"p4"}{"p3"}}
!{\xunderv~{"d9"}{"d8"}{"p5"}{"p4"}}
!{\xunderv~{"d1"}{"d10"}{"p1"}{"p5"}}
!{\vloop~{"D3"}{"D2"}{"d2"}{"d1"}>|{a}}
!{\vloop~{"D7"}{"D6"}{"d4"}{"d3"}>|{b}}
!{\vloop~{"D11"}{"D10"}{"d6"}{"d5"}>|{c}}
!{\vloop~{"D15"}{"D14"}{"d8"}{"d7"}>|{d}}
!{\vloop~{"D19"}{"D18"}{"d10"}{"d9"}>|{e}}
\]
```

Finally, with the help of the five-fold, ten-fold, and twenty-fold symmetric polygons (shown with dotted lines), we construct the above cinquefoil. The inner pentagon is identified by "p", the middle decagon by "d", and the outer polygon by "D". The relative rotation angle of the vertices and the scaling factor of the polygons are defined inside braces following the !P. . specifier. Furthermore, the ~ syntax on the \xunder v and \vloop commands lets us precisely control the position and size of the crossing and joining elements. The loops of the foil can be made longer or shorter by tuning the scaling factor of the external polygon (a value of 2.2 is used here).

CHAPTER 6

Applications in chemistry, physics, and engineering

Because of its unsurpassed mathematical typesetting, $\TeX$ is undoubtedly used more widely by scientists and mathematicians than by any other group of people. It is not surprising, therefore, that the scientific and engineering community has developed a number of packages to typeset the diagrams and schematics needed in various disciplines. This chapter surveys some of these specialized tools. We must stress, of course, that for many applications, entirely general packages like PSTricks (Chapter 4) or METAPOST (Chapter 3), or slightly more directed packages like Xy-pic (Chapter 5), provide all the functionality you need. The packages described in this chapter address some very specific problem areas.

We start with a brief discussion of typographic conventions in scientific texts. The next two sections describe two contrasting packages for drawing complex chemical structures, $\text{\texttt{X}\text{\textit{M}}\text{\textit{T}}\text{\textit{E}}\text{\textit{X}}$ and $\text{\texttt{ppchtex}}$, while Section 6.4 explores how to construct Feynman diagrams used by physicists. The last three sections turn to electronics, and describe special packages for drawing timing and circuit diagrams.

6.1 Typographical rules for scientific texts

In scientific texts the typographic representation of a symbol often implies a semantic meaning and authors providing camera-ready copy thus need to know the many typographical conventions. While a generic coding scheme (like $\text{\textit{L}}\text{\textit{T}}\text{\textit{E}}\text{\textit{X}}$ or SGML) makes it easier to provide

consistent layout, there are nevertheless a number of pitfalls and even grand masters of $\text{\LaTeX}$, like Knuth (1989), have to admit defeat. We therefore provide a brief summary of the most important rules for composing scientific texts (see also International Union of Pure and Applied Physics (1987); Knappen (1992)).

The most important rule in all circumstances is *consistency*: a given symbol should always be presented in the same way, whether it appears in a formula or in text, on the main line or as a superscript or subscript. An important corollary for $\text{\LaTeX}$ users is: always typeset a symbol in either math or text mode but do not mix the two, even if the results appear to be the same. The reason for this is that with a system like $\text{\LaTeX}$ they may look quite different after a change to the class file or the addition of a new package. For example, when using PostScript fonts, digits in text are taken from the PostScript text face and can look quite different from those used in formulae. Therefore, numbers that refer to a result or part of a formula should be always typeset in math mode, i.e., surrounded by \$.

In scientific texts, many symbols are traditionally typeset as *roman* (upright) characters¹ and may not be understood properly otherwise. Below are the most important such symbols:

- *units*, for example g, cm, s, keV. Note that physical *constants* are usually set in italics, so that units involving constants are mixed roman-italics, e.g., GeV/*c* (where *c* is the speed of light, a constant);
- *chemical elements*, for example Ne, O, Cu, and *elementary particle names*, for example p, K, q, H. To help the typist produce typographically correct texts packages that contain commands representing the various names have been developed. In particular, chemists can use chemsym (see Section 6.1.1), while for high energy physics the PEN (Particle Entity Notation) scheme has been proposed (Goossens and van Herwijnen, 1992);
- standard mathematical functions (sin, det, cos, tan, $\Re$, $\Im$, etc.), for which the built-in $\text{\LaTeX}$ functions should be used;
- numbers;
- names of waves or states (p-wave) and covariant couplings (A for axial, V for vector); names of monopoles (E for electric, M for magnetic);
- abbreviations that are pieces of words (exp for experimental; min for minimum);
- the 'd' in integrands (e.g., dp).

Obeying these typesetting conventions helps the reader understand at first glance the meaning of a symbol. Table 6.1 shows a few examples in which the meaning of a symbol depends on its typographic representation.

¹ With $\text{\LaTeX}$ roman type in mathematics mode can be achieved by the `\mathrm` command.

Table 6.1: The importance of typographic rules in scientific texts.

<i>roman type</i>		<i>italic type</i>	
A	ampere (electric unit)	<i>A</i>	atomic number (variable)
e	electron (particle name)	<i>e</i>	electron charge (constant)
g	gluon (particle name)	<i>g</i>	gravitational constant
l	litre (volume unit)	<i>l</i>	length (variable)
m	meter (length unit)	<i>m</i>	mass (variable)
p	proton (particle name)	<i>p</i>	momentum (variable)
q	quark (particle name)	<i>q</i>	electric charge (variable)
s	second (time unit)	<i>s</i>	c.m. energy squared (variable)
t	tonne (weight unit)	<i>t</i>	time (variable)
V	volt (electric unit)	<i>V</i>	volume (variable)
Z	Z boson (particle name)	<i>Z</i>	atomic charge (variable)

6.1.1 Typesetting chemical symbols

Chemistry texts often contain the names of the elements. From the discussion in the previous section we know that element names should be set in roman type. Mats Dahlgren developed the chemsym package to ensure the typographic correctness of the element names. It provides commands for the 109 chemical elements, using the same names as those in the periodic table of the elements. Some supplementary commands handle chemical groups like `\OH`, `\COOH`, and `\CH`, and the command `\keamtkn` lets you define further customized chemical symbols. As there are already $\text{\LaTeX}$ commands that use some of the element names, the chemsym package renames these commands as follows²:

`\H` → `\h` Hungarian umlaut $\ddot{u}$,
`\O` → `\OO` Danish and Norwegian $\text{\O}$,
`\P` → `\PP` Paragraph sign ¶,
`\S` → `\SS` Section sign §,
`\Re` → `\re` Real part $\Re$ (math mode),
`\Pr` → `\pr` Probability function $\Pr$ (math mode).

In addition, the `Sb` environment of the `amstex` package is renamed `SB`.

To simplify entering chemical formulae, $\text{\LaTeX}$'s super- and subscript symbols `^` and `_` are redefined to be available outside mathematics mode; i.e.,

`2\H_2 plus \O_2` gives `2\H_2\O` in an explosive reaction.

produces:

`2H2 plus O2` gives `2H2O` in an explosive reaction.

²It should be noted that renaming of commands that depend on $\text{\LaTeX}$'s encoding, i.e., commands that might change internal meaning if the font encoding changes, such as `\H` or `\O`, can never be fully successful. To a certain extent these commands remember their old name; for example, they write that name into external files, which produces surprises in the table of contents, etc.

is highly portable. It consists of a set of package files, each specialized to treat a particular kind of chemical structure. These are:

chemstr	basic typesetting commands for atoms and bonds;
aliphat	aliphatic compounds;
carom	vertical and horizontal carbocyclic compounds;
ccycle	bicyclic compounds, etc.;
lowcycle	carbocycles with five members or fewer;
hcycle	pyranose and furanose derivatives;
hetarom	vertical heterocyclic compounds;
hetaromh	horizontal heterocyclic compounds;
locant	helper package producing locant numbers and bond letters needed for producing the $\LaTeX$ manual;
xymtex	$\LaTeX$ 2.09 document style (needed only in producing the $\LaTeX$ manual).

The packages can be loaded individually using $\LaTeX$'s `\usepackage` command, but the chemstr package is loaded automatically by any of the other packages.

6.2.1 General conventions

The $\LaTeX$ command names are based on the standard nomenclature used for organic compounds (International Union of Pure and Applied Chemistry, 1979). The invariant part of a structure, i.e., the base skeleton containing fixed bonds and atoms, is automatically printed with no designation, while the varying part, i.e., substituent groups, additional bonds and atoms, are specified by a set of arguments.

The commands can be subdivided into those for *general* and *specific* use. The specific commands offer convenient shortcuts for drawing a narrow range of structures in a particular category, while the general commands support a wider range at the cost of additional specification. This distinction is also reflected in the command names; e.g., specific commands for drawing certain N-heterocycles would be `\isoindolev` or `\purinev` (with the N atoms automatically typeset), while the name for the underlying general command would be `\nonaheterov` (i.e., describing the geometric structure of the base skeleton without any atoms in predefined positions).

Syntax for specific commands

For carbocycles and heterocycles we have the abstract syntax

$\backslash\text{Com}[opt]\{sublist\}$

where $\backslash\text{Com}$ is a command name corresponding to the compound to be typeset. It is usually suffixed by either *v* or *h* denoting a vertical or horizontal representation of the structure. If alternate orientations are possible, the name might be further suffixed with the letter *i*.

The preselected bond pattern can be modified by the argument *opt* consisting of one or two letter combinations denoting a certain bond pattern. The values allowed and their exact meaning depend on the command; a detailed discussion appears in the manual (Fujita, 1993).

The *sublist* argument lists substituents, with their bonds separated by semicolons if there is more than one bond-substituent pair. A bond is specified by a number giving the atom's position in the structure to which it is attached, optionally followed by a bond modifier (one or more letters) that classifies the type of bond to be used. The substituent is separated from this by two equals signs; e.g., `1D=O`; `3==OCH$_3$`; . . . means that the bond at the atom in position one takes an oxygen atom (O) through a double bond, the bond in position three takes a methoxy group through a single bond, and so on. Atom positions are numbered sequentially, usually in a clockwise fashion starting at the top for vertically oriented structures and on the left for horizontally oriented ones. Possible bond modifiers for the bond at an atom at position *n* are:

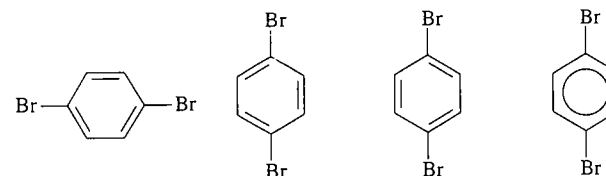
<i>nS</i>	exocyclic single bond at <i>n</i> -atom (equivalent to specifying just <i>n</i>);
<i>nD</i>	exocyclic double bond;
<i>nA</i>	alpha single bond represented as a dotted line;
<i>nB</i>	beta single bond represented as a bold line.

The *nA* modifier relies on the availability of the `\dottedline` command as defined by the epic or eepic packages. If neither package is loaded, the bond appears as a single line of normal width, i.e., it looks the same as if *n* were specified.

If *n* is greater than nine it must be surrounded by two braces, otherwise the commands fail to parse their *sublist* argument correctly. For example, while you can write `1D=O` to attach an O-atom with a double bond to the first atom of a structure, you specify `\{13\}D=O` to attach it to the thirteenth atom. Unfortunately there is no error message—the only indication of a problem is an incorrectly drawn structure.

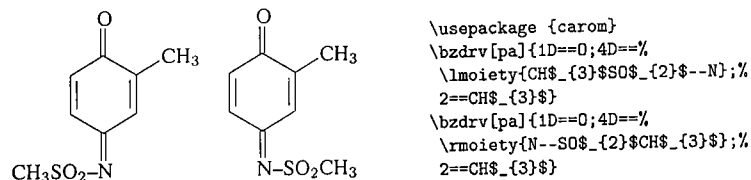
As a first example, let us look at the 1,4-dibromobenzene compound (also known as para-dibromobenzene or *p*-dibromobenzene). We can get several different representations by using the *h* and *v* suffixes on the command name and specifying a bond pattern in the optional argument, e.g.,

```
\usepackage {carom}
\bzdrh{1==Br;4==Br} \bzdrv{1==Br;4==Br}
\bzdrv[1]{1==Br;4==Br} \bzdrv[c]{1==Br;4==Br}
```



The optional parameter [1] designates a left-oriented set of double bonds (the default is [r]) and [c] an aromatic circle. Other legal values for the optional argument for `\bzdrv` and `\bzdrv` are [pa] to [pc] for the three *p*-quinone variants (two examples shown below) and [oa] to [of] for the six *o*-quinone variants.

Instead of letting the command decide on the handedness of a substituent, you can specify it explicitly using `\rmoiety` and `\lmoiety` commands, as shown in the following example with left-handed and right-handed methanesulfonimido groups:

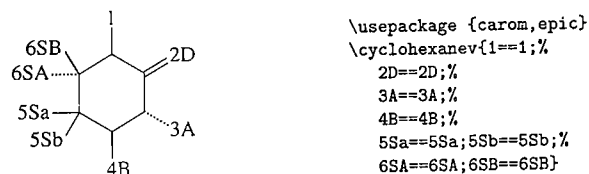


6-2-2

Depending on the base structure, two bonds might also be possible at a locant; in that case the modifiers for a bond in position *n* are:

- n*SA alpha single bond represented as a dotted line;
- n*SB beta single bond represented as a bold line;
- n*Sa unspecified alpha single bond;
- n*Sb unspecified beta single bond.

Again, the dotted line for the alpha bond requires the use of the epic or eepic package. As an abstract example we show below all bond modifiers in a single structure; real-life examples follow later on.



6-2-3

Syntax for general commands

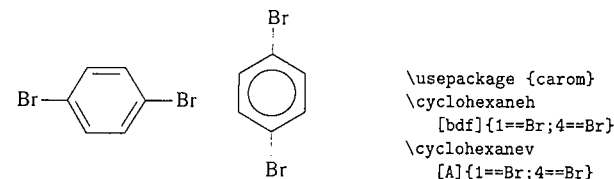
The general commands, such as `\cyclohexanev` in the previous example, are similar to the specific commands, but can have a variable set of skeletal heteroatoms in order to cover a wider range of compound structures. For carbocycles the general syntax is

```
\Com[bondlist]{sublist}
```

where, instead of an optional argument specifying a predefined bond pattern, we now have an optional *bondlist* argument that lets us specify double bonds individually. The possible

double bonds of the structure are internally labeled a, b, c, etc., and to make the ones desired appear, you specify them in the *bondlist* argument. Aromatic cycles are usually labeled with uppercase letters A, B, etc., if the structure is large enough to permit more than one cycle, e.g., in the case of perhydro anthracene derivatives.

All the specific commands are internally implemented as calls to the general forms with specific argument settings. For instance, one of our previous examples involving `\bzdrv` can also be typeset like this with the general command `\cyclohexaneh`:

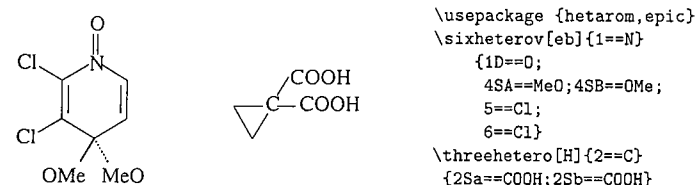


6-2-4

The syntax is a little more complex for heterocycles as it must allow a way to specify individual atoms on the base skeleton:

```
\Com[bondlist]{atomlist}{sublist}
```

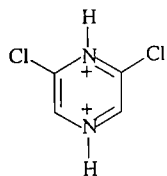
Endocyclic bonds are specified as before with the *bondlist* argument, as are the substituents, including their bonds, with the *sublist* argument. What is different for this type of commands is the additional mandatory *atomlist* argument, in which the individual atoms on the base skeleton are specified if necessary. The argument syntax is comparable to that of the *sublist* argument: atom specifications are separated by semicolons and atom positions are denoted by a number, followed by a double equals sign, followed by the string denoting the atom.



6-2-5

Charges on atoms

The charge on individual atoms of the base skeleton can sometimes be specified in the *opt* or *bondlist* argument, using the atom number followed by a plus sign and surrounded by a set of braces.



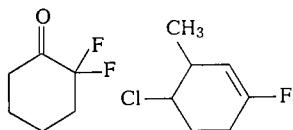
```
\usepackage {hetarom, epic}
\pyrazinev[1{1+}{4+}]
{1==H;4==H;2==Cl;6==Cl}
```

6-2-6

Having reviewed the design principles of the $\text{\LaTeX}$ system, we are now ready to look in more detail at some of its packages for drawing chemical diagrams.

6.2.2 The carom package

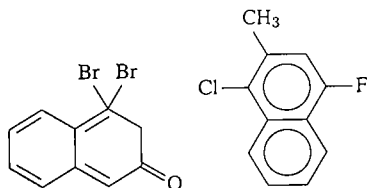
$\text{\LaTeX}$'s carom package allows you to draw a large variety of carbocycles. We have already seen some examples of the `\bzdrv` and `\bzdrh` commands to draw benzene derivatives. General cyclohexane derivatives are available with the commands `\cyclohexanev` and `\cyclohexaneh`. Using the *bondlist* parameter described earlier, you can introduce endocyclic double bonds at appropriate places, e.g.,



```
\usepackage {epic, carom}
\cyclohexanev
{1D==O;2Sa==F;2Sb==F}
\cyclohexaneh[c]
{1==Cl;4==F;2==CH$_{3}$}
```

6-2-7

Fused rings, such as naphthalenes and naphthoquinones, are drawn with the `\naphdrv` set of commands. The *opt* argument for bond patterns supports all type of o-quinone variants as well as the aromatic circle.



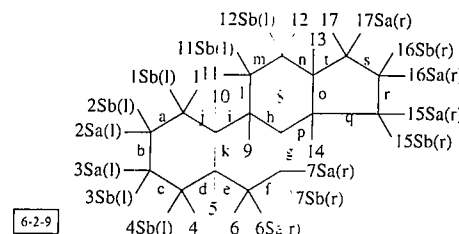
```
\usepackage {epic, carom}
\naphdrv[ob]
{1Sb==Br;1Sa==Br;3D==O}
\naphdrh[A]
{1==Cl;4==F;2==CH$_{3}$}
```

6-2-8

Commands for other complex structures are available, such as tetraline derivatives (the `\tetralinev` and `\tetralineh` commands), decaline derivatives (`\decalinev` and `\decalineh`), tricyclic carbocycles like anthracene derivatives (`\anthracenev` and `\anthraceneh`) and phenanthrene derivatives (`\phenanthrenev` and `\phenanthreneh`) as well as steroid derivatives with and without sidechains (`\steroid` and `\steroidchain`). These commands all employ the same general syntax, as outlined in the previous section, and are thus easy to use, although memorizing position

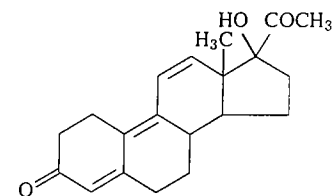
6.2.3 The lowcycle package

numbers and bond letters might take some time. These are well documented both in the package files and in the extensive online manual (more than 120 pages of diagrams and examples), where they are shown in diagrams like the following:



6-2-9

With diagrams like this and tables listing the allowed values for the *opt* or *bondlist* argument it is not too hard to produce formulae like:



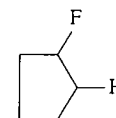
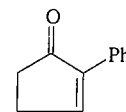
```
\usepackage {carom}
\steroid[dim]
{3D==O;%
{{13}B}==\lmoiety{H$_{3}$C};%
{{17}SB}==HO;%
{{17}SA}==COCH$_{3}$}
```

6-2-10

The optional argument [dim] indicates the positions of the three endocyclic double bonds and the command `\lmoiety` connects the right terminal carbon of the methyl to the corresponding bond at the fused 13-position.

6.2.3 The lowcycle package

Lower-order cycles (less than six carbon atoms in one of the cycles) are available by loading the lowcycle package. Five-member cycles can be drawn with the `\cyclopentanev` set of commands (all four suffix types). Below are a few examples.



```
\usepackage {lowcycle}
\cyclopentanev[b]{1D==O;%
2==Ph}
\cyclopentaneh[H]{1==H;2==F}
```

6-2-11

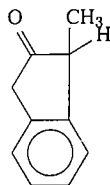
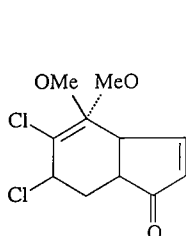
Beside putting charges on each atom, these structures also support a charge in the center using {O+} or {O\$-\$}.



```
\usepackage {epic,lowcycle}
\cyclopentanev[A{O{$-$}}]{}
```

6-2-12

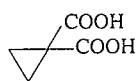
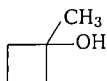
Also provided by this package are indane derivatives (fused six- and five-member rings) with the \indanev set of commands; again horizontal, vertical, and inverse forms exist.



```
\usepackage {epic,lowcycle}
\indanev[eb]
{1D==0;4SA==MeO;%
4SB==OMe;5==Cl;6==Cl}
\indanehi[A]{2D==0;%
3Sa==CH$_3$;3Sb==H}
```

6-2-13

Finally, four and three-member carbon cycles are supported through \cyclobutane and \cyclopropane. Not surprisingly, due to their geometric form they do not have different suffix variants.



```
\usepackage {lowcycle}
\cyclobutane
{3Sa==OH;3Sb==CH$_3$}
\cyclopropane{2Sa==COOH;%
2Sb==COOH}
```

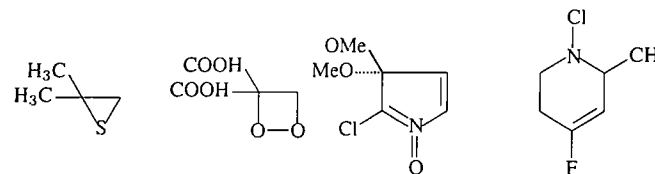
6-2-14

6.2.4 The hetarom and hetaromh packages

Heterocyclic compounds, i.e., those in which one or more of the carbon atoms in the cyclic chain have been replaced with other atoms, are available by loading the hetarom package (vertical-form commands) or the hetaromh package (the corresponding horizontal forms).

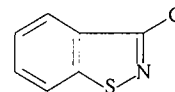
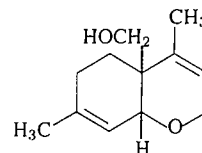
General commands for three- to six-member heterocycles are available with predictable names like \sixheterov; here are some representative examples:

```
\usepackage {epic,hetarom}
\threehetero {1==S} {3Sa==H$_3$C;3Sb==H$_3$C}
\fourhetero {1==O;2==O}{4Sa==COOH;4Sb==COOH}
\fiveheterov[eb] {1==N} {1D==0;4SA==MeO;4SB==OMe;5==Cl;6==Cl}
\sixheterov [c] {1==N} {1==Cl;4==F;2==CH$_3$}
```



6-2-15

In addition, fused six-member rings and fused five- and six-member rings are supported by general commands. They, and the previous two as well, have horizontal and (where applicable) invariant forms.



```
\usepackage {epic,hetarom}
\decaheterov[af]{4==0}
{1==CH$_3$;6==H$_3$C;9A==H;%
{{10}A}==\lmoiety{HOCH$_2$}}
\nonaheterov[bjge]{1==S;2==N}{3==Cl}
```

6-2-16

Clearly, with this many general commands it is easy to provide an even richer set of application commands, and indeed the two packages implement more than 80 of them (if we include in all the suffix variations in drawing the structures in different ways). This includes pyridine derivatives and other six-member ring structures; for five-member rings we have pyrrole derivatives and others. As specializations of the general fused-ring commands \nonaheterov and \decaheterov, there are commands for drawing N-heterocycles and N,O-heterocycles such as indoles and isobenzofuranes. Table 6.2 provides a reference list of these commands.

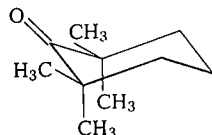
Table 6.2: Application commands of the hetarom package; corresponding horizontal forms are defined by the hetaromh package.

Six-Member Heterocycles	
\pyrazinev	\pyridazinevi
\pyridazinev	\pyridinevi
\pyridinev	\pyrimidinevi
\pyrimidinev	\triazinevi
\triazinev	
Five-Member Heterocycles	
\imidazolevi	\imidazolevi
\isoxazolevi	\isoxazolevi
\oxazolevi	\oxazolevi
\pyrazolevi	\pyrazolevi
\pyrrolevi	\pyrrolevi
Six-Six-Fused Heterocycles	
\cinnolinevi	\cinnolinevi
\isoquinolinevi	\isoquinolinevi
\pteridinevi	\pteridinevi
\quinazolinevi	\quinazolinevi
\quinolinevi	\quinolinevi
\quinoxalinevi	
Six-Five-Fused Heterocycles	
\benzofuranevi	\benzofuranevi
\benzoxazolevi	\benzoxazolevi
\indolevi	\indolevi
\indolizinevi	\indolizinevi
\isobenzofuranevi	
\isobenzofuranevi	\isobenzofuranevi
\isoindolevi	\isoindolevi
\purinevi	\purinevi

6.2.5 The ccycle and hcycle packages

Use of stereochemical compounds is facilitated with commands for drawing cyclohexane chair forms, bicyclo[2.2.1]heptane, and adamantane derivatives, which are loaded with the ccycle package.

The \chair command uses *n*Se as a bond modifier to denote an equatorial single bond at the *n*th atom, for example

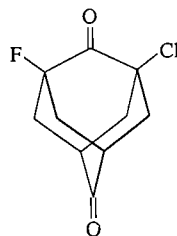
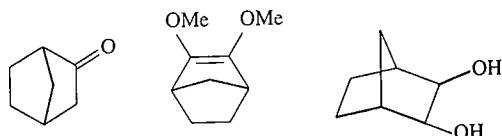


```
\usepackage {ccycle}
\chair{1D==0;%
2Se==H$_3$C;2Sa==CH$_3$;%
6Se==CH$_3$;6Sa==CH$_3$}
```

6-2-17

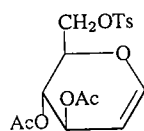
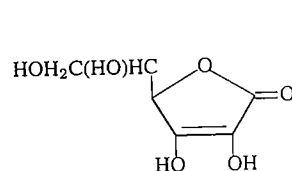
The other commands follow the standard syntax, except that the handedness of substituents cannot be specified explicitly:

```
\usepackage {ccycle}
\bicycpepv{2D==0}
\bornane{3B==OH;2A==OH}
\bicycpeph[b]{2==OMe;3==OMe}
\adamantane{2D==0;6D==0;1==F;3==Cl}
```



6-2-18

The commands for drawing furanoses and pyranoses derivatives are defined in the hcycle package. Below is an example of both.



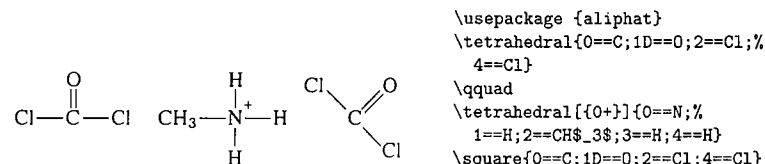
```
\usepackage {bcycle}
\furanose[b]{1D==0;2Sa==OH;%
3Sa==\lmoiety{HO};%
4Sb==HOH$_2$C(HO)HC}
\pyranose[a]{3Sb==OAc;%
4Sa==AcO;5Sb==CH$_2$OTs}
```

6-2-19

6.2.6 The aliphatic package

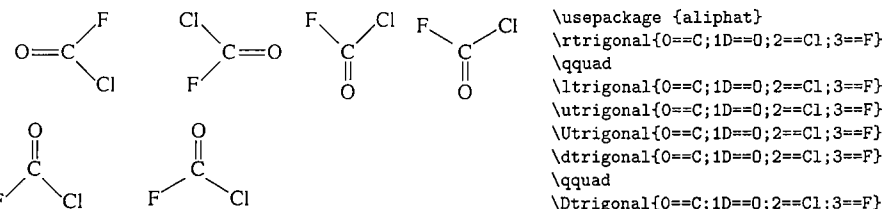
6.2.6 The aliphatic package

To draw tetrahedral and other aliphatic compounds you need the aliphatic package. As the skeleton of tetrahedral or trigonal units consists only of a single atom, the commands for drawing them have no *bondlist* argument. They do, however, take an optional argument in which a charge for this atom can be specified, as seen below. Another specialty of this type of command is the possibility of assigning an atom symbol to the central position by specifying, e.g., 0==C.



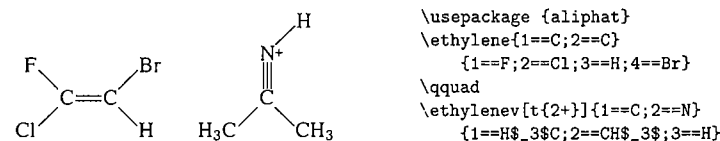
6-2-20

There are six commands for drawing trigonal units; those starting with an uppercase letter have an angle of 120° between the diagonal bonds, while the others are 90° apart.



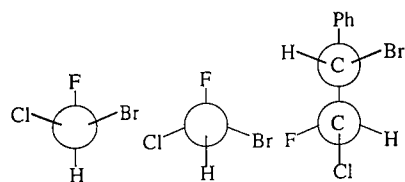
6-2-21

Drawing ethylene derivatives is supported with the \ethylene command (notice the missing suffix) and its vertical forms \ethylenev and \Ethylenev, the latter having the diagonal bonds 120° apart. Inner double and triple bonds are specified with d (default) and t in the optional argument.



6-2-22

Finally, the package supports the drawing of configurations of the tetrahedral carbon in different projections through the commands \tetrahedral and \dtetrahedral, and the configuration of ethane with the command \ethanestereo; here the central atoms can be specified explicitly.



```
\usepackage {aliphat}
\setlength\unitlength{0.1pt}
\tetrastereo{1==F;%
2==Cl;3==H;4==Br}
\dtetrastereo{1==F;%
2==Cl;3==H;4==Br}
\ethanestereo{1==C;2==C}
{1==F;2==Cl;3==H;4==Br;%
5==Ph;6==H}
```

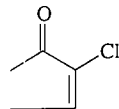
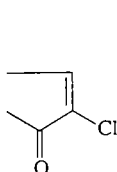
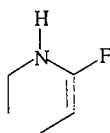
6-2-23

6.2.7 Combining structures

The `hetarom` and `hetaromh` packages also support six- and five-member building blocks, which can be fused with other ring structures to produce new ring systems. Their syntax is comparable to that for general heterocycles but with one additional mandatory argument:

```
\Com[bondlist]{atomlist}{sublist}{omit}
```

The *omit* argument specifies which bonds are to be deleted from the structure, e.g.,



```
\usepackage {hetarom}
\sixunitv[b]{1==N}{1==H;2==F}{d}
\fiveunitv[b]{1D==O;2==Cl}{d}
\fiveunitvi[b]{1D==O;2==Cl}{d}
```

6-2-24

The corresponding horizontal commands are defined in the `hetaromh` package.

More complicated structural formulae can be constructed from such fragments or from the commands in previous sections by combining two or more structures inside a `picture` environment (this technique is explained in detail in Chapter 14 of Fujita (1993)). Another way to construct complex structures is to use one $\text{\LaTeX}$ command inside the argument of another (*ibid.*, Chapter 15).

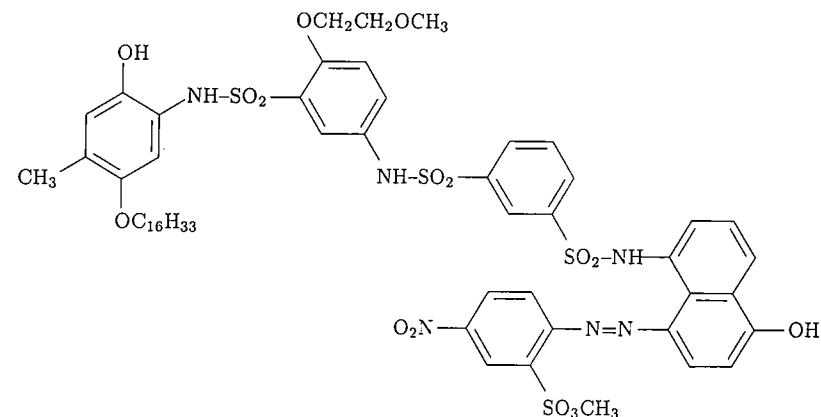
6.2.8 Defining your own structures

It is easy to implement new application macros with the general commands. For example, had the `\pyridinev` command been unavailable, you could add it easily with the following definition in the document preamble.

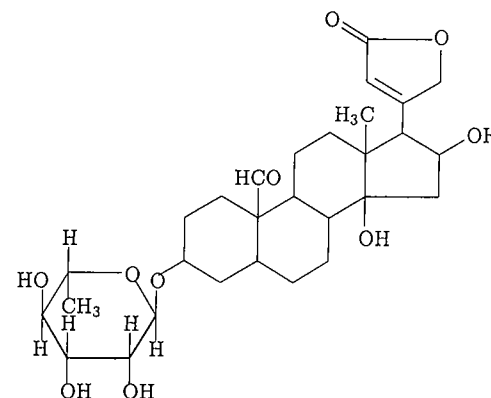
```
\newcommand{\pyridinev}[2][r]{\sixheterov[#1]{1==N}{#2}}
```

Notice the second optional argument to `\newcommand` specifying that `\pyridinev` should have an optional first argument with the default value `r`.

We conclude the overview of the $\text{\LaTeX}$ package by giving two complex examples without code. Both are discussed in detail in the manual (Fujita, 1993, 115–117). The first is the formula of a cyan dye releaser for instant color photography:



The second is the formula of adonitoxin, in which two complex substituents are placed on a steroid skeleton:



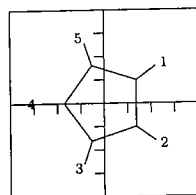
6.3 The ppchtex package

A completely different approach to drawing chemical formulae has recently been taken by Hans Hagen and A.F. Otten (1996), who wrote PPCHTEX, a package for typesetting chemical formulae based on PCTEX (Wichura, 1987); an interface to PSTricks is under development. This package describes the chemical structures more by their graphical representation than their chemical significance. Unlike X_YTEX, which relies on the L^AT_EX picture environment, this package can be used with both plain T_EX and L^AT_EX.

According to the authors, the package is still evolving. Rather than trying to explain its current syntax in detail we merely present some examples, with the hope of giving you a feeling for the approach taken; for details, see the documentation accompanying the system.

6.3.1 Structures

Chemical structure formulae are typeset with the help of four basic commands, all of which are used in the following example:

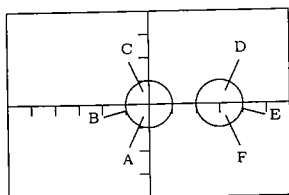


```
\usepackage {m-chemic}
\setupchemical
  [axis=on, border=on, scale=small,
  size=medium]
\startchemical
  \chemical [FIVE, B, RZ] [1, 2, 3, 4, 5]
\stopchemical
```

6-3-1

```
\setupchemical {typesetting-data}
\startchemical {typesetting-data}
```

The `\setupchemical` command establishes various characteristics for setting the following chemical formulae. Its scope is limited to the current group. Typesetting instructions can also be specified on the `\startchemical` command itself, in which case they are valid only up to the corresponding `\stopchemical` command, i.e., for the current formula.



```
\usepackage {m-chemic}
\setupchemical [axis=on]
\startchemical
  [width=6000, scale=small, size=medium]
  \chemical [CARBON, CB1] [A, B, C, D, E, F]
\stopchemical
```

6-3-2

6.3.1 Structures

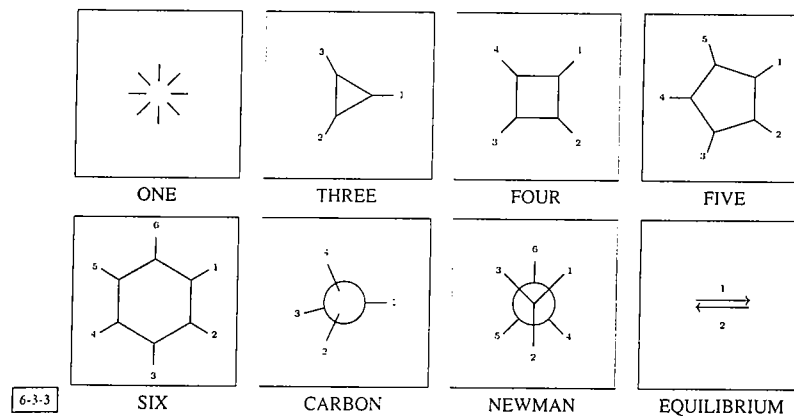


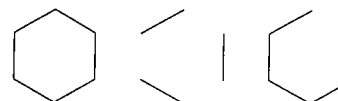
Figure 6.2: Some predefined structures in PPCHTEX.

```
\chemical [structure] [atoms/molecule-list]
```

`\chemical`, the main command, can occur several times between the `\startchemical` and `\stopchemical` commands that delimit a chemical formula. Its first argument defines the *structure* that is filled with the data presented in the second (optional) argument. Any text in the *atoms/molecule-list* is typeset in math mode (but with letters set properly in roman). If the argument is missing, the command produces a skeleton with no atoms.

Fig. 6.2 shows the list of base structures that can be used in the first argument *structure*. Chemical bonds between C-atoms inside these structures are indicated, for example, by the letter B and are referenced by numbers. Various representations are possible, as shown below for a six-ring structure:

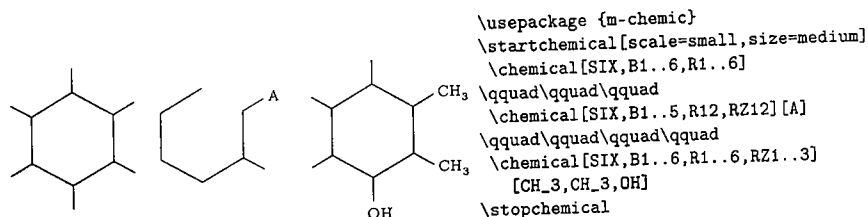
6-3-4



```
\usepackage {m-chemic}
\startchemical [scale=small, size=medium]
\chemical [SIX, B1, B2, B3, B4, B5, B6]
\quad\quad\quad
\chemical [SIX, B135]
\quad\quad\quad
\chemical [SIX, B1..5]
\stopchemical
```

Substructures can be added by the R tag, where the position on the ring again is indicated by a number. This tag merely draws bonds for subcomponents; the actual subcomponent must be declared explicitly with a RZ tag and its "value", e.g., an atom, is then specified

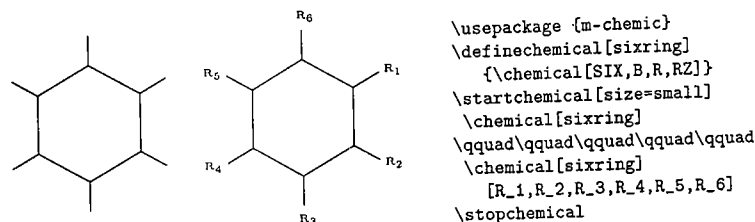
inside the second optional argument. When the second argument is omitted or doesn't contain enough material, no text is placed and the corresponding RZ tags have no effect.



6-3-5

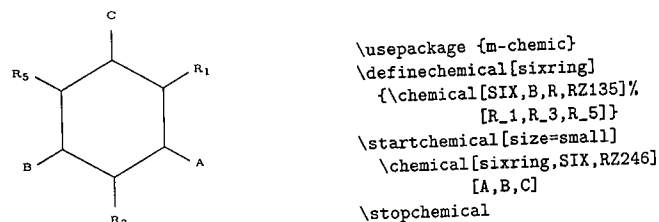
6.3.2 Definitions

It is straightforward to build a library of structures that can be used as building blocks of more complex components. For instance, you can define the string "sixring" as an unadorned six-ring by using the `\definechemical` command. With a definition like the one below, a simple six-ring without substituents can be typeset and the six substituents can be added by specifying them in the second argument:



6-3-6

More complex structures are possible; remember that the substituents in the second argument are placed according to the sequence specified in the first argument. An example with a slightly modified definition of `sixring` is:



6-3-7

Table 6.3: Bond identifiers for ppchtext.

<i>Saturated bonds</i>	B	bond	SB	single bond
			-SB	left single bond
			+SB	right single bond
<i>Unsaturated bonds</i>	EB	extra bond	DB	double bond
			TB	triple bond
<i>Special bonds</i>	S	shortcut	C	circle
<i>Bonds to substituents</i>	R	radical	SR	single radical
	-R	left radical	-SR	single left radical
	+R	right radical	+SR	single right radical
<i>Double bonds to substituents</i>	ER	extra radical	DR	double radical
<i>Atoms and molecules (radicals)</i>	Z	atom	RZ	radical atom
			-RZ	left radical atom
			+RZ	right radical atom

6.3.3 Bonds

Different kinds of bonds can be used in structures in the PPCHTEX package, as shown in Table 6.3. The left-hand pair of columns corresponds to complete bonds and the right-hand columns are for shortened variants. Bonds can be shortened on both sides, on the left (-) or at the right (+). Shortened variants make it possible to attach atoms and molecules to a bond.

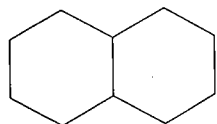
Bond specifiers can be followed by one or more numbers or a range (as in B1, B135, or B1..5). A specifier without numbers (e.g., B), applies to all bonds. Text to be linked to bonds is collected from the second optional argument of `\chemical` in the order given. Items are numbered clockwise. A text can be placed in the center of a structure with the Z0 ("Z-zero") specifier.

6.3.4 Combinations

Structures can be positioned or rotated to form complex compounds with the help of the following specifiers:

- MOV (Move) move an identical structure in the direction of a bond;
- ADJ (Adjace) move two structures relative to each other in the *x*- or *y*-direction to make them share a common side;
- SUB (Substitute) move two structures relative to each other in the *x*- or *y*-direction and connect them via a bond;
- ROT (Rotate) rotate a structure.

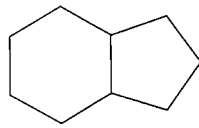
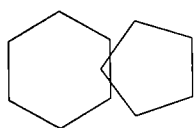
Note that these commands can have different effects for different structures; for instance, the rotation angle differs in the commands `\chemical [FIVE,ROT1,B]` and `\chemical [SIX,ROT1,B]`. A few examples are given below.



```
\usepackage {m-chemic}
\startchemical[size=small]
\chemical[SIX,B,MOV1,B]
\stopchemical
```

6-3-8

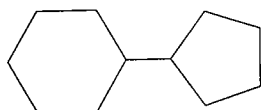
Here we can study the effects of the ROT tag:



```
\usepackage {m-chemic}
\mbox{\startchemical[size=small]
\chemical[SIX,B,ADJ1,FIVE,B]
\stopchemical\startchemical
\chemical[SIX,B,ADJ1,FIVE,ROT3,B]
\stopchemical}
```

6-3-9

As you see, connected structures are built with the ADJ specifier. Often one of the two structures must be rotated with ROT to get a good connection. If there is a bond between the two structures, then use the SUB specifier. Rotations are in steps of 90 degrees clockwise and displacements with ADJ and SUB are in the direction of one of the four axes.



```
\usepackage {m-chemic}
\startchemical
\chemical[SIX,ROT2,B,R6,%
SUB1,FIVE,B,R4]
\stopchemical
```

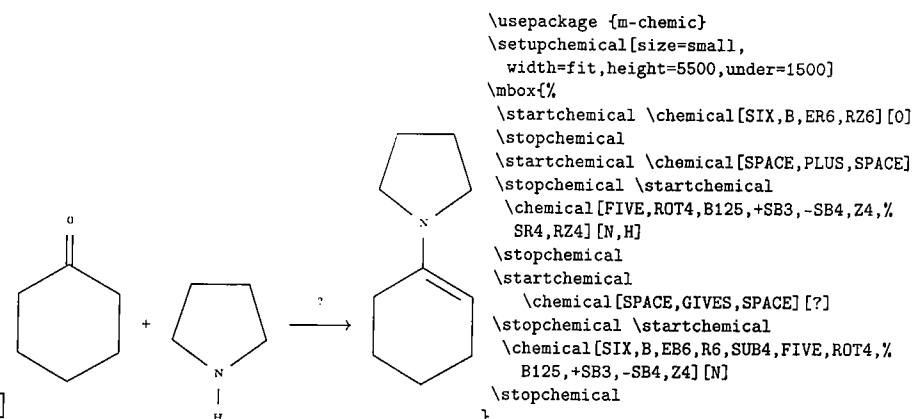
6-3-10

The above examples indicate that the sequence in which the tags are specified must follow a given order:

```
\chemical
[structure & rotation,      % SIX, FIVE, ...
bonds inside the structure, % B, C, EB, ...
bonds outside of the structure, % R, DR, ...
locations of the atoms,      % Z
locations of the subcomponents, % RZ, -RZ, ...
connection,                  % MOV, ADJ, ...
structure & rotation,        % SIX, FIVE, ...
bonds inside the structure,   % B, C, EB, ...
...
]
[atoms, subcomponents]
```

6.3.5 Chemical equations

Reaction equations can be typeset using symbols specially provided for the purpose, as the following example shows:

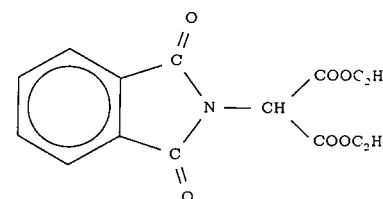


6-3-11

The \mbox command ensures that the structures are typeset on a line; the symbols GIVES and PLUS are self-explanatory, and SPACE inserts a little extra space. An equilibrium can be shown with EQUILIBRIUM and a text can be typeset above GIVES and EQUILIBRIUM (here a question mark).

6.3.6 Special features

With the ONE tag, ZO can consist of more than one atom. If the reserved space is insufficient, bonds 1, 2, and 8 can be moved with the command OFF (for "offset"), as shown in the following example, where the offset is one extra character ("1"). Note that in complex constructs like this, rotations are best made last. The CRZ (for "centered radical atom") command is used to align an atom or a molecule with a bond.



6-3-12

```
\usepackage {m-chemic}
\startchemical[width=fit]
\chemical
[SIX,B,C,ADJ1,FIVE,ROT3,SB34,
+SB2,-SB5,Z345,DR35,SR4,CRZ35,
SUB1,ONE,OFF1,SB258,ZO,Z28]
[C,N,C,O,O,CH,COOC_2H_5,COOC_2H_5]
\stopchemical
```

There are two other forms of the `\chemical` command:

```
\chemical{formula}
\chemical{formula}{text}
```

These forms typeset a formula inside a paragraph of text, for instance

```
\usepackage {m-chemic}
\[ \chemical{2H_2}{+}
  \chemical{O_2}{+}
  \chemical{O_2}{+}
  \chemical{GIVES}{+}
  \chemical{2H_2O}{+} \]
```

$$2\text{H}_2 + \text{O}_2 \longrightarrow 2\text{H}_2\text{O}$$

6-3-13

Such a formula can also be typeset inside a paragraph in the running text where a smaller fontsize is chosen. Note that the same formula could have been obtained with either of the two (shorter) `\chemical` commands in which everything is specified in the first argument:

```
\[\chemical{2H_2,PLUS,O_2,GIVES,2H_2O}{+}\]
\[\chemical{2H_2,+,O_2,->,2H_2O}{+}\]
```

The second argument *text* of the `\chemical` commands can be used to include explanatory text in the formula, as in:

```
\usepackage {m-chemic}
\[ \chemical{2H_2}{hydrogen}
  \chemical{PLUS}{+}
  \chemical{O_2}{oxygen}
  \chemical{GIVES}{violent}
  \chemical{2H_2O}{+} \]
```

$$\underset{\text{hydrogen}}{2\text{H}_2} + \underset{\text{oxygen}}{\text{O}_2} \xrightarrow{\text{violent}} 2\text{H}_2\text{O}$$

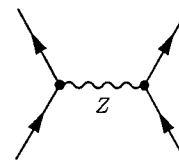
6-3-14

The package also supports other features such as coloring parts of a formula.

6.4 Drawing Feynman diagrams

Feynman diagrams are used in many branches of physics to show how to organize the various calculational contributions for basic elementary processes. Several packages for drawing these diagrams have been developed over the years. Michael Levine's `feynman` package (1990) is an implementation on top of standard $\text{\LaTeX}$'s `picture` environment. This makes it completely portable, but the graphics output is necessarily less than perfect and complex graphs are often impossible to draw. Jos Vermaseren's `axodraw` package (1994) uses `\special` commands to directly access PostScript primitives for drawing the diagrams, a very flexible approach that produces visually more appealing results. `Feynarts` is a Mathematica package with procedures to calculate and draw Feynman diagrams (Eck and

6-4-1



```
\usepackage {feynmp}
\begin{fmfgraph*}(100,70)
  \fmfleft{em,ep}
  \fmf{fermion}{em,Zee,ep}
  \fmf{photon,label=$Z$}{Zee,Zff}
  \fmf{fermion}{fb,Zff,f}
  \fmfright{fb,f}
  \fmfdot{Zee,Zff}
\end{fmfgraph*}
```

Figure 6.3: Simple example of the `feynmp` package.

Kühlbeck, 1991). In this section, however, we have a closer look at Thorsten Ohl's sophisticated `FeynMF` system (1995, 1996), which fully exploits the formal structure of Feynman graphs, freeing the user from specifying the layout manually with low-level graphic primitives.

6.4.1 Using FeynMF

The aim of the `FeynMF` system is to provide a user interface so that the user need not specify graph layouts at the level of points and curves. The package was designed with the following goals:

- *Simplicity and conciseness* for common diagrams. For example, the Z-particle production diagram (very common in high-energy physics) in Fig. 6.3 can be specified with eight $\text{\LaTeX}$ commands. The position of vertices is calculated automatically.
- *Expressiveness and extensibility* for complex diagrams.
- *Portability*: only $\text{\LaTeX}$ and METAFONT (or METAPOST) are needed.
- *Integration* with $\text{\LaTeX}$: all labels, including complex math expressions, are done in $\text{\LaTeX}$.

There are two versions of the `FeynMF` package: `feynmp` uses METAFONT for drawing, while `feynmp` uses METAPOST and can also add color to diagrams. Apart from this, the programs differ little at the $\text{\LaTeX}$ level. Just as in Chapter 3, we write `META` if the explanation is valid for both programs.

The system works by writing METAFONT or METAPOST code into an external file; the sequence of use is as follows:

1. run $\text{\LaTeX}$; this writes METAFONT or METAPOST code for all the pictures into a file;
2. run METAFONT or METAPOST; this generates either a set of PostScript files (METAPOST) or a font with a set of characters, one for each picture; in addition, an auxiliary file with labeling information is created;
3. run $\text{\LaTeX}$ again; this time the pictures or the font and the labeling information are used.

If the FeynMF code in the $\text{\LaTeX}$ file does not change, there is no need to run METAFONT or METAPOST again every time you process the $\text{\LaTeX}$ file.

The choice to use METAFONT or METAPOST is taken in the preamble of a document by loading either the `feynmf` or the `feynmp` package. The commands defining your diagrams are then placed within the scope of an `fmffile` environment, which has an argument specifying the name of the file for intermediate METAFONT or METAPOST code. Thus the skeleton of a complete $\text{\LaTeX}$ file might look like this:

```
\documentclass{article}
\usepackage{feynmf}
\begin{document}
\begin{fmffile}{fmpict}
.... diagram commands ....
\end{fmffile}
\end{document}
```

Here we chose `feynmf`, so a METAFONT file called `fmpict.mf` is created in which each character represents one diagram.

To avoid a naming clash between the transcript files of $\text{\LaTeX}$ and METAFONT or METAPOST, which usually all have the extension `log`, the $\text{\LaTeX}$ file *cannot* have the same name as the FeynMF file specified as the argument on the `fmffile` environment, i.e., `fmpict` in the above example.

When a file like the above is run through $\text{\LaTeX}$ for the first time, you see messages like the following:

```
This is TeX, Version 3.14159 (C version 6.1)
(feynmfex.tex
LaTeX2e <1995/06/01> patch level 3
(/usr/local/lib/texmf/tex/latex/base/article.cls
Document Class: article 1995/06/26 v1.3g Standard LaTeX document class
(/usr/local/lib/texmf/tex/latex/base/size10.clo)
(/usr/local/lib/texmf/tex/latex/feynmf/feynmf.sty
Package: 'feynmf' v1.0 (rev. 1.12) <1995/05/06> (oh1))
No file feynmfex.aux.
feynmf: File fmpict.tfm not found:
feynmf: Process fmpict.mf with METAFONT and then reprocess this file.
feynmf: Label file fmpict.tl not found:
feynmf: Process fmpict.mf with METAFONT and then reprocess this file.
[1] (feynmfex.aux)
Output written on feynmfex.dvi (1 page, 388 bytes).
Transcript written on feynmfex.log.
```

The resulting file `fmpict.mf` must be processed by the METAFONT program to generate the $\text{\TeX}$ font metric file `fmpict.tfm`, as well as a *generic font* `gf` file. This file is then transformed with `gftopk` into a packed bitmap image of type `pk` so that it can be used with a `dvi` driver (see Chapter 11):

```
> virmf "\mode=localfont;\input fmpict"
This is METAFONT, Version
2.71 (C version 6.1) (fmpict.mf
(/usr/local/lib/texmf/fonts/feynmf/src/feynmf.mf
:1:\fmfL(19.98138,14.59717,b){\ZS}% [1] )
```

```
Font metrics written on fmpict.tfm.
Output written on fmpict.300gf (1 character, 2184 bytes).
Transcript written on fmpict.log.
> gftopk fmpict.300gf
```

When METAPOST rather than METAFONT is used to generate the images, the FeynMF system writes a METAPOST file called `fmpict.mp`. When the METAPOST program runs on this, PostScript drawings in files named `fmpict.n` are created for each diagram in the input file (where n is the sequence number of the graph). These are automatically included in following $\text{\LaTeX}$ runs using the standard graphics package.

6.4.2 Writing FeynMF diagrams

You can run FeynMF in two different modes, depending on how you want to tackle a particular problem:

- *vertex mode*, in which the layout is determined automatically from the mathematical description of the graph (including its vertices and arcs); physical coordinates are not normally specified;
- *immediate mode*, in which you can completely control all physical coordinates by coding in META.

Vertex mode and algorithmic layout

In this mode everything can be specified at the $\text{\LaTeX}$ level and no knowledge of META is needed. For maximal flexibility, FeynMF accepts a *mathematical* description of a graph and creates the layout of the corresponding Feynman diagram automatically from that specification.

FeynMF has commands to place external vertices along the sides of a diagram. To calculate optimal positions for the vertices, FeynMF minimizes a weighted sum of squared lengths for the internal vertices with the help of META:

$$L(v_1, \dots, v_n) = \frac{1}{2} \sum_{i,j} t_{ij} (v_i - v_j)^2,$$

where i, j run over all combinations of vertices. The elements of the tension matrix t_{ij} are taken as 1 by default, but the user can specify other values to tune the layout. The tension parameters can be viewed as rubber bands, that let you pull together or push apart adjacent vertices (see Fig. 6.4).

Practice has shown that the most effective way to draw Feynman diagrams is a combination of step-by-step construction of subgraphs and, if necessary, adjustment of tensions. Often the default settings for the tensions give a quite satisfactory result straightaway, and only fine-tuning the tension of a single arc or loop is necessary.

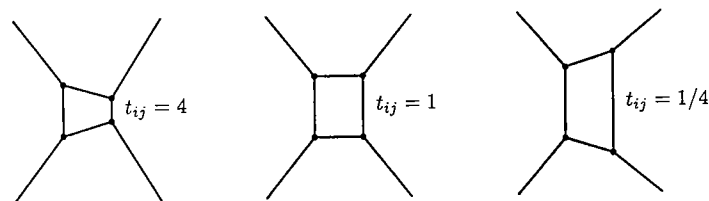


Figure 6.4: Varying the tension parameter of a single line.

A large choice of line, vertex, and fill styles are used by physicists, and FeynMF provides the most common (see Tables 6.4 and 6.5).

```
\fmfleft{v_1,...}      \fmfleftn{v}{n}
\fmfright{v_1,...}     \fmfrightn{v}{n}
\fmftop{v_1,...}       \fmftopn{v}{n}
\fmfbottom{v_1,...}    \fmfbottomn{v}{n}
\fmfsurround{v_1,...}  \fmfsurround{v}{n}
```

These are FeynMF's basic commands in vertex mode; they place the set of external vertices $v_1, \dots$ at the left, right, top, bottom, or surrounding the diagram. The right-hand form of the commands (with suffix n) places all vertices v from 1 to n , without no need to list them explicitly.

```
\fmfcurved   \fmfstraight
```

By default the external vertices are put on a smooth curved path. When the `\fmfstraight` command is specified, they are put on a straight path from then on (i.e., `\fmfcurved` and `\fmfstraight` switch between both alternatives).

```
\fmf{lsty}{v_1,...}      \fmfn{lsty}{v}{n}
\fmfcyclen{lsty}{v}{n}  \fmfrcyclen{lsty}{v}{n}
```

The command `\fmf` connects a set of vertices $v_1, \dots$ with linestyle *lsty* (see Table 6.5 on the facing page). This linestyle can be customized further by specifying a number of options (see Table 6.6). For instance,

```
\fmf{fermion,tension=.5}{vw,vn,ve,vs,vw}
```

connects the specified internal vertices with a “fermion” line using a “tension” parameter value of one half. The other commands `\fmfn`, `\fmfcyclen`, and `\fmfrcyclen` connect vertices v_1 to v_n , normally, cyclically, or cyclically in reverse order, respectively.

Table 6.4: FeynMF vertex and fill styles.

	filled=-.5	filled=0	filled=.5	filled=1
circle				
square				
triangle				
diamond				
pentagon				
hexagon				
triagram				
tetragram				
pentagram				
hexagram				

Table 6.5: FeynMF linestyles.

	curly		phantom
	dbl_curly		phantom_arrow
	dashes		plain
	dashes_arrow		plain_arrow
	dbl_dashes		dbl_plain
	dbl_dashes_arrow		dbl_plain_arrow
	dots		wiggly
	dots_arrow		dbl_wiggly
	dbl_dots		zigzag
	dbl_dots_arrow		dbl_zigzag

`\fmfpen{wgt}`

This command sets the thickness (weight) of the lines to `wgt`. Predefined sizes are `thin` and `thick`. To change the width of individual arcs, use the `width` parameter in Table 6.6.

`\fmfv{vopt}{v1,...} \fmfvn{vopt}{v}{n}`

The command `\fmfv` declares the set of internal vertices $v_1, \dots$ with options `vopt` (`\fmfvn` does the same for vertices v_1 to v_n). Table 6.4 on the previous page shows some of the available vertex forms and fill styles, and Table 6.7 shows possible values for the options associated with vertices.

`\fmfdot{v1,...} \fmfdotn{v}{n}`

This is a special case of the `\fmfv` command, in which a set of vertices is drawn as dots. For instance, the two following commands are equivalent:

```
\fmfdotn{v}{4}
\fmfv{decor.shape=circle,decor.filled=full,
decor.size=2thick}{v1,v2,v3,v4}
```

`\fmfblob{dia}{v1,...} \fmfblobn{dia}{v}{n}`

Similarly, Thorsten Ohl has created a shorthand for drawing a “blob”; both commands below have the same result:

```
\fmfv{decor.shape=circle,decor.filled=shaded,
decor.size=5mm}{vblob}
\fmfblob{5mm}{vblob}
```

`\fmfpoly{vopt}{v1,...} \fmfpoly{vopt}{v}{n}`

Complex vertices are commonplace in solid-state physics. These can be constructed with polygons. The command `\fmfpoly` places the vertices $v_1, \dots$ on a polygon using options `vopt` (`\fmfpoly` is similar for vertices v_1 to v_n). Possible options are listed in Table 6.8.

`\fmffreeze`

A number of commands let one influence the automatic layout algorithms of FeynMF. Perhaps the most important one is `\fmffreeze`, which calculates the diagram up to the current point and “freezes” it, so that arcs added later do not affect its positioning. This important technique of using skeletons in the construction of diagrams is described in detail in the manual (Ohl, 1996); Fig. 6.6 shows an example.

Table 6.6: FeynMF line-drawing options.

Option name	Explanation
label	TeX text used for arc label
label.side	place label at “left” or “right”
label.dist	place label at given distance
left	draw half circle on left
right	draw half circle on right
straight	draw straight line (default)
tag	tag for disambiguating arc (if needed)
tension	draw tighter (> 1) or looser (< 1) arc
width	line width
foreground	foreground color (METAPOST only)
background	background color (METAPOST only)

Table 6.7: FeynMF vertex-drawing options.

Option name	Explanation
decoration.shape	shape of decoration
decoration.size	size of decoration
decoration.filled	fill, shade or hatch decoration
decoration.angle	rotate decoration
label	TeX text used for vertex label
label.angle	place label at angle with respect to vertex
label.dist	place label at given distance
foreground	foreground color (METAPOST only)
background	background color (METAPOST only)

METAPOST lets you use color in your diagrams (via the `foreground` and `background` specifiers in the line- and vertex-drawing options of Tables 6.6 and 6.7). The predefined colors are `white`, `black`, `red`, `green`, and `blue`, while other colors can be specified as RGB (red, green, blue) triplets. For instance, `foreground=(1,,1)`⁴ and `foreground=red+blue` are equivalent. For arcs the background color is used only for the interior between double lines. For example, the following command draws a red gluon line between the vertices `in` and `out`:

```
\mf{gluon,fore=red}{in,out}
```

⁴Note the double commas “,”, which are needed to disambiguate the comma as option separator in the commands and inside the option values.

Table 6.8: FeynMF polygon options.

Option name	Explanation
empty	only outline drawn
fill	filled interior
filled	interior filled, shaded or hatched
hatched	hatched interior
label	TeX text for labelling polygon
label.angle	place label at angle with respect to vertex
label.dist	place label at given distance
phantom	nothing is drawn
pull	edges pulled in (<0) or out (>0)
shade	shaded interior
smooth	corners are drawn smoothed
tension	tension used for edges
foreground	foreground color (METAPOST only)
background	background color (METAPOST only)

Note that option keywords can be abbreviated to their shortest non-ambiguous form, e.g., `fore` for `foreground` in the previous example. This works for each dot-separated component of an option name; thus `l.d` is interpreted as `label.dist`.

FeynMF can calculate optimal positions for labels with the help of METAFONT. Since METAFONT can write only to its log file, the positioning information needed to typeset the labels with L^AT_EX is written in that file, which is subsequently read and parsed by L^AT_EX. By default all labels are placed at the outside of the arc or vertex with which they are associated. Explicit user placement of labels is, of course, possible, as described in the manual.

To give a flavor of how to specify Feynman diagrams in vertex mode, look at the diagram in Fig. 6.5 on the facing page. The environment `fmfgraph` contains the description of a single Feynman diagram. In analogy with L^AT_EX's standard `picture` environment, the parameter specifies the width and height of the diagram in units of `\unitlength`. This environment does not allow labeling the diagram. To add labels, for instance to label the central arc and the external vertices in our figure, use the starred version `fmfgraph*`.

Line 3 in Fig. 6.5 declares two incoming particles (at the left of the diagram, `\fmfleftn` command) and four outgoing particles (at the right of the diagram, `\fmfrightn` command) and lines 4–8 assign them a label. The inner vertices are numbered `v1` to `v4` from left to right, so that Line 9 connects the incoming fermions `i1` and `i2` with the first inner vertex `v1`. Then in line 10 this vertex is connected with a boson linestyle to the left of the “blob” (inner vertex `v2`), and a label is added. Line 11 draws the boson line between the internal vertices `v2`, `v3`, and `v4`, while lines 12–13 connect these latter two inner vertices with outgoing fermion lines. Finally, on line 14, vertices `v1`, `v3`, and `v4` get a dot, while vertex `v2` puts a blob with a diameter equal to `.12w` (`w` being the total width of the diagram), yielding a blob of 0.12×50 mm or 6 mm.

```

1 \usepackage {feynmp}
2 \begin{fmfgraph*}(100,60)
3   \fmfleftn{i}{2} \fmfrightn{o}{4}
4   \fmflabel{$e_-}{i1}\fmflabel{$e_+}{i2}
5   \fmflabel{$\mu_-}{o1}
6   \fmflabel{$\nu_{\mu}$}{o2}
7   \fmflabel{$s$}{o3}
8   \fmflabel{$\bar{c}$}{o4}
9   \fmf{fermion}{i1,v1,i2}
10  \fmf{boson,label=$\gamma,Z$}{v1,v2}
11  \fmf{boson}{v3,v2,v4}
12  \fmf{fermion}{o1,v3,o2}
13  \fmf{fermion}{o4,v4,o3}
14  \fmfdot{v1,v3,v4}\fmfblob{.12w}{v2}
15 \end{fmfgraph*}

```

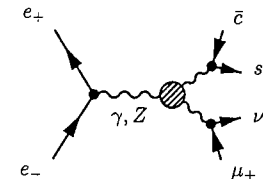


Figure 6.5: Drawing a Feynman diagram in vertex mode.

Immediate mode

FeynMF's vertex mode discussed in the previous section operates on abstract vertices, and the result depends on how these are connected. In most cases this “automatic” vertex mode suffices to obtain the desired layout. However, with minor exceptions, this mode can produce straight lines only. Therefore if you want curved arcs, you should use FeynMF's *immediate* mode. This mode also lets you control the positioning of the diagram elements more closely, since it acts on the vertex coordinates and the arcs (*paths* in Knuth (1986c)) connecting them.

Let us consider the loop diagrams in Fig. 6.6 on the next page. The left-hand graph is drawn in vertex mode, while the right-hand version, using immediate mode, has a more attractive appearance.

For clarity the vertices of both loop diagrams are named, going clockwise, `vw`, `vn`, `ve`, and `vs` (for west, north, east, and south). In the first diagram we observe on line 5 the use of the `tension` parameter to control the fermion loop. Line 7 has a `\fmffixed` command, which fixes the distance between subsequent vertices in the list. Here the distance between the top and bottom vertex is fixed to the height of the diagram `h`, i.e., 20 “units” (such a constraint is used in the METAFONT processing step for calculating the layout of the diagram). Without this command the loop would collapse.


```

1 \usepackage {feynmp}
2 \fbox{\begin{fmfgraph}(100,40)
3 \fmfleft{w}\fmfright{e}
4 \fmf{boson}{w,vw}\fmf{boson}{ve,e}
5 \fmf{fermion,tension=.5}{vw,vn,ve,vs,vw}
6 \fmf{gluon}{vn,vs}
7 \fmffixed{(0,h)}{vn,vs}
8 \fmfdot{vw,vn,ve,vs}
9 \end{fmfgraph}}
10 \fbox{\begin{fmfgraph}(100,40)
11 \fmfleft{w}\fmfright{e}
12 \fmf{boson}{w,vw}\fmf{boson}{ve,e}
13 \fmf{phantom,left,tension=.4}{vw,ve,vw}
14 \fmfdot{vw,ve}
15 \fmffreeze
16 \fmfipath{pn,ps}\fmfipair{vn,vs}
17 \fmfiequ{pn}{vpath (__vw,__ve)}
18 \fmfiequ{ps}{vpath (__ve,__vw)}
19 \fmfiequ{vn}{point .5length(pn) of pn}
20 \fmfiequ{vs}{point .5length(ps) of ps}
21 \fmfi{fermion}{subpath (0,.5)*length(pn) of pn}
22 \fmfi{fermion}{subpath (.5,1)*length(pn) of pn}
23 \fmfi{fermion}{subpath (0,.5)*length(ps) of ps}
24 \fmfi{fermion}{subpath (.5,1)*length(ps) of ps}
25 \fmfi{gluon}{vn--vs}
26 \fmfiv{dec.sh=circle,dec.siz=2thick}{vn}
27 \fmfiv{dec.sh=circle,dec.siz=2thick}{vs}
28 \end{fmfgraph}}

```

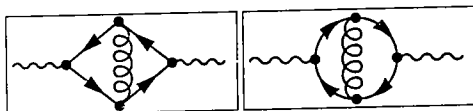


Figure 6.6: Vertex mode and improved version using immediate-mode commands.

Now look at the “improved” diagram. Lines 11–12, corresponding to the “external” lines, are similar to those in the previous diagram. The `\fmffreeze` command (line 15) ensures that this part of the diagram remains fixed (i.e., it cannot be influenced by subsequent FeynMF commands). From line 16 onwards we use FeynMF’s *immediate* commands, which all start with the four letters `fmfi`⁵. The `\fmfipath` and `\fmfipair` commands declare a META path and coordinate pair, respectively. Lines 17–20 are assignments (`arg1=arg2`). Note the `vpath` commands, which get the META path between two vertices

⁵For a detailed understanding of these commands you should have some idea about META’s constructs, e.g., how they connect vertices using Bézier curves (see Knuth, 1986c).

(after `\fmffreeze`), and note also that the vertices must be preceded by a double underscore (e.g., `ve` becomes `__ve`). The `\fmfi` commands on lines 21–25 draw a line in the given linestyle (first argument) along a path (second argument). Line 25 also shows META’s `--` operator, which forces a straight line (for the gluon). Finally, lines 26–27 draw a vertex with the `\fmfiv` command at the META coordinates specified as the second argument.

6.4.3 Extending FeynMF

Sometimes it is necessary to go beyond FeynMF’s predefined facilities and in this case it is possible to use META commands directly, either by inputting a META file or by exploiting FeynMF’s `\fmfcmd` command.

```
\fmfcmd{MFcmds}
```

The `\fmfcmd` command enters the META commands *MFcmds* directly into the output file. This facility can be useful for defining new linestyles. A META macro `style_def` is used to register the new style with FeynMF and to define a macro to be called whenever the new style is referenced, for instance as the first argument in an `\fmf` command. Such functions are called *transformers* since they take a META path as their argument and return a transformed (embellished) path. This facility has already been used to obtain the various linestyles of Table 6.5 on p. 233. For instance, you can first transform a line into a wiggly line and then add an arrow with the help of the predefined style `wiggly` to which an arrow is added:

```

\fmfcmd{%
  style_def charged_boson expr p =
  draw (wiggly p);
  fill (arrow p)
  enddef;}

```

In general, all META’s path-related commands are available. To handle color (with META-POST) you should use FeynMF’s explicit color-aware functions like `cdraw`, `cfill`, etc.

6.5 Typesetting timing diagrams

Jens Leilich and Ludwig May wrote the timing package to typeset timing diagrams for digital circuits. They developed a METAFONT alphabet of symbols and used METAFONT’s ligature mechanism to typeset logic transitions. Figure 6.7 on p. 241 shows a complete example of the use of the timing package.

6.5.1 Commands in the timing environment

The commands described in this section can only be used inside the `timing` environment; this is an extension of L^AT_EX’s `picture` environment, so that all picture-specific commands are available as well. The chosen `\unitlength` unit is 1sp.

```
\begin{timing}[symbol-type]{label-width}...\end{timing}
```

The optional argument *symbol-type* specifies which of the four timing-symbol font variants is to be used. Its value can be 1, 1s, 2, or 2s, where the digit represents the width (about one or two mm) and the letter s selects oblique rather than vertical lines to connect the signal levels. By default symbols of type 2 are used. Table 6.9 on p. 242 shows all signal symbols with their representative letters and examples in all four font variants. The mandatory argument *label-width* gives the width of the widest label describing the signals. These labels are typeset to the left of the signal lines.

```
\til{y-pos}{symbols}
```

A signal line in a timing diagram is typeset using the `\til` command; *y-pos* denotes the line position in the diagram. In most cases you should use consecutive integer values. The second argument *symbols* contains a combination of letters (see Table 6.9 on p. 242) representing various signal states. Because of how the symbol fonts are implemented, it is best to have at least two identical letters representing each state. Otherwise the ligature mechanism that draws the state transitions may not work correctly.

The *symbols* argument can also contain one of following commands:

```
\timingcounter{separation}{Start-value}{End-value}{Interval}
\conttimingcounter{separation}{Start-value}{End-value}{Interval}
```

These typeset a scale of numbers representing timing values. The second form `\conttimingcounter`, for use after an interruption, also leaves the necessary space.

```
\tin{y-pos}{text}
```

The `\tin` command describes the label of the signal line. By using the same *y-pos* value as in the corresponding `\til` command, the argument *text* is centered properly to the left of the line.

```
\tnote{y-pos}{x-pos}{text}
```

In contrast, the `\tnote` command lets you place annotations anywhere on a signal line. Again, the *y-pos* should correspond to the value in the `\til` command (however, you may want to add or subtract a bit in order to move the *text* vertically). The *x-pos* denotes the horizontal start position—the width of the symbols produced by letters in the *symbols* argument of the `\til` command is used as a unit, e.g., a value of 5 denotes the position after LHHHL.

```
\usepackage {timing}
\begin{timing}[2s]{1.4cm}
\tnote{0.5}{4}{\mathrm{T}_1}
\tnote{0.5}{12}{\mathrm{T}_2}\tnote{0.5}{20}{\mathrm{T}_i}
\tnote{0.5}{28}{\mathrm{T}_1}\tnote{0.5}{36}{\mathrm{T}_2}
\tnote{0.5}{44}{\mathrm{T}_i}\tnote{0.5}{52}{\mathrm{T}_1}
%% Clock
...1111...2222...iiii...1111...2222...iiii...1111
\tin{1}{CLK} \til{1}{LHHHLLHHHLLHHHLLHHHLLHHHLLHHHLLHHHLL}
%% Addresses line
...1111...2222...iiii...1111...2222...iiii...1111
\tin{2}{ADDR} \til{2}{VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV}
\tnote{1.85}{10}{Valid}\tnote{1.85}{22}{Invalid}%
\tnote{1.85}{34}{Valid}\tnote{1.85}{46}{Invalid}
%% Addresses status
...1111...2222...iiii...1111...2222...iiii...1111
\tin{3}{ADS#} \til{3}{HHHLLHHHLLHHHLLHHHLLHHHLLHHHLLHHHLL}
%% Write/Read
...1111...2222...iiii...1111...2222...iiii...1111
\tin{4}{W/R#} \til{4}{HHHLLHHHLLHHHLLHHHLLHHHLLHHHLLHHHLL}
%% Burst ready
...1111...2222...iiii...1111...2222...iiii...1111
\tin{5}{BRDY#} \til{5}{UUUUUUUUUUZZZZZZUUUUUUUUUUUUUUZZZZZZUUUUUUUUUU}
%% Data lines
...1111...2222...iiii...1111...2222...iiii...1111
\tin{6}{DATA} \til{6}{ZZZZZZZZZZVVVVVVZZZZZZZZZZZZZZVVVVVVZZZZZZZZZZ}
\tnote{5.85}{14}{To CPU}\tnote{5.85}{37}{From CPU}
\slsine{0.6}{0}{6.}\slsine{0.6}{8}{6.}\slsine{0.3}{16}{5.5}\slsine{0.6}{24}{1.5}
\slsine{2.1}{24}{6.}\slsine{0.6}{32}{6.}\slsine{0.3}{40}{5.5}\slsine{0.6}{48}{1.5}
\slsine{2.1}{48}{6.}\slsine{0.6}{56}{6.}
\end{timing}
```

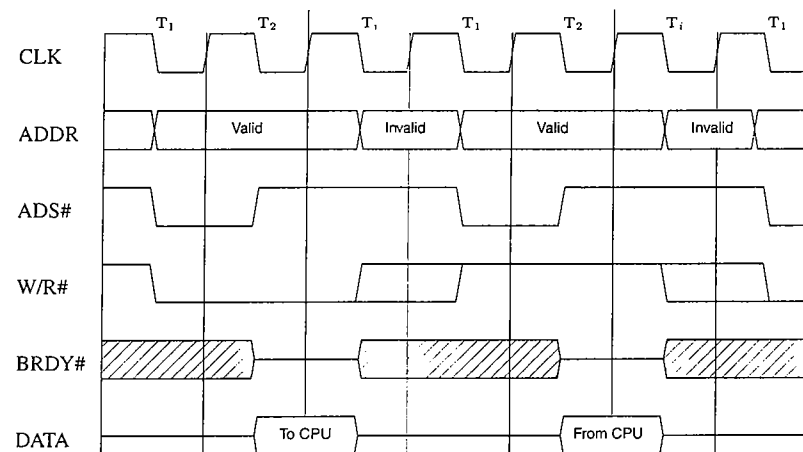


Figure 6.7: Timing diagram of a memory read followed by a memory write.

Table 6.9: Symbol combinations in all font variants.

Letter	Symbol		Font variants			
			1	1s	2	2s
L	Low level	HLLLLH →				
H	High level	LHHHHL →				
F	Floating Line (unknown level)	LFFFFH →				
l	Low level with marks	hllllh →				
h	High level with marks	lhhhlh →				
.	Empty line with marks	 →				
V	Valid bus	ZVVVVU →				
X	Bus with change of state	VVXXVV →				
U	Invalid bus	ZUUUUU →				
Z	Tristate	VZZZU →				
T	Top line with time mark	TtttTt →				
t	Top line without time mark	TtTtTt →				
B	Bottom line with time mark	BbbbBb →				
b	Bottom line without time mark	BbBbBb →				
-	Interruption sign	UUU-UU →				

`\rarw{y-pos}{x-pos}{length}{text}` `\larw{y-pos}{x-pos}{length}{text}`

These two commands produce horizontal arrows: `\rarw` points to the right, `\larw` to the left. To position such an arrow over a signal line, make the `y-pos` a little smaller (e.g., 0.6) than the line value.

`\sline{y-pos}{x-pos}{y2-pos}`

A vertical line is drawn with the `\sline` command starting at `y-pos/x-pos` down to `y2-pos/x-pos`. The width of such lines can be influenced with L^AT_EX's `\linethickness` declaration.

6.5.2 Customization

A diagram can be further fine-tuned with the commands

`\timescalefactor`

which controls the separation between lines (the default value is 2, which means there is one empty line between two signal lines that are one vertical `y-pos` apart), and

`\timadjust`

to adjust the vertical lines (the default value is 0pt). The latter command can be of help if the printer driver does not position the vertical lines properly in the middle of the state transitions. Both these values are set using `\renewcommand`.

6.6 Electronics and optics diagrams

The `circ` package by Sebastian Tannert and Andreas Tille (1996) can be used to typeset circuit and optics diagrams. This package provides a convenient way to draw diagrams containing not only resistors, capacitors, transistors, but also lenses, mirrors and the like. Symbols are coded in METAFONT, so that the output can be printed or viewed on any device.

The principles of the `circ` package are similar to those in a turtle system: all symbols and wires are drawn with respect to a "current" point that is advanced automatically, though if necessary the drawing position and direction can be set by hand. The package has commands to *draw*, *justify*, *link*, and *position* symbols and wires absolutely or relatively.

The package is subdivided into several parts that can be specified separately as options to the `\usepackage` command:

- `basic` basic symbols, such as resistors, capacitors, switches, diodes, transistors (see Table 6.10 on the following page);
- `box` blackbox, oscilloscope, generator, and amplifier;

Table 6.10: Some basic electronic circuit symbols.

(variable) resistor	<code>\R</code>		<code>\Rvar</code>	
(variable) capacitor	<code>\C</code>		<code>\Cvar</code>	
electrolytic capacitor	<code>\Cel</code>			
(Zener) diode	<code>\D</code>		<code>\ZD</code>	
light-emitting diode	<code>\LED</code>			
capacitance-variation diode	<code>\Dcap</code>			
(variable) voltage source	<code>\U</code>		<code>\Uvar</code>	
(variable) current source	<code>\I</code>		<code>\Ivar</code>	
voltmeter, amperemeter	<code>\V</code>		<code>\A</code>	
coil, switch	<code>\L</code>		<code>\S</code>	
quartz crystal	<code>\Q</code>			
incandescent lamp	<code>\La</code>			
nnp- npn-transistor	<code>\npn</code>		<code>\npn</code>	
n- p-channel FET	<code>\nfet</code>		<code>\pfet</code>	
n- p-channel VMOS	<code>\nvmos</code>		<code>\pvmos</code>	

gate logical circuits;
ic integrated circuits, such as flip-flops (see Table 6.11 on the next page);
optics optical elements, such as lenses, mirrors, etc. (see Table 6.12 on p. 246).

Table 6.11: Some integrated circuit symbols.

<code>\NRFSS</code>	<code>\DFF</code>	<code>\JKMSFF</code>	<code>\fff</code>
6-6-2 RS-flip-flop	D-flip-flop	JK master-slave flip-flop	timer NE 555

6.6.1 General circuit diagram commands

Every circuit diagram is enclosed inside a circuit environment.

```
\begin{circuit}{magstep} ... \end{circuit}
```

The argument *magstep* is an integer (in the interval 0 to 4) specifying the size of the symbols. Tables 6.10–6.12 show some of the commands available and associated symbols that can be used inside the circuit environment.

The general syntax of a drawing command is

```
\symbolname number_label_specs_dir
```

The basic option has more than sixty symbol commands to choose from. The *number* argument is an addition identifier, i.e., the combination of `\symbolname` and *number* must be unique within one diagram, so that the various circuit elements can be connected unambiguously. The direction in which a symbol is drawn is specified with the parameter *dir* (h for horizontal, v for vertical, l for left, etc.). Each symbol can be annotated by using the parameter *label* (note the position of the symbol and its annotation with respect to the current point in Tables 6.10–6.12.) The optional parameter *spec* specifies the pin position at the current drawing point (for variable resistors, transistors, etc.).

In addition to the inscription produced by *label*, the symbol is also labeled automatically with an abbreviation indexed by *number*. The former can be suppressed by using `\nv` before the command, the latter by using `\ln`. In addition, the command `\cc` exchanges the position of both labels.

```
\.number      \junction number
```

Junctions are made with the command `\.` (or alternatively `\junction`). The only parameter is a *number* identifying the junction for further reference.

Table 6.12: Some optic diagram symbols.

(half) thin lens	\SLens	X  L ₁	\HSLens	X  L ₁
(half) thick lens	\BLens	X  L ₁	\HBLens	X  L ₁
(half) concave lens	\VLens	X  L ₁	\HVLens	X  L ₁
polarizer, pinhole	\Polar	X  P ₁	\Pinhole	X  1
beam splitter	\BSplit	X 		
camera	\Cam	X 		
screens (look behind)	\ScrL	X  1	\ScrR	X  1
screens (look through)	\ScrTL	X  1	\ScrTR	X  1
laser, photometer	\Laser	X 	\PM	X 

6-6-3

len{dir} \wire_{len_{dir}}

Simple connections between symbols are drawn with the _ (wire) command. The first argument *len* specifies the length in 2.5 mm steps, while the second argument *dir* indicates the direction (l for left, r for right, u for up, d for down). Thus _8u draws a wire 8 units (2 cm) long upwards. Variants are \dashed, \bundle and \wwire, which draw a dashed wire, a bundle and a wire pair, respectively.

\htopin_{pinref} \vtopin_{pinref}

The current position is connected horizontally or vertically to the *x*- and *y*-coordinates of a given pin by the commands \htopin and \vtopin, where the argument *pinref* is the symbolic identifier of a pin of some symbol. For example, if the resistor R2 was previously defined with an \R command, then the succession of commands

\vtopin R2r \htopin R2r

draws a wire starting from the current position vertically to the *y*-position and then horizontally to the *x*-position of the right side of resistor R2.

\at_{x_y} \from_{x_y}
 \moverel_{x_y} \shift_{x_y}

The current drawing position can be changed with several commands. The first two above set the current point to the absolute coordinate (*x*, *y*) in the circ coordinate system (in steps

of 2.5 mm). In general, such absolute positioning is not desirable, since all diagrams are implicitly drawn with respect to the point (0, 0) and introducing absolute coordinates destroys the logical structure. It is much better to use the command \moverel (or \shift), which use relative positioning to move the current points *x* units right and *y* units up.

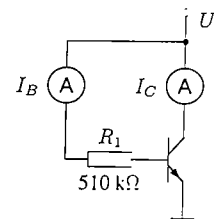
\atpin_{pinref} \frompin_{pinref}

You can move the drawing position with the command \atpin or alternatively \frompin. After executing this command the drawing continues at the given pin position: for instance, the current point can be set to the down pin of the coil labeled L4 with the command \atpin_{L4d}.

The above commands are only a few of the possibilities described in Tannert and Tille (1996). The circ package has commands to center objects horizontally or vertically with respect to other objects, commands to add text at the current drawing position and a few more bells and whistles, most of which appear in the examples below.

6.6.2 Examples

Our first example is taken from the circ reference manual, which discusses in detail how the drawing is produced; it shows a circuit for measuring the current-amplifying characteristics of an npn transistor.



6-6-4

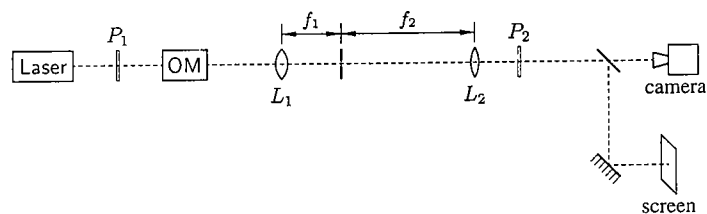
```

\usepackage [basic]{circ}
\begin{circuit}{0}
\npn1 {?} B 1    % transistor in the center
\frompin npn1C    % draw from the collector
\_ 1 u    % a little line up
\nl\A1 {\$I_C\$} v % meter for collector current
\atpin npn1B    % next draw from base
\_ 1 l    % a little line left
\<\R1 {510 k\$ \Omega\$} h % resistance on base
\_ 1 l    % one more little line left
\centerto A1    % and align on first A-meter
\nl\A2 {\$I_B\$} v % draw second A-meter
\frompin A2d    % connect second A-meter to
\vtopin R1l    % resistance.
\frompin A1u
\_ 1 u \.1    % Connection point
\frompin A2u
\vtopin .1
\htopin .1    % connect second A-meter to 1.
\_ 1 u \cc\connection1
{\$U_b\$} u    % connect to voltage
\frompin npn1E    % connect emitter to ground
\_ 1 d \GND1    % ground
\end{circuit}

```

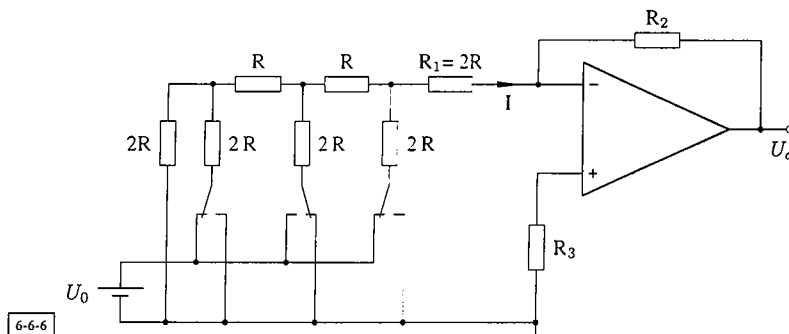
Another example from the reference manual is an experimental setup in optics, with a laser, modulator, lenses, mirrors, a camera and a screen:

```
\usepackage [basic, optics]{circ}
\begin{circuit}{0}
  \nl\Laser1 {} h           % laser
  \oa 2 r                   % optical axis
  \Polar1 {} h              % polarizer
  \oa 2 r
  \nl\OM1 P1 {} {} {} {} h % optical modulator
  \atpin OM1P3
  \oa 4 r
  \cc\BLens1 {} h           % 1. lens
  \oa 2 r
  \nl\Pinhole1 {} h         % pinhole
  \oa 7 r
  \cc\SLens2 {} h           % 2. lens
  \oa 1 r
  \Polar2 {} h              % analyzer
  \oa 3 r
  \oa 3 r
  \nl\BSplit1 {} + d        % beam splitter
  \atpin BSplit1+          % continue drawing right
  \oa 2 r
  \nl\Cam1 {camera} 1       % camera
  \atpin BLens1: \shift 0 2 \P3 % mark middle of 1. lens
  \atpin Pinhole1: \shift 0 2 \P4 % mark middle of pinhole
  \atpin SLens2: \shift 0 2 \P5 % mark middle of 2. lens
  \Dtext{\small $f_1$} from P3 to P4 % $f_1$
  \Dtext{\small $f_2$} from P4 to P5 % $f_2$
  \atpin BSplit1.          % second part of beam down
  \oa 5 d
  \Mirror1 {} * R           % mirror
  \oa 2 r
  \nl\ScrL1 {screen} h      % white screen
\end{circuit}
```

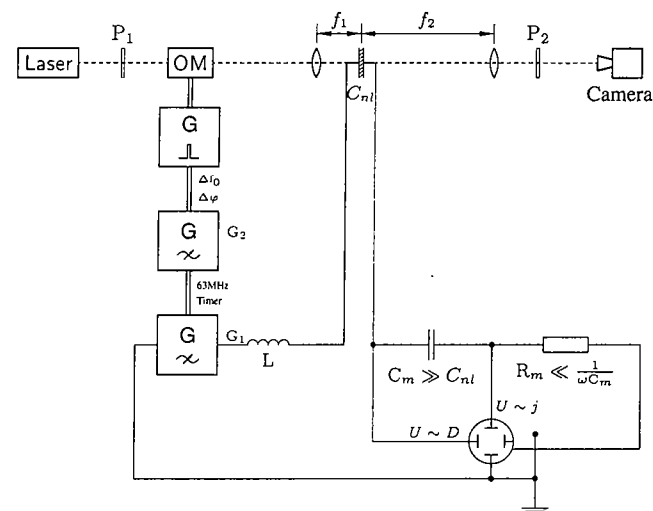


6-6-5

Two more complex examples (courtesy of the package authors) follow, given without code. The first is an operational amplifier and the second is a variant of our previous example. In this experimental setup in optics one measures the characteristics of a probe, using strobe and reference generators, an LC circuit, an oscilloscope, plus polarizer, analyzer, and a camera.



6-6-6



6-6-7

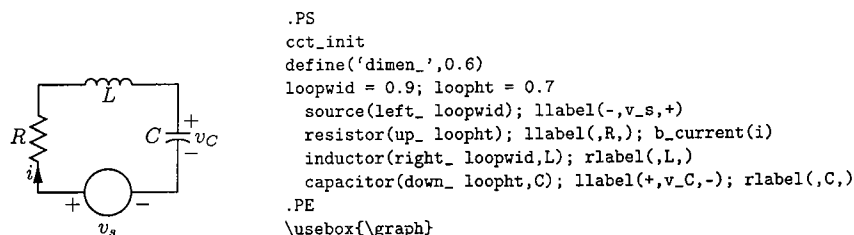
6.7 Using the m4 macro processor for electronics diagrams

A procedure completely independent of $\text{\TeX}$ and METAFONT forms the basis of the circuit_macros package written by Dwight Aplevitch. His approach uses a series of macros written for the m4 macro processor available on Unix systems. These macros generate code in the pic language, which can be handled by a pic (gpic) interpreter generating $\text{\TeX}$ input in the form of $\text{\texttt{tpic \specials}}$, as explained in Appendix A.2. Aplevitch has developed a special $\text{\texttt{tpic}}$ interpreter $\text{\texttt{dpic}}$ that can generate output for $\text{\texttt{mpic}}$ and $\text{\texttt{PSTricks}}$.⁶

6.7.1 Basic principles

The package has a number of m4 libraries, each containing a set of basic macros, that let you construct complex electronic and flow diagrams. The IEEE Standard 315 of 1975 is followed for drawing the electronics and electrical elements.

Quite complex diagrams can be composed fairly easily (the distribution contains the worked-out code of the complex example at the end of Maclean and Burns (1991)). Here we merely show how to obtain a simple circuit. The m4 code for our first example looks like this:



6-7-1

The macros .PS and .PE enclose each picture (these are the usual delimiters for the pic program). The first macro command $\text{\texttt{cct_init}}$ initializes some local variables for the circuit package (global parameters, such as line widths, page size, scaling factors). The default value for the width of the picture is half an inch. On the following two lines we set the width of the body element ($\text{\texttt{dimen_}}$) and, since we want to draw a loop circuit, we specify its width ($\text{\texttt{loopwid}}$) and height ($\text{\texttt{loopht}}$). Then we put a current source moving left and label it with plus and minus signs and v_s , draw a resistor (going up), label it R , and add a current arrow at i . This is followed by a self (inductor) at L and the loop is closed by a capacitor at C , with an indication of the voltage V_C and the polarity of the charges on the plates. Note the use of the $\text{\texttt{llabel}}$ and $\text{\texttt{rlabel}}$ commands to place texts to the left and right of the circuit element.

⁶The program $\text{\texttt{dpic}}$ does not come with the distribution, but you can contact the author if you want more information or need diagrams to be translated for use with $\text{\texttt{mpic}}$ and $\text{\texttt{PSTricks}}$.

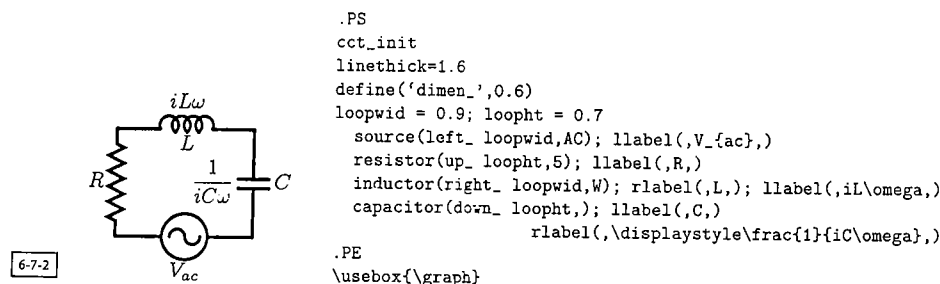
Before we can use this input with $\text{\texttt{L\TeX}}$ we have to perform several translation steps. First we have to translate the m4 instructions into pic code by executing m4, using the library $\text{\texttt{libcct.m4}}$ that contains definitions for commonly used elements. This file is then run through a pic-language interpreter like $\text{\texttt{gpic}}$ to generate $\text{\texttt{tpic \specials}}$. These can then be processed by $\text{\TeX}$ and finally interpreted by a dvi-reading program like $\text{\texttt{dvips}}$. The command sequence for this on a Unix machine would be similar to the following (depending on where the m4 files are stored):

```
m4 /usr/local/lib/m4/libcct.m4 cirexa.m4 > cirexa.pic
gpic -t cirexa.pic > cirexa.tex
```

This leaves us with a $\text{\TeX}$ file $\text{\texttt{cirexa.tex}}$ containing only the $\text{\texttt{tpic}}$ code for the example. To process it further we could include it into a $\text{\texttt{L\TeX}}$ source using $\text{\texttt{\input}}$. This stores the picture in a box register named $\text{\texttt{\graph}}$, so we have to add a $\text{\texttt{\usebox{\graph}}}$ statement into the document where we want it to appear.

6.7.2 Customizing the diagram

To show the flexibility of the circuit_macros approach, let us modify our example slightly to see how it behaves with an alternating current.

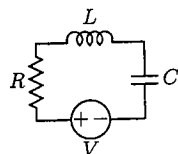


6-7-2

After specifying thick lines, we draw an alternating current (AC) source. The resistor is made a little bigger and we specify a complex value for the impedance of the self and capacitor. Note how text is placed at either side of the element with the $\text{\texttt{llabel}}$ and $\text{\texttt{rlabel}}$ commands. As the label text is set in mathematics mode, you can freely use math symbols and other specific commands for math mode (like $\text{\texttt{\displaystyle}}$ to choose a larger type size for the capacitor's numerator and denominator).

Some authors prefer to draw their circuit elements using a grid. We can write an m4 macro $\text{\texttt{grid}}$, which has two parameters $\text{\texttt{\$1}}$ and $\text{\texttt{\$2}}$ that define the x - and y -coordinates at

which the element is to be drawn.



```
.PS
cct_init

gridsize = 0.1
define('grid','(gridsize*'$1',gridsize*'$2')')

source(left_ from grid(7,0) to grid(0,0),V); llabel(V,)
resistor(up_ from grid(0,0) to grid(0,5),4); llabel(R,)
inductor(right_ from grid(0,5) to grid(7,5),W); llabel(L,)
capacitor(down_ from grid(7,5) to grid(7,0)); llabel(C,)

.PE
\usebox{\graph}
```

6-7-3

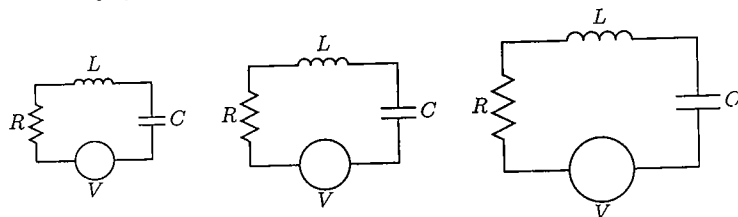
As a last simple example (a set of quite complex examples comes with the package distribution), we exploit the loop construct of the pic language. The code below also shows how to scale a diagram by expressing movements and dimensions as functions of the lengths `dimen_` (the body size of two-terminal elements, like a resistor, capacitor, etc.) and `elen_` (the default length for an element). Both these lengths are expressed as functions of the parameter `linewid`, which has a default value of 0.5 inches. As the code below shows, a variation of the latter parameter is reflected in a modification of both `dimen_` (by default equal to `linewid`) and `elen_` (by default equal to 1.5 `dimen_`).

```
.PS
cct_init

define('loop',[
  source(left_ elen_); llabel(V,)
  resistor(up_ dimen_); llabel(R,)
  inductor(right_ elen_); llabel(L,)
  capacitor(down_ dimen_); llabel(C,)
])

for linewid = 0.5 to 1 by *1.3 do { loop; move right }

.PE
\usebox{\graph}
```



6-7-4

CHAPTER 7

Preparing music scores

Preparing music scores of high quality is a very complex task, since music notation can represent a huge amount of information about the structure and performance of a musical piece. Musicians reading a score must gather information they need for the performance while playing, e.g., information about the pitch and the length of the notes, the rhythm, the articulation etc. Depending on the instrument, the musical notation may span more than a single staff (e.g., three or more for the organ), so the amount of data to be processed concurrently can be extremely large. This of course makes great demands on the musician's ability, especially when sight-reading a piece. The quality of the typesetting has an important role in this game; it can offer the musician much support by showing the structure of the piece clearly, or it can make a piece nearly impossible to play at first sight.

High-quality music typesetting requires a good eye and much experience. Until recently, this type of work has been done by highly trained music engravers who manage, according to Helene Wanske (1990), no more than one or two pages a day. As in typesetting of text, a criterion of high quality is the overall look of the page, especially the distribution of black and white. There are of course rules about how to typeset special constellations of notes, but they are not as important as the more general guidelines about the aesthetic form of the whole. In recent years several computer systems for writing scores have been developed, but they cannot yet replace an experienced music engraver. All they can do towards high-quality typesetting is create a draft: they contribute to a high-quality score only if they leave the aesthetic decisions to the *experienced user*.

In this chapter we concentrate on the MusiX_{TeX} system and some of its preprocessors. MusiX_{TeX} is a set of $\text{\TeX}$ macros that build a very powerful and flexible tool for typesetting scores. It must be emphasized that MusiX_{TeX} makes no aesthetic decisions—these must all be made by the typesetter. MusiX_{TeX} is, therefore, quite complex to use, and the preprocessors have been developed to provide an easier interface.

7.1 Using TeX for scores — an overview

Early attempts to use TeX for score preparation were made by Andrea Steinbach and Angelika Schofer (Schofer and Steinbach, 1987) and later by François Jalbert, who developed M_UTeX (Jalbert, 1989). It was Daniel Taupin, however, who really made a breakthrough by developing MusicTeX (Taupin, 1992, 1993a, 1995a,b).

Taupin's main aim in developing the MusicTeX package was to typeset complex polyphonic orchestral or instrumental music. He first thought he could extend M_UTeX to several staves, but soon decided to write a completely new set of macros, only adopting M_UTeX's METAFONT code as a starting point. However, the resulting system (MusicTeX) was still inadequate, as its one-pass system did not suffice to compute an optimal spacing of the notes. MusiXTeX, a new system, developed by Daniel Taupin together with Ross Mitchell and Andreas Egler, was derived from MusicTeX. It is backward-compatible with the earlier package, but uses a three-pass approach to optimize of slurs and the spacing between the notes¹. Since the principles and commands for writing the source are to a large extent identical, differences in notation between MusicTeX and MusiXTeX are usually irrelevant to the ordinary user and most of the following description can be used for both systems.

Beside the commands for handling a large number of instruments, for different kinds of clefs, notes, chords, beams, slurs, ornaments etc., MusiXTeX has several extension libraries containing commands that are less frequently used or are necessary only for special kinds of music. The extension libraries cover such features as:

- 128th notes;
- specialties for choral music;
- Gregorian chant;
- guitar tablatures;
- string instruments.

The MusiXTeX system is not intended to translate standard music notation into TeX, nor does it attempt to include aesthetic considerations. The system typesets staves, notes, chords, beams, slurs and ornaments by slavishly following the instructions of the typesetter (i.e., you). In the hands of a specialist, MusiXTeX is an extremely flexible and powerful tool and can generate very pleasing results.

Because MusiXTeX is quite complex, several preprocessors have been developed to provide an easier interface. These can of course only cover a subset of MusiXTeX's facilities. Each has its own main emphasis and input language. The program `midit2tex` (Section 7.5), for example, translates `mid`i data files into MusicTeX code and is therefore interesting for those who prefer playing the music on an instrument to writing notes.

At least three preprocessors are available that provide a text-based input language. `abc2mtex` (Section 7.3) uses the ASCII-format music-notation language `abc` as its input.

¹ Andreas Egler is now developing his own system, which will be *incompatible* with MusicTeX but will provide many additional features (e.g., dotted ties, slurs, and other PostScript-based features).

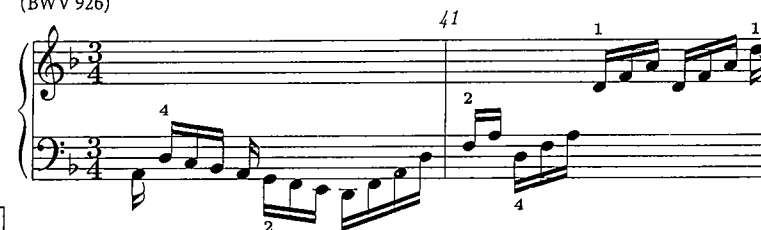
This language, developed primarily for writing down tunes, is very intuitive to read and write. It is particularly suitable if only one staff is needed, but can also handle writing guitar chords above a staff. M_Pp (Section 7.4) uses its own fairly easy-to-read input language. You may prefer M_Pp if you are writing music with several staves, since it allows you to separate them into different files. The PMX program by Don Simons (1995) supports yet another textual input language, with a built-in facility to notate figured bass.

In the next section we give an introduction to the main concepts of writing input for MusiXTeX. Because MusiXTeX is a very large system, only a few of all its features can be described. The full documentation (Taupin et al., 1996) should be consulted for all the details. The last three sections of this chapter describe how to use `abc2mtex`, M_Pp, and `midit2tex`. If you plan to use one of these preprocessors only, you can skip the next section on the first reading, as it is not necessary for understanding the preprocessors. Afterwards you should read at least Section 7.2.5 to learn how to run MusiXTeX, because you will need this in all cases.

7.2 Using MusiXTeX

7.2.1 The structure of a MusiXTeX source

The score of a musical piece, for example bars 40–41 of a prelude by Johann Sebastian Bach (BWV 926)



7-2-1

contains important information that can be retrieved only with knowledge about the surrounding context:

- The pitch of a note: this cannot be determined from its position in relation to the lines of the staff only: it depends additionally on the clef and on accidentals valid for the note. The clef is normally found at the beginning of the line but may also appear anywhere else on the staff. When considering the accidentals, you must distinguish between local accidentals valid for the bar only and accidentals valid for a larger part of the piece; the latter are found at the beginning of the current bar or in one of the previous bars.
- The time at which a note should be played: this usually depends on the sequence of notes or rests in the same bar before the note, but may be obscured if a voice changes staves as in the example above.

note sequence three	note seq. six	note seq. nine	note seq. twelve
note sequence two	note seq. five	note seq. eight	note seq. eleven
note sequence one	note seq. four	note seq. seven	note seq. ten

Figure 7.1: Sequence of score pieces coded in MusiXTeX.

The MusiXTeX input specifies the structure and kind of information in scores. But the score of a piece of music with several staves has two dimensions: the x -axis represents the time and is divided by the bars, and the y -axis is used for the staves. To quote Taupin:

The musician reads or thinks several consecutive notes (typically a long beat or a group of logically connected notes), then he goes [...] to the next instrument or voice and finally assembles the whole to build a sequence of music lasting roughly a few seconds. He then proceeds to the next bar or beat of this score.

Using this theory, Taupin implemented an input syntax that describes the score in small chunks starting with some notes in the bottom staff and then moving upwards (see Fig. 7.1). How natural this syntax is is debated in the literature and some of the preprocessors for MusiXTeX handle staves in the opposite order, i.e., top-down.

The fundamental command of MusiXTeX is

```
\notes... &... &... \enotes
```

which describes one vertical row of note sequences. The character & separates note sequences that are to be typeset on one staff of the various voices or instruments, starting from the bottom. Multiple staves are separated by the | character: for instance, the score of a song for violin and piano would be coded using:

```
\notes... |... &... \enotes
```

for each column of *groups of notes* considered a logical unit, not merely chords containing notes to be played simultaneously but also small sequences of consecutive notes that build a musical phrase. Therefore the main part of a MusiXTeX source is determined by the \notes... \enotes commands. One or more of these commands is followed by the \bar command denoting the end of a bar. MusiXTeX provides a number of variant forms, e.g., \Notes, \NOTes (see Table 7.2 on p. 260), that differ only in how they set the space between individual notes.

7.2.2 Writing notes

MusiXTeX provides many commands for typesetting score elements (Table 7.1 shows those most frequently used) that are the basis for work with this system. We concentrate here on some of the basic commands for writing notes to show the principles.

Table 7.1: Overview of MusiXTeX commands (prepared by Daniel Taupin).

Pitches

'A 'B 'C 'D 'E 'F 'G A B C D E F G H I J K L M N a b c d e

Notes, Accidentals, Accents, Clefs and Rests

\zlonga \zuga \wh \hu \hl \qu \ql \cu \cl \ccu \ccl \cccu \cccl \grcu \grcl

Accidentals:

\dqu¹²³ \yqu¹²³ \dqu² \dhqu² \doqu² \xqu² \oxqu² \roqu² \sgu² \kqu² \squ² \lsqu³ \rsqu³ \cqu⁴ \cql⁴ \chu⁴ \chl⁴

1 musixdia.tex 2 musixper.tex 3 musixgre.tex 4 musixlit.tex 5 musixext.tex

\lpz \upz \lsf \usf \lst \ust \lppz \uppz \lsfz \usfz \lpzst \upzst \downbow \upbow \flageolett \whp \qupp

Accent on beam with prefix b and beam reference number instead of the pitch

\trebleclef \bassclef \smallaltoclef \drumclef² \gregorianCclef³ \oldCclef⁴

\qqs \hs \qs \ds \qp \hpause \hpausep \pause \pausep \PAUSE \PAUSEp \Hpause⁴

Other Symbols

\Trille \trille \shake \Shake \mordent \Mordent \turn \backturn \Shakel \Shakesw \Shakene \Shakenv

\allabreve \meterC \reverseC \reverseallabreve \meterplus \duevolte \l[r]par

\setvoltabox \setvolta \coda \Coda \segno \Segno \caesura \cbreath \PED \sPED \sDEP \sDEP

\metron

\fermataup \arpeggio d5 \uptrio \octfinup \slide⁵ \boxit A \circleit B

\fermataup \fermataup \bracket

\fermatadown \fermatadown \downtrio \octfindown \leftrepeat \rightrepeat \lefttrirepeat \righttrirepeat

Specifying note pitches

The pitch of notes is denoted by letters (see Table 7.1 on the previous page). It is important to note that this specification does not depend on the current clef. This is one deviation from the MusiX_{TeX} principle that the representation of information about a musical piece is normally very similar to the way it is represented in the score itself. The reason for this feature is so that you can change the clef e.g., for a few bars only, without having to correct all the notes affected by it. The position of the notes in the staves of the typeset score is automatically adjusted to the new clef.

Accidentals must be specified separately and are handled very similarly to the score itself. To put a flat or sharp before a certain note, e.g., with pitch *c*, you write `\fla c` or `\sh c` or simply `_c` or `^c`, respectively. To define the key signature of the whole piece or a part of it, the command `\generalsignature` is used, which has a number as argument, positive for sharps and negative for flats.

Note symbols

In order to build a note the pitch specification must be combined with a command specifying the duration of the note and the direction of the stem. The most frequently used note commands, which also set a single space after the note, are as follows:

- `\wh p`: whole note at pitch *p*;
- `\hu p`: half note at pitch *p* with stem up;
- `\hl p`: half note at pitch *p* with stem down;
- `\ha p`: half note at pitch *p* whose stem direction is chosen automatically according to the convention used for melodies: the stem points upwards if it is below the third line of the staff otherwise it is stemmed down;
- `\qu p`: quarter note at pitch *p* with stem up;
- `\ql p`: quarter note at pitch *p* with stem down;
- `\qa p`: quarter note at pitch *p* with automatic stem;
- `\cu p`: eighth note at pitch *p* with stem up (the letter *c* in this and the following commands comes from the French word “croche,” meaning eighth note; *cc*, *ccc* stand for “double croche,” “triple croche,” etc.);
- `\cl p`: eighth note at pitch *p* with stem down;
- `\ca p`: eighth note at pitch *p* with automatic stem;
- `\ccu p`: sixteenth note at pitch *p* with stem up;
- `\ccl p`: sixteenth note at pitch *p* with stem down;
- `\cca p`: sixteenth note at pitch *p* with automatic stem;
- `\cccu p`: 32nd note at pitch *p* with stem up.

- `\ccccl p`: 32nd note at pitch *p* with stem down.
- `\ccccc p`: 64th note at pitch *p* with stem up.
- `\cccccl p`: 64th note at pitch *p* with stem down.

With these commands available we can typeset a first simple example:

```
\usepackage {musixtex}
\begin{music}
\startextract
\Notes\qu c\cu d\cu e\ccu f\ccu g\ccu h\ccl i\enotes\bar
\Notes\cca j\cca i\cca h\cca g\ca f\ca d\qa c\enotes
\endextract
\end{music}
```



As we are using L^AT_EX, everything must be placed into a music environment (if plain T_EX is used the wrapper is slightly different). The coding of the notes starts with command `\startextract` and is ended by `\endextract`. If you are writing a whole piece, use `\startpiece` and `\endpiece` instead.

To get a dotted quarter with pitch *c*, you can write `\qu{.c}` or use the command `\pt` and write `\pt_c\qu_c`. For rests, the following commands are provided among others: `\pause` for a full bar rest, `\hpause` for a half-note rest, `\qp` for a quarter-note rest and `\ds` for an eighth-note rest.

The note symbol commands introduced above are not enough, however. To handle chords you need commands that do *not* add a space after them, and to typeset chords with very small intervals, you need to be able to put the head of the note to the right or left side of the stem. For these purposes there is a set of non-spacing commands like the commands shown above but starting with `\z...`, and a set of commands for shifting note heads to the right or left starting with `\r...` or `\l...` respectively.

In typesetting chords, use non-spacing commands for all notes in the chord but the last. If the last note is stemmed, one stem for the chord is produced that links all the simultaneous notes:

```
\usepackage {musixtex}
\begin{music}
\startextract
\Notes\zw c\zw e\zw g\wh j\enotes\bar
\Notes\zh c\zh f\zh h\hu j\zh d\rh f%
\zh g\hu i\enotes\bar
\Notes\zw c\zw e\zw g\wh j\enotes
\endextract
\end{music}
```



Table 7.2: Variant forms of the `\notes` command

Notation	Spacing	Suggested use
<code>\znotes ... & ... & ... \enotes</code>	0 <code>\elemskip</code>	special
<code>\notes ... & ... & ... \enotes</code>	2.0 <code>\elemskip</code>	16th
<code>\notesp ... & ... & ... \enotes</code>	2.5 <code>\elemskip</code>	dotted 16th
<code>\Notes ... & ... & ... \enotes</code>	3.0 <code>\elemskip</code>	8th
<code>\Notesp ... & ... & ... \enotes</code>	3.5 <code>\elemskip</code>	dotted 8th
<code>\NOTes ... & ... & ... \enotes</code>	4.0 <code>\elemskip</code>	quarter
<code>\NOTesp ... & ... & ... \enotes</code>	4.5 <code>\elemskip</code>	dotted quarter
<code>\NOTes ... & ... & ... \enotes</code>	5.0 <code>\elemskip</code>	half
<code>\NOTesp ... & ... & ... \enotes</code>	5.5 <code>\elemskip</code>	dotted half
<code>\NOTes ... & ... & ... \enotes</code>	6.0 <code>\elemskip</code>	whole

7.2.3 Note spacing

Positioning the notes in an aesthetically pleasing way is a complex matter. Within a column of notes the internote spacing is not necessarily constant, since it ideally depends on the duration of the shortest of the simultaneous notes. However, this is not an absolute rule, since spacing does not depend exclusively on the local notes, but also on the context, at least in the same bar, and exceptions to the rule are easily found.

MusixTeX, therefore, handles spacing in several steps. The first step is to specify the size of the spaces in the source. For this the sequence `\notes ... \enotes` is used. There is a set of variants on this command, all working in the same way but setting a different amount of space (`\noteskip`) between consecutive notes (see Table 7.2). In each of these variants all typeset spaces are initially the same size. With the command `\sk` you get an extra space with a width equal to `\noteskip` between consecutive notes. By default the length `\noteskip` increases linearly from `\notes` to `\NOTes`. This can be changed to a geometric progression, in which `\Notes` is $\sqrt{2}$ wider than `\notes`, etc., by the command `\geometricsskip`. In fact, both the basic spatial unit (`\elemskip`) and the note-specific spacing (`\noteskip`) can be freely adjusted, as explained in Taupin et al. (1996).

The second step is done by the `musixflx` program, which takes the default value of `\elemskip` and computes for each line a new value that is the one used for the typesetting. This is done to make sure that all lines are filled with bars, and to prevent a line from breaking in the middle of a bar. In Section 7.2.5 we take a closer look at this procedure.

To illustrate how the spacing commands work, here is the example from above but with proper spacing:

```
\usepackage {musixtex}
\begin{music}
\startextract
\NOTes\qu c\enotes \Notes\cu d\cu e\enotes
\notes\ccu f\ccu g\ccu h\ccl i\enotes \bar
```

```
\notes\cca j\cca i\cca h\cca g\enotes
\Notes\ca f\ca d\enotes \NOTes\qa c\enotes
\endextract
\end{music}
```



```

\Notes \zq{N}\ql{b}|\zq{g}\qu{i}\enotes\bar
\Notes
\isluruON\iblON{-2}\qb0{N}\tslurOM\qb0{M}%
\isluruOM\qb0{M}\tslurOL\tbl0\qb0{=L}%
|\ibu0j0\zqb0{g}\rq{h}\qb0{j}\zqb0{g}\rq{h}\qb0{j}%
\zqb0{g}\rq{h}\qb0{j}\tbu0\zqb0{g}\rq{h}\qb0{j}%
\enotes\bar
\Notes \ql{k}\ql{K}|\zq{f}\zq{h}\qu{k}\zq{f}\zq{h}\qu{k}\enotes
\endextract
\end{music}

```



7-2-5

The code for the above score looks quite complicated. The reason for this are the slurs and beams. For each beam or slur two commands are needed, one for the beginning (which must appear before the first note belonging to it) and one for the end (which must appear after the last note belonging to it).

For *slurs* the most commonly opening command is `\isluru np` if the slur is to be above the notes and `\islurd np` if below them. The two parameters of this command specify the reference number of the slur² and the pitch to which the beginning of the slur belongs. The accompanying terminating command is `\tslur`, which takes as parameters the slur reference number and the pitch at which the slur is to terminate.

Frequently used opening commands for *beams* are `\ibu` for beams above the notes and `\ibl` for beams below them. To produce double, triple, or quadruple beams just double, ... the number of b's in the command, e.g., `\ibbu` for double beams above the notes, etc. There are also commands for repeated beam patterns and semi-automatic beams whose slope is computed. It is even possible to print beams across bars.

These commands have three parameters: the reference number, the pitch to which the beginning of the beam belongs and the slope, an integer in the range $[-9, 9]$ that creates a slope between -45% and 45% . The beam is drawn three lines above or below the line of the referenced pitch. To end a beam the commands `\tbu` or `\tbl` respectively are provided, which take as an argument only the reference number.

The opening and terminating command are not enough to create a beam: the notes belonging to it must be connected to it. For this the note command `\qb` is provided, which works like `\qu` or `\ql` but has as additional argument the reference number of the beam to which the stem is connected.

²The reference number is normally set to 0 if there is no more than one simultaneous slur in a staff.

7.2.5 Running MusiXTeX

As remarked above, MusiXTeX was derived from MusicTeX in order to produce “beautiful scores” more easily. Whereas MusicTeX is a single-pass system, MusiXTeX is a three-step system.

As the first step $\text{\LaTeX}$ or plain $\text{\TeX}$ ³ is run on the file containing the MusiXTeX source. This produces a file with the extension `mx1` containing relevant information about the music piece, e.g., distances between notes, etc. In the second step the optimal between-note spacing is computed by the external program `musixflx`. This program reads the `mx1` file and writes a new file with the extension `mx2` containing proper settings for `\elemskip` for every line of the score. As the third step ($\text{\LaTeX}$) is run again on the source file. This time MusiXTeX finds the corresponding `mx2` file and uses the information contained therein.

MusiXTeX might in the first step find an old `mx2` file from some previous run containing incorrect information. In such a case the `mx1` produced is incorrect as well and thus the output after the third step will not be good either. Thus after changes to the source file, or when you get inscrutable error messages or unexpected output, delete all the auxiliary files and start again.

As a small example of the full sequence, consider the following MusiXTeX source:

```

\documentclass{article}
\usepackage {musixtex}
\pagestyle{empty}
\begin{document} \begin{music}
\instrumentnumber{1} \generalmeter{\meterfrac{3}{4}} \nobarnumbers
\startpiece
\Notes\qp\zcharnote N\ff\ql{lm}\enotes\bar %1
\Notes\isluru0m\ql{m}\tslur0l\ql{1}\ql{p}\enotes\bar %2
\Notes\ibl0l0\qb0{p~np}\enotes
\Notes\nbb10\isluru0m\qb0{m}\tbl0\tslur0l\qb0{1}\enotes
\Notes\qb0p\tbl0\qb0k\enotes\bar %3
\Notes\isluru0k\ql{k}\tslur0j\ql{j}\ql{o}\enotes\bar %4
\Notes\ibl0l0\qb0{mkpnr}\tbl0\qb0m\enotes\bar %5
\Notesp\qlp{1}\enotes\Notes\ibl0n{-4}\qb0{n1}\tbl0\qb0j\enotes\bar %6
\Notes\ibl0l0\qb0{omokr}\tbl0\qb0m\enotes\bar %7
\Notesp\qlp{p}\enotes\Notes\cl{n}\enotes\Notes\itenu0r\ql{q}\enotes\bar%8
\Notes\ibl0q{-3}\tten0\qb0q\tbl0\qb0o\enotes
\Notesp\Uptext{\it tr}\isluru0p\qlp{p}\enotes
\Notes\tslur0q\cl{q}\enotes\bar %9
\Notes\ql{q}\enotes \Notes\hpause\enotes
\endpiece
\end{music} \end{document}

```

³What $\text{\TeX}$ flavor to use depends on the content of the file. The examples in this section are set up to use the $\text{\LaTeX}$ interface; this is a sensible choice if music and text are to be mixed. However, if only scores are to be produced, the additional functionality of $\text{\LaTeX}$ is rarely needed, so most preprocessors produce files to be run with plain $\text{\TeX}$.

Processing this with $\text{\LaTeX}$ for the first time yields the following output, with all lines containing equal (and thus incorrect) element spacing.



If we then process the generated `mx1` file with `musixflx` and afterwards reprocess the source we get the final masterpiece (the first bars of Sonata Sesta for treble recorder and basso continuo by Francesco Maria Veracini):



7-2-6

7.3 abc2mtex — easy writing of tunes

If you want to write melodies, perhaps with some chords as accompaniment, then the $\text{Musix}\text{\TeX}$ preprocessor `abc2mtex` by Chris Walshaw (1996) could be a good choice. This system converts the music notation language `abc` developed by Walshaw into $\text{Musix}\text{\TeX}$ source.

With `abc2mtex` you need make no decisions about the layout at all; you simply enter the musical information in `abc` notation which is comfortably simple. The program then uses some internal defaults⁴ to create $\text{Musix}\text{\TeX}$ code.

The `abc` music notation was developed as a computer-readable ASCII form that people can easily read and even play. It was designed originally for folk and traditional tunes of western Europe that can be written on one staff. Since its introduction in 1991, it has become widely used; several collections of tunes written in `abc` notation are available on the Internet. About a dozen other programs use `abc`, e.g., to convert `abc` into MIDI-files or vice versa, or to play the music directly through the speakers of a computer.

The version of `abc2mtex` described in this book is 1.6. The program is still under active development, however, and the reader is advised to obtain the latest release, since some of the features described below are not available in previous releases.

7.3.1 Writing `abc2mtex` source

To see how `abc2mtex` source is built up, consider the following example:

7-3-1

M: 2/4
L: 1/8
K: Gm
g f e d | c d B2 | c c c c | d2 d2 |]

The `abc2mtex` source for a tune has two parts, header and body. The header contains information fields, each starting with an uppercase letter to denote the kind of information, followed by a colon. The body consists of the music piece itself. Within the body, further information fields can be inserted that are used for changes to the header information, e.g., of key, meter or tempo.

There are 26 possible information fields, most of them optional.⁵ Some of the commoner ones are:

- Musical information:
 - K: the key. You can use major keys (e.g., K:E), minor keys (K:g), Mixolydian (K:AMix) and Dorian modes (K:EDor);
 - M: the meter, e.g., M:3/4, M:C, or M:C|.
- Technical information:
 - E: the value of `\elemskip` in $\text{Musix}\text{\TeX}$, i.e., to manipulate the internote spacing in the output;
 - X: the number of the tune in the file; mandatory if the file contains more than one tune.

⁴These defaults are stored in the file header .tex and thus can be manipulated if desired.

⁵A complete list can be found in Walshaw (1996).

- General information such as T: title (mandatory), C: composer, D: discography, H: history, O: origin, S: source.

The order of the fields is not important except that T: must come first (or second, if there is an X: field) and K: must come last since it denotes the start of the body (and thus is mandatory as well).

Returning to the above example, you can easily identify the *bars*, separated by |. *Double bars* are denoted by ||, [|, or]| and *repeats* are generated by :|, ::, or |:.. First and second repeats are obtained by using [1 and [2.

For each of the *notes* you write the letter used in music. Uppercase letters are used for low notes, lowercase letters for higher notes; and a right quote (') is suffixed to a lowercase letter to represent the next higher octave, while suffixing the uppercase letters with a comma denotes the next lower octave. This too is very familiar to musicians. To obtain a rest the letter z is used.

7-3-2

G, A, B, C D E F G A B c d e f g a b c' d' e' f' g' z

But how is the *length of the notes* specified? There is a default note length, 1/16 or 1/8, that depends on the meter specified. The sixteenth note is used if the meter is less than 0.75 and the eighth note otherwise, so for a 4/4 meter the default note length is an eighth note. This default note length can be changed with the information field L:. The next example shows the automatic change in the default length when the meter switches from 7/8 to 2/4:

7-3-3

M:7/8
K:C
CDEF GFE | \
M:2/4
CDEF GFED |[1 CEGE GEGE :| [2 CEGE C4 |]

By default every source line produces one line of the score. This can be prevented by finishing the line with a \, as in the previous example.

Notes or rests of different lengths are obtained by writing a factor after the letter designating the note. For example, if the meter is 4/4 then the length of G (or G1) is 1/8, the length of G2 is 1/4 and the length of G/2 is 1/16.

7-3-4

M:4/4
L:1/4
K:C
C D E F | G/2 F/2 E/2 D/2 C2| z4 |]

Accidentals are produced by prefixing a note with the symbols ^ (sharp), = (natural), or _ (flat), and double sharps and flats by ^^ and ^^_. There are also simple readable conventions for dotted rhythm:

7-3-5

M:2/4
L:1/8
K:C
^F<G ^^F<^G | ^G2>=G2 |
_A<<G _A<<_G | F2>>^F2 | G4 |]

and for duplets, triplets etc.

7-3-6

M:3/4
K:C
(3CDE F2 G2 | (2FE (4gabC | C6 |

As said before, when you write abc2mtex source, you make no decisions about the layout, since the program uses its own defaults. You can, however, use some facilities to influence the layout locally, such as:

- The information field E: can be used to set or change the MusiX_{TeX} parameter \elemskip.
- A *beam* is generated between notes if the letters in the source are written with no intervening space. The height and slope of the beam are set automatically and cannot be influenced.
- *Slurs* are generated by placing parentheses around the notes to be affected (with no space directly after the opening and before the closing parenthesis). *Ties* between two notes are obtained by using -, which even works across bars.

The next example shows beams, slurs, and ties as well as staccato marks, the latter being generated by placing a dot before the note:



M: 6/8
K: C
(g~f).e (ed)~c- | (~c3 d) z g- |
(g3-gec) | .B.c.A G3 |

7-3-7

Chords are coded using bracket signs. They can be grouped in beams but cannot contain spaces. Guitar chords can be put beneath the melody staff; they are set between double quotes.



M: 4/4
L: 1/4
K: G
"D" [FA] | "G" G4 | "D" [F4A4] | \\
"Em" [(B/2E4)A/2 B) - B2 |

7-3-8

One can also produce *grace notes* by enclosing notes in curly braces. Such notes have no time value and thus expressions like {c3} or {c>d} are illegal. The following example from the tune "Athol Brose" exhibits complex Highland pipe gracing in all its glory:



M: 4/4
K: D
{g}c<{GdG}e {gf}ga {f}g>e {g}f>d | \
{g}e/f/g {Gdc}d>c {gBd}B<{e}G {dc}d2 |

7-3-9

abc2mtex can also be used for pieces with more than one voice, although, according to the author "[this] is to be regarded as an essentially unsupported feature."

7.3.2 Running abc2mtex

abc2mtex is available for Unix and DOS. For DOS an exe file is provided in the distribution, but Unix users have to compile it from the C source code.

The program can be called without arguments, in which case it asks questions interactively. Alternatively, it can be called as follows:

abc2mtex options tune:selection

In this case one or more tunes or musical pieces contained in *tunefile.abc* are converted to MusiX_TE_X-source and written to a file called *music.tex*. This file can then be processed by MusiX_TE_X as explained in Section 7.2.5, to get the final printed score. *selection* indicates which tunes you want; - stands for all tunes. You can give a comma-separated list of tune numbers or ranges; e.g., if you want piece 1, pieces 3 to 5 and 7 onwards, *selection* would be 1,3-5,7-. The following options exist:

- x generate output for MusiX_TE_X; without this option output for Music_TE_X is generated;
- o *name* name of the output file, if different from *music.tex*;
- t *number* transpose the piece to another key while processing it;
- i generate an index of all the pieces of music in the file.

7.4 M_P, a MusiX_TE_X preprocessor

Jan Nieuwenhuizen developed the MusiX_TE_X preprocessor M_P (Nieuwenhuizen and Nienhuys, 1996) to make MusiX_TE_X easier to use; he characterizes it thus:

Compared to MusiX_TE_X, M_P is quite a simple program, but it has the potential to make MusiX_TE_X much more *accessible* if not *usable*. Its goal is not to cover all of MusiX_TE_X's possibilities, but to make the coding of simple pieces (or parts) of music a lot easier.

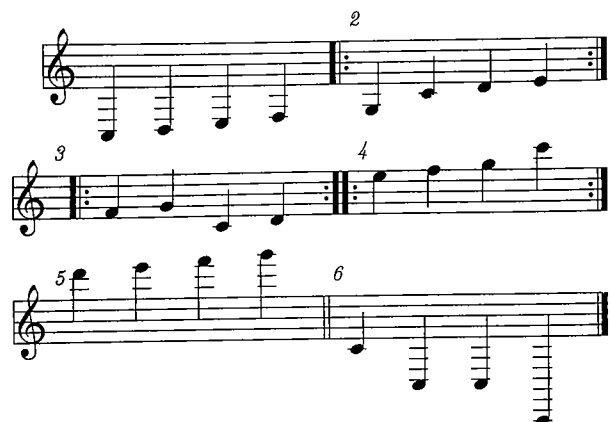
The concepts are in many ways similar to those of abc, and the resulting code is similar as well. One big difference is the way the coding of multiple staves is handled. In MusiX_TE_X the music piece is split up into very small pieces, often less than one bar. These pieces are coded sequentially, first the notes of the bottom staff and then the notes of the next higher staves. This is one reason why it is very hard to read the code. For M_P a different method is used to handle the two dimensions of a score: a separate file is written for each staff, and M_P combines them to produce MusiX_TE_X source.

7.4.1 Writing M_P-source

To typeset one note four kinds of information can be given to M_P: *octave*, *notename*, *duration*, and *features*. Only the *notename* is mandatory. All this information is coded in the given order with no intervening spaces; a blank separates two notes.

Notes are denoted by the normal letters: lowercase letters are used for the upper octave and uppercase letters for the lower octave. A left quote (') in front of the letter transposes the note one octave deeper while a right quote (') makes it an octave higher. To generate different clefs, use the command \clef followed by either \treble, \alto, \tenor or

\bass. The following example shows some pitches and how to generate different kinds of bar lines.



```
C D E F | : G c d e :||:
f g c d :|: 'e 'f 'g ''c :|
''d ''e ''f ''g || c 'c C 'C |||
```

7-4-1

In contrast to abc2mtex, accidentals are denoted by extra *notenames*, such as *gis* or *bes*, not by prefixing the base note with a symbol.⁶ When writing music with M_P you define the key of the piece using the command \key. For a single note you need only write the correct notename—M_P sets the corresponding accidentals (depending on the key you have chosen):



```
\key\A
A cis c cis |
fis f g gis | a1 |||
```

7-4-2

As *durations* halves, quarters, eighths, 16ths, 32ths, and 64ths are provided, specified by placing the corresponding number after the note name. For dotted notes a dot can be put after this number. A default note length can be set with the command \duration, that is then used for all notes for which no length is specified. If the \duration is not set the default note length is a quarter.



```
\duration8 | c4 d4 e4 f4 |
g g g16 g16 g16 g16 a4. a1.
```

7-4-3

⁶The names of the accidentals are based on the Dutch-German system: the suffix “is” indicates a sharpened note, e.g., “cis” means “c sharp”, and the suffix “es” indicates a flatted note, e.g., “ges” is “g flat”.

This representation of note length has its limitations, as you cannot represent duplets, triplets, etc., in a natural way. However, as we see below, these can be specified by using a note-grouping mechanism.

Other information, such as typesetting details and accents or more generally sub- or superscripts, can be added to the end of the code for a note. Subscripts and superscripts are added in the way familiar to L^AT_EX users, with _{and ^{. The parameter of a sub- or superscript can contain some text or a *macro*; \tr (a trill), \upbow, and \downbow are currently available.}}

To manipulate the typesetting of notes, M_P provides, among others, the following commands:

- \stemdown, \stemup, \stemno: to determine the kind of stem;
- \headleft, \headright: to determine at which side of the stem the note is to be printed;
- \grace: to produce a grace note;
- \normal, \small, \tiny: to set the size of the following notes.

```
\key\F
\duration8
f\grace 'f{-} 'c\stemup{-} 'a{-} \tiny 'f c \normal a_{.} |
g_{-} \small g \tiny g \normal g4. |
```

7-4-4



Sometimes you may want some empty space in your staff to have room for annotation. You can obtain this by inserting a note named *s* (for *script*) in your source. *s* is handled very similarly to note, as it has a duration (which determines the space it generates) and can have sub- and superscripts. To insert a *rest*, put in a note named *r*.

```
r1 | r2 r2 | r r r8 r8 r8 s1_{room for ...} r16 r16 |.
```

7-4-5



room for ...

A *beam* is obtained by enclosing all notes belonging to it in brackets, e.g., [c8 d8 e8 f8]. For *chords*, braces are used and for *slurs*, parentheses. The opening bracket for a slur is written after the first note and the closing one before the last note, e.g., to generate a slur including the notes c d e f, write c(d e)f. Normally the direction of note stems

within a beam are at the system's discretion, but they can be forced in a particular direction with [$\wedge$...] or [$\vee$...]. In a similar fashion, slurs can be adjusted by suffixing one or both parentheses with a + or - sign.

```
\key\Bes
\duration8
\meter{2/4}
['g( )'f 'es( )'d ] | ['c( )'d ] {g4_{-} bes4} | [{g a\headright 'c}
{g a\headright 'c} {g a\headright 'c} {g a\headright 'c}] | {f4_{-} a4
'd4} {f4_{-} a4 'd4} |.
```



7-4-6

To generate duplets, triplets, etc., with beams, put the notes belonging to the x -plet in brackets and add / x after the closing bracket. x -plets without beams are not easy to write, but they are possible, as the following example shows.

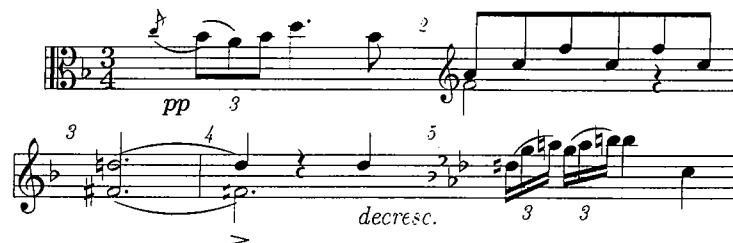
```
\meter{2/4} \duration{8}
['c( b )a]/3 f4 |
f4/3( a4/3_{\xpletstyle 3}} ) g4/3 |.
```



7-4-7

M $\text{\P}$ has several other features and the distribution comes with a number of example files that are worth studying. Below is a last example showing some more of its functionality.

```
\meter{3/4}
\key\F \clef\alto \duration8
'c\grace\stemup( ) bes_{pp}( ) a bes ]/3 'd4. bes |
\clef\treble
[ ^ { a f2 } 'c 'f 'c { 'f r4 } 'c ] |
{ fis2.(v ='d2.\stemup( ^ ) } |
{ } 'd4 ) =f2._> { r4 s4_{decr} } 'd4\stemup \key\As ||
\duration{16}
[ 'd( 'g )'a ]/3 [ 'g( 'a )'b ]/3 'b4 'c4 |.
```



7-4-8

7.4.2 Running M $\text{\P}$

To run M $\text{\P}$ you must first create one or more files with the extension mpp. Running them through the program mpp then generates a single file with the extension tex that can be processed further using MusiX $\text{\TeX}$.

The parameter to mpp is a space-separated list of the files that each contain one staff of the piece. The order of the files determines the order of the staves in the output. In contrast to MusiX $\text{\TeX}$, the staves are ordered top-down. If two of these staves are to be tied together, e.g., for piano, put a comma between the corresponding file names on the command line. Thus the command

```
mpp violin1 violin2 pianoR , pianoL
```

compiles the files violin1.mpp, violin2.mpp, pianoR.mpp and pianoL.mpp into one $\text{\TeX}$ input file called violin1.tex—i.e., the name is created from the name of the first input file. This file can then be further processed by the MusiX $\text{\TeX}$ system, as explained in Section 7.2.5.

Our moderately complex example from Section 7.2.4 is much easier to write with M $\text{\P}$. We have already developed the code for the upper staff in Example 7-4-6. The code of the lower staff is as follows:

```
%file pianoL.mpp
\key\Bes \duration8 \meter{2/4} \clef\bass
['G( )'F 'Es( )'D ] | ['C( )'D ] {G4 Bes4_{-}} |
[G( )F F( )E ] | D4_{-} D4_{-} |.
```



7-4-9

So if you run

```
mpp pianoR , pianoL
```

you get the following output:



Perhaps the output of the example directly coded in MusiX_{TEX} is a bit nicer (see Example 7-2-5). The distance between the notes and beams in bar 1 is a bit too small here and beams with a slope look better. For better quality, however, you can just edit the MusiX_{TEX} code generated by M₂_T. This takes much less time than coding the whole piece in MusiX_{TEX} directly.

7.5 midi2tex

A different kind of preprocessor is H.J.P. Kuykens's midi2tex (1992), which converts type-1 MIDI files into Music_{TEX} format. There is no version for MusiX_{TEX}, but since MusiX_{TEX} provides a mode for backward compatibility midi2tex can also be used with MusiX_{TEX} if you pay attention to a few things described in detail in Taupin et al. (1996).

MIDI (Musical Instrument Digital Interface) is an international standard for coupling electronic instruments and computers. Many sequencer programs are available to record and edit MIDI data, and they can be used to generate the MIDI file you need to typeset a music piece via midi2tex.

MIDI data consist of a series of messages of two or three bytes. Each message contains information about the command, for example, *note on* to denote the beginning of a note or *note off* defining the end of a note. Besides this, a track number from 1 to 16 is defined and parameters for the command are given, for example, which note is to be put on or off. MIDI data is very much oriented toward a keyboard where the keys are numbered from 1 to 128. Thus for MIDI there is no difference between G sharp and A flat (*gis* and *as*).

7.5.1 Running midi2tex

midi2tex is available for DOS and Atari systems. The program is called with the following syntax:

```
midi2tex inputfile outputfile [option] [option] ...
```

The input file is expected to be a type-1 MIDI file. Some problems arise when using type-0 files; if such a file is used, midi2tex displays a warning that the result may be unpredictable, but Kuykens nevertheless encourages users to experiment with them.

One problem with MIDI files generated by a sequencer recording keyboard or another MIDI-interfaceable instrument is that the timing may not be very exact. To obtain a good score with correct note beginnings and lengths, a "clean" MIDI file is needed. It is worth taking a lot of care to get a MIDI file that is as good as possible in order to prevent a lot of mistakes in the Music_{TEX} file.

Sequencers often provide functions to support this task. If not, midi2tex's option `-q` followed by a number can be used to *quantize* the output. This number then specifies the length of the shortest note (or rest) in the music piece. If you write, for example, `-q8`, no note shorter than an eighth occurs in the generated score. The following values can be used: 1 (whole note), 2 (half note), 4 (quarter), 8, 16, 32, and 64.

To get debugging information the option `-d` is provided. It must be followed by either `SCREEN`, `FILE` or `PRINTER`, which specifies where the information is directed to.

Most of the other options of midi2tex let you determine how many staves the output has, how the MIDI data is distributed into them, what key and clef are used, etc.

There is a group of options which specify how the *tracks* of the MIDI file should appear in the output. If none of them is used, a staff is generated for each track. Track 1 is omitted by default, since in a type-1 MIDI file it does not normally contain a voice.

- `-otracklist` Change the order of the tracks to *tracklist* (comma-separated list of tracks). But be sure to include all the tracks in the list, as otherwise midi2tex stops with an error.
- `-stracklist` Skip the tracks specified by *tracklist*.
- `-itracklist` Assign the tracks specified with *tracklist* to one instrument, e.g., to the right and left hands of a keyboard.

Another group of options lets you specify the *clefs* used for the staves; the default is the treble clef:

- `-btracklist` Set the bass clef for all tracks in *tracklist*.
- `-akey tracklist` Set an alto clef for tracks in *tracklist*, where *key* is 1, 2, 3, or 4 denoting one of the four alto clefs.
- `-knumber` Set the *key* of the music; *number* must be between -8 and 8, and denotes the number of sharps (positive number) or flats (negative number). Flats and sharps are inserted according to the key chosen.

You can also add text to the Music_{TEX} file:

- `-ttracklist` Add to the output file text that comes from a separate file with the same name as the MIDI file but the extension `.txt`. The *tracklist* specifies the tracks under which the text is to be inserted. In the text file, the contents are divided by | characters to distinguish the bars. Text for more than one track is separated by /.

Finally, there are a variety of options to influence how Music_{TEX} typesets the music, including the following:

- `-f` Generate all slopes of beams with value 0.
- `-edimension` Set the `\elemskip` parameter of Music_{TEX}.
- `-mnumber` Set the music size; *number* can be 16 (16pt) or 20 (20pt).

CHAPTER 8

Playing games

Board and card games have a long history, and thousands of books in hundreds of languages have been dedicated to chess, Go, cards, and the like. These books almost always use diagrams to explain the rules or show the evolution of a game. In the present chapter we look at a number of examples showing how to prepare such graphical presentations with $\LaTeX$.

Most game packages are concerned with making available either a special font for typesetting the right symbols or macros for producing nice examples of the state of play. The highly developed field of chess notation, however, with which we begin, lends itself well to an algorithmic typesetting system like $\LaTeX$. The chess packages keep track of the state of moves and allow various forms of output.

We move next to the rather similar games of Chinese chess, and Go, followed by backgammon. We then look at cards, where the classic game, bridge, has a special package, before concluding the chapter with the esoteric subject of crossword puzzles. Although crossword design is not a game, it has some similar typesetting problems, and $\LaTeX$ -using crossword makers will enjoy two sophisticated packages to help them.

8.1 Chess

Typesetting chess using $\TeX$ started in 1988 when Wolfgang Appelt developed a number of macros for this purpose (Appelt, 1988). This early attempt was done without a font containing chess figures, so Appelt denoted them with symbols like Ω . A little later Zalman Rubinstein (1989) produced a first simple chess font with METAFONT. Nearly four years later Piet Tutelaers, a Dutch chess enthusiast, designed a rather classical looking chess font and developed an accompanying $\LaTeX$ package (Tutelaers, 1992) that turned $\LaTeX$ into a mature typesetting system for chess journals.

Table 8.1: Coding for chess pieces in the chess fonts.

P		O		p		o		O	
N		M		n		m		Z	
B		A		b		a			
R		S		r		s			
Q		L		q		l			
K		J		k		j			

8.1.1 A METAFONT chess font

The chess font by Piet Tutelaers has 26 characters: for each chess piece (Pawn, Knight, Bishop, Rook, Queen, King) four characters represent it as Black and White on both squares (light and dark). In addition, there are two characters for the empty light and empty dark square.

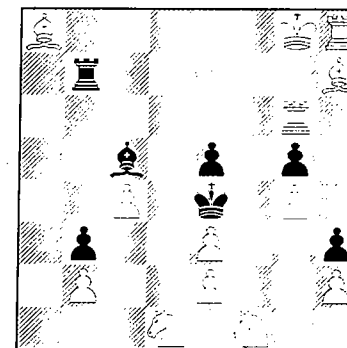
Borders for the game board are not part of the font but are added by chess diagram macros (discussed in the following section) as horizontal and vertical rules. Table 8.1 shows all characters present in the font `chess20` (the size of a square in that font is 20 points; the distribution also contains 10- and 30-point versions, `chess10` and `chess30`, and if desired other sizes could be generated with ease). An additional font, `chessf10`, contains only the chess figurines (King, Queen, Rook, Bishop, Knight) and is used to note chess moves in the short algebraic notation, e.g., `l25. N5*g3l` is typeset as `25. ♘5×g3l`.

8.1.2 chess — a package for typesetting chess

Piet Tutelaers (1992) developed the chess package to ease the typesetting of chess diagrams and annotated games, basing the code in parts on ideas published by Wolfgang Appelt four years earlier. The package offers commands that describe moves that are then typeset in an algebraic notation. Internally it maintains information about the current positions of all pieces; at any time it is possible with a single command to produce a display of the current board.

The above features are most useful if a whole game is to be annotated; when posing chess questions, automatic move updates are usually not required. For the latter situation, however, the chess package offers the command `\board`, which takes eight arguments denoting the eight rows of a chess board. Within each argument a simple convention is used to produce the graphic: empty light squares are represented by a space, empty dark squares

8-1-1



```
\usepackage {chess}
\board{B* * *KR}
{ * r * * B}
{ * * * R *}
{ * b p p }
{ * P * k * P *}
{ * p * P * p}
{ P * P * P}
{ * * N * N *}
\[ \showboard \]
```

Figure 8.1: Chess example — White to move without mating Black.

by a *, and figures by their abbreviations King, Queen, Rook, Bishop, kNight, and Pawn (upper case for white, lower case for black) independently of whether or not they are placed on a dark square. An example of this command is shown in Fig. 8.1. The solution to this problem is on p. 292.

Annotating games

A game is started either with a `\newgame` command or with a position environment; the latter allows you to start with a board position other than the initial one. For instance, the chess position after the 25th move in the Fisher–Tal game (see Fig. 8.2 on p. 281) is defined with:

```
\begin{position}
\White(Kh1,Qe6,Re1,a2,b2,c2,g2,h2)
\Black(Kf8,Qb8,Rd7,Rg8,Be7,a6,b4,b7)
\Whitetrue \setcounter{movecount}{25}
\end{position}
```

`\Whitetrue` says that White is to move next, while `\Whitefalse` says that the next move is Black's. In addition, we set the move counter to 25 thereby signaling that the next move is the 26th.

Positions defined by the position environment or the `\newgame` command are automatically updated with the commands `\ply` and `\move`. If White's move is not followed by an analysis, the `\move` command should be used, otherwise the move can be broken down into two plies (half moves) with `\ply`.

```
\move WhiteMove □ BlackMove
\ply Move
```

The arguments identify the origin and target squares of the moving piece (or the King in case of castling). (European) algebraic notation is used for identifying squares, i.e., columns are represented by a *letter* in the range a...h and rows by a *digit* in the range 1...8. Thus, for example, a move of the white knight from its initial position g1 to f3 is denoted as g1f3 without any spaces.

A Pawn arriving at its final destination is promoted to a Queen, Rook, Bishop or knight by appending the abbreviation Q, R, B, or N. If the promotion piece is omitted a default Queen is taken. Moves can also be commented with symbols like “!” for good moves, “??” for exceptionally bad moves, and so on. In addition to automatic promotion, the macros also handle en passant capture by pawns.

This short notation is translated internally into long algebraic notation, so that, for instance, specifying the move g1f3 would result in ♖g1-f3 in the case that square f3 is empty or ♖g1×f3 in case of a capture, while the move e1c1 (castling) gives 0-0-0. Examples are shown below; parts of a fully annotated game between Fisher and Tal during the Candidates' Tournament of 1959 are given in Fig. 8.2. The annotation is by Fisher (see Fig. 8.3 on p. 283 for a short algebraic representation of the complete game).

1.	♖g1-f3	d7-d6	\usepackage {chess}
2.	d2-d4	♙c8-g4	\newgame
3.	c2-c4	♖g8-f6	\move g1f3 d7d6
4.	♖b1-c3		\move d2d4 c8g4
5.	...	♙g4×f3	\move c2c4 g8f6
6.	g2×f3	♖b8-d7	\ply b1c3
7.	e2-e3	e7-e5	\ply g4f3
8.	d4×e5	d6×e5	\move g2f3 b8d7
9.	♙d1-c2	c7-c6	\move e2e3 e7e5
10.	♙c1-d2	♙d8-c7	\move d4e5 d6e5
	0-0-0	0-0-0	\move d1c2 c7c6
			\move c1d2 d8c7
			\move e1c1 e8c8

Tournament style

To save space, tournament bulletins prefer short algebraic notation in which the *from* square of a piece is omitted whenever possible. If a move is ambiguous because several pieces can reach the same destination, the letter of the column or the number of the row of the originating square is added. Short and long algebraic notation is also used in analysis mode when annotating moves from the main line, i.e., the actual game as played or a variation. To ease typesetting of moves in short algebraic form, the compact input notation |...| is available. In cases where the | character cannot be used, it can alternatively be invoked with

```
\begin{position}
\White(Kh1,Qe6,Re1,a2,b2,c2,g2,h2)
\Black(Kf8,Qb8,Rd7,Rg8,Be7,a6,b4,h7)
\Whitetrue \setcounter{movecount}{25}
\end{position}
\begin{center}
\centerline{Diagram: Fisher--Tal
after 25.~\ldots, {\Fig K}f8!}
\[\showboard\]
\end{center}
```

```
\ply e6d7
Not |26. Rf1+, Kg7; 27. Rf7+, Kh8;
and if 28. Q×d7, Rd8; 29. Qg4, Qe5|
wins.
```

```
\ply b8d6
\move d7b7 g8g6
Within a handful of moves the game
has changed its complexion. Now it
is White who must fight for a draw!
```

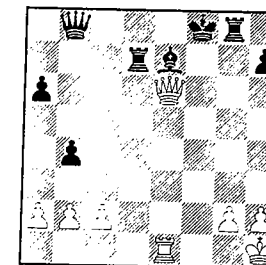
```
\ply c2c3!
Black's extra piece means less with
each pawn that's exchanged.
```

```
\ply a6a5
On |28.: b×c3; 29. Qc8+, Bd8;
30. Q×c3|=.
```

```
\ply b7c8+
On the wrong track. Right is
|29. c×b4!, Q×b4 (if 29.: a×b4;
30. a3!, b×a3; 31. b×a3, Q×a3 draws);
30. Qf3+, Kg7; 31. Qe2| draws, since
Black can't possibly build up a
winning K-side attack and his own
king is too exposed.
```

```
\ply f8g7
\move c8c4 e7d8
\move c3b4 a5b4
On |31.: Q×b4; 32. Qe2| White should
draw with best play.
\[\showboard\]
```

Diagram: Fisher--Tal after 25. ..., ♙f8!



26. ♙e6×d7
Not 26. ♙f1+, ♙g7; 27. ♙f7+, ♙h8; and if 28. ♙×d7, ♙d8;
29. ♙g4, ♙e5 wins.

26. ... ♙b8-d6
27. ♙d7-b7 ♙g8-g6
Within a handful of moves the game has changed its complexion. Now it is White who must fight for a draw!

28. c2-c3!
Black's extra piece means less with each pawn that's exchanged.

28. ... a6-a5
On 28. ..., b×c3; 29. ♙c8+, ♙d8; 30. ♙×c3=.

29. ♙b7-c8+
On the wrong track. Right is 29. c×b4!, ♙×b4 (if 29. ..., a×b4;
30. a3!, b×a3; 31. b×a3, ♙×a3 draws); 30. ♙f3+, ♙g7; 31.
♙e2 draws, since Black can't possibly build up a winning K-side
attack and his own king is too exposed.

29. ... ♙f8-g7
30. ♙c8-c4 ♙e7-d8
31. c3×b4 a5×b4
On 31. ..., ♙×b4; 32. ♙e2 White should draw with best play.

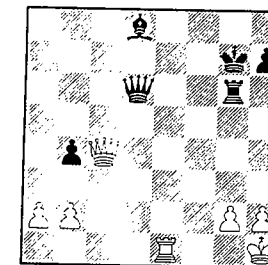


Figure 8.2: Example of L^AT_EX source and output of a chess game.

the environment chess. Below are a few examples of long and short algebraic notation in analysis mode:

input	output
21. Nf3-e5??, 0-0!	21. ♖f3-e5??, 0-0!
21.: Ke8*f8	21...., ♔e8×f8
21. Nfe5, K*f8	21. ♖fe5, ♔×f8
21.: f8Q+	21...., f8♙+

In analysis mode certain characters have a special meaning: “*” becomes “×” (a capture), “_” is mapped to “-” and “:” is an abbreviation for “...,” (needed if the analysis starts with a Black move).

The use of the | character in this fashion has one undesirable side-effect: it can no longer be used for other purposes in the same document. Perhaps the most important of these usages is its rôle as a vertical rule in the syntax of the environment. To resolve this problem, the package provides the environment `nochess`, within which the | character has its usual meaning; if necessary, the chess environment can be used instead.

In tournament style, the mainline is usually typeset in boldface and the variations in roman. Certain common abbreviations indicating the status or the quality of certain moves can be accessed using the following commands:

TeX	meaning
±	\wbetter White stands slightly better
≡	\bbetter Black stands slightly better
±	\wupperhand White has the upper hand
≡	\bupperhand Black has the upper hand
+-	\wdecisive White has a decisive advantage
-+	\bdecisive Black has a decisive advantage
→	\withattack with attack
△	\betteris better is
o-o	\seppawns separated pawns

To illustrate tournament style, Fig. 8.3 shows the complete game of Bobby Fisher against Mikhail Tal (Belgrade, 1959). To ease the coding the following environments, which are not part of the chess package, are used:

```
\newenvironment{mainline}[2]{\bfseries
\newcommand{\result}{#1}\newcommand{\commentator}{#2}\begin{chess}
{\end{chess}}\finito{\result}{\commentator}}
\newenvironment{variation}{\mdseries\ignorespaces}{\ignorespaces}
\newcommand{\finito}[2]{\bfseries\hfill#1\hfill#2\par}}
```

```
1. e4 c5 2. ♘f3 d6 3. d4 cd4 4. ♘d4 ♘f6 27. ♖a6 ♙b7! 28. ♙b7 ♘b7 29. ♖af6
5. ♘c3 a6 6. ♘c4 e6 7. ♘b3 [7. 0-0 ♘e7 ♖g7=; 22... 6? 23. ♙f7 ♘d7 24. ♖d1!
8. ♘b3 ♙c7 9. f4 b5 10. f5 b4 11. fe6! (11. ♖d6 25. ♖d6 ♘d6 26. ♖f6!+-] ♖d7 23.
♘ce2 e5 12. ♘f3 ♘b7≡) bc3 12. ef7 ♘f8 13. ♖ae1 [23. ♖ad1 ♘d6 24. ♖f7 (24. ♖f6
♘g5 ♘g4!≡) b5! [7... ♘e7? 8. f4 0-0 9. ♖g6 25. ♖dd6 ♙d6! ♙c7; 23. ♖f7 ♙d6
♙f3 ♙c7 10. f5! (10. 0-0? b5 11. f5 b4! 12. ♖e7 [23... ♘d8? 24. ♖f7! ♘e7 25.
♘a4 e5 13. ♘e2 ♘b7≡) e5 11. ♘de2 b5 12. ♖fe7 ♖e7 26. ♖d1+-] 24. ♖f7 ♘f7 25.
a3 ♘b7 13. g4±→] 8. f4? b4! 9. ♘a4 ♘e4 ♖e6 ♘f8! [25... ♘g7? 26. ♙d7+-] 26.
10. 0-0 g6? [→10... ♘b7] 11. f5! gf5 ♖d7 [26. ♖f1? ♘g7 27. ♖f7 ♘h8 (28.
[11... ef5? 12. ♘d5 ♖a7 13. ♘f5! gf5 14. ♖d7 ♖d8 29. ♙g4 ♙e5+-] ] ♙d6 27.
♙d4] 12. ♘f5! ♖g8 [→12... d5 13. ♘h6 ♙b7 ♖g6 28. c3! a5 [28... bc3 29. ♙c8
♘h6 14. ♘h6; 12... ef5? 13. ♙d5 ♖a7 ♘d8 30. ♙c3=] 29. ♙c8 [→29. cb4! ♙b4
14. ♙d4] 13. ♘d5! ♖a7 [13... ed5 14. (29... ab4 30. a3! ba3 31. ba3 ♙a3=) 30.
♙d5 ♘f5 15. ♖f5 ♖a7 16. ♙e4 ♖e7 17. ♙f3 ♘g7 31. ♙e2=] ♘g7 30. ♙c4 ♘d8
♙b4 ♖e2 18. ♘g5! ♖g5 19. ♖g5 ♙g5 20. 31. cb4 ab4 [31... ♙b4 32. ♙e2=] 32. g3?
♖b8+-PANOV] 14. ♘e4? [→14. ♘e3! [32. ♙e4 (32... ♘c7 33. ♙e7 ♘g8 34.
♘c5 15. ♙h5! ♖g6 (15... ♘a4 16 ♘a7 ♙e8 ♙f8 35. ♙e4) ] ♙c6 33. ♖e4 ♙c4 34.
ed5 17 ♖ae1] 16. ♖ae1! KEVITZ] ef5 15. ♖c4 ♖b6! [34... ♘e7? 35. a3! ] 35. ♘g2
♘f5 ♖e7! 16. ♘c8 ♙c8 17. ♘f4? [→17. ♘f6 36. ♘f3 ♘e5 37. ♘e3 ♘g5 38. ♘e2
c3! ♙c6 18. ♖f2; 17. ♙d6? ♖g2 18. ♘g2 ♘d5 39. ♘d3 ♘f6 40. ♖c2? ♘e5 41. ♖e2
♖e2 19. ♘f3 ♘d6 20. ♘e2 ♙c2+-] ♖g ♖f6 42. ♖c2 ♖f3 43. ♘e2 ♖f7 44. ♘d3
♙c6! 18. ♙f3 ♙a4 [18... ♙f3 19. ♖f3 ♘d4! 45. a3 [45. b3? ♖f3 46. ♘e2 ♖f2 47.
♖e2 20. ♖f2 ♖f2 21. ♘f2±o-o] 19. ♘d6 ♙d3 ♖c2 48. ♘c2 ♘e5+-] b3 46. ♖c8
♙c6! 20. ♘b8 ♙b6 [20... ♙f3? 21. ♖f3 [46. ♖e2 ♖f3 47. ♘d2 ♘b2; 46. ♖d2 ♖f3
♘g7 22. c3+-] 21. ♘h1 ♙b8 22. ♙c6 47. ♘e2 ♖f2] ♘b2 47. ♖d8 ♘c6 48. ♖b8
[22. ♖ae1 ♘d8! 23. ♖d1 ♘c7! (23... ♖f3 49. ♘c4 ♖c3 50. ♘b4 ♘c7 51. ♖b5
♘c8? 24. ♙c6+-] 24. ♙f4 (24. ♖d4 ♘a1 52. a4 b2! [53. ♘c3 b1♙! ] 0:1
♙b7!) ♘b7 25. ♖d6 ♙c7 26. ♙b4 ♘c8 [Fisher]
```

```
\begin{mainline}[0:~1]{Fisher}
1. e4 c5 2. Nf3 d6 3. d4 cd4 4. Nd4 Nf6 5. Nc3 a6 ...
13. Bd5! Ra7
\begin{variation}
13: ed5 14. Qd5 Bf5 15. Rf5 Ra7 16. Qe4 Re7 17. Qb4 Re2
18. Bg5! Rg5 19. Rg5 Qg5 20. Rb8\wdecisive
\begin{nochess} PANOV \end{nochess}
\end{variation}
14. Be4? ...
\end{mainline}
```

Figure 8.3: Chess tournament notation.

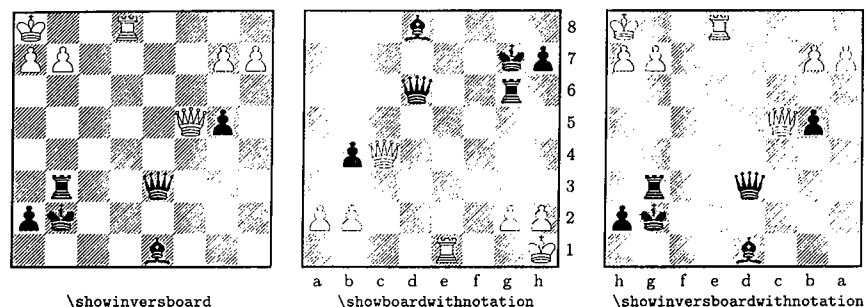


Figure 8.4: Additional chessboard visualization commands.

8.1.3 Extensions to the chess package

Frank Hassel developed an interesting add-on package to facilitate managing chess games played by correspondence (Hassel, 1993). His package `bdfchess` also provides a few useful extensions like those shown in Fig. 8.4.

It is customary in correspondence chess to play two games simultaneously, each participant playing one game as White, i.e., the starting player. This is supported by the package through the two environments `gameone` and `gametwo` that record the current state in either game. Within these environments you can use the commands `\postmove` and `\postply` that are similar to `\move` and `\ply` but also maintain information about arrival dates of postcards and the time each player needed to think about the move.

```
\postmove WhiteArrival □ Days □ Move □ BlackMove □ Arrival □ Days
\postply Move □ Arrival □ Days
```

The *Arrival* dates are given in the notation `dd.mm.yyyy`, the time needed for the move in full days. The `gameone` and `gametwo` environments generate a listing of the moves in long algebraic notation. The moves or half moves can be annotated with comments and one can generate at any time a board display by using the `\showboard` command or one of its variants.

```
\begin{gameone}
\postmove 11.09.1996 0 d2d4 g8f6 12.09.1996 2
\postmove 15.09.1996 1 c2c4 g7g6 17.09.1996 2
\postply 21.09.1996 1 b1c3
\end{gameone}
```

8.1.4 Interfacing chess databases

Date			White	Black	Date		
Arrival	Leaving	Σ			Arrival	Leaving	Σ
11.9.96	11.9.96	0	d2-d4	1.	12.9.96	14.9.96	2
15.9.96	16.9.96	1	c2-c4	2.	17.9.96	19.9.96	4
21.9.96	22.9.96	2	g7-g6	3.			

The package offers several commands to produce a postcard from the information stored with the `gameone` and `gametwo` environments; the most important of these are shown below.

```
\sender{address}
\receiver{address}
\postcard[frame] ( h-offset, v-offset )
\postcardaddress[split] ( h-offset, v-offset )
```

The *address* of the sender and receiver can be specified in free style using $\text{\LaTeX}$ commands and environments as necessary. The `\postcard` command produces the back of the postcard showing the status of the games; its optional argument *frame*, if present, must contain the letter `f`, in which case a frame is put around the postcard. If used, `\postcardaddress` produces the front of the postcard containing the addresses specified by `\sender` and `\receiver`. The optional argument *split* can contain either or both of the letters `f` (for *frame*) and `t` (which adds a vertical line on the front and typesets the word “postcard”). The *offset* arguments of both commands are dimensions that move the postcard on the paper if necessary. A full example, showing input and results (except for the listings produced by the `gameone` environments), is given in Figs. 8.5 and 8.6.

Frank Hassel added the font `chess15` to generate diagrams better suited to postcard size. To use it with either `chess` or `bdfchess`, one must add the following line after loading the package:

```
\font\Chess=chess15
```

He also developed a package `chesskey` with a single command `\key` whose only parameter is an identifier between A00 to E99, corresponding to the key of the opening in the International Chess-Openings Classification Scheme.

8.1.4 Interfacing chess databases

There are several chess-related databases containing complete games, such as `ChessBase` and `NicBase`.¹ To be able to typeset games stored in such databases Jürgen Lamers developed a set of Pascal programs based on earlier work by Carsten Hansen. These programs transform chess database formats into $\text{\LaTeX}$ input files that can be used with the chess package.

¹ Information about these formats can be found in the CTAN archives, `support/chesstools`.


```
\begin{gameone}
\postmove 03.09.1996 0 e2e4 d7d6 11.09.1996 0
\postmove 12.09.1996 2 d2d4 g8f6 15.09.1996 1
\postmove 17.09.1996 2 b1c3 g7g6 21.09.1996 1
\end{gameone}
\begin{gametwo}
\postmove 11.09.1996 0 d2d4 g8f6 12.09.1996 2
\postmove 15.09.1996 1 c2c4 g7g6 17.09.1996 2
\postply 21.09.1996 1 b1c3
\end{gametwo}
\sender{\small sender:\\
\large\textbf{TUG Office}    \\
\TeX{} User Group          \\
1850 Union Street, \#1637   \\
San Francisco CA 94123     \\
USA}
\receiver{\setlength{\parskip}{5mm} \LARGE \bfseries
DANTE                      \vskip 2pt \hrule
Postfach 10\,18\,40        \vskip 2pt \hrule
69008 Heidelberg           \vskip 2pt \hrule
Germany}
\cardmessage{The office will be closed from\ 30.09.96 until 14.10.96}
\postcard[f](0mm,-20mm)

\vspace{10mm}

\postcardaddress[tf](-1mm,0mm)
```

Figure 8.5: Chess-Postcard input for bdfchess.

For instance, the command to extract a game from a ChessBase-formatted database is:

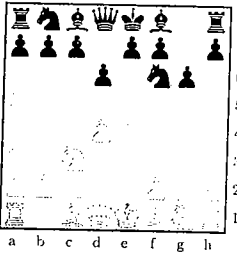
```
cb2tex fritz2 fritz2.tex
```

This generates “long-form” output in which every move is written as `\move e4e5 d6*e5`. Additionally, every 15 moves a `\showboard` command is written to the output file.

Alternatively, one can use the “short” form in the output file by specifying something like:

```
nic2tex fischer3 fischer.tex -s
```

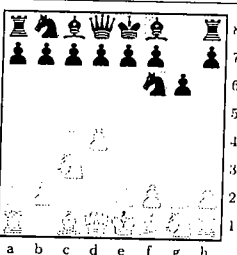
In this case a game stored in a NicBase-formatted database is written in short form into the $\LaTeX$ file, e.g., as `12. g2g4, c6d4`. These programs, together with some example databases, are available as binaries and source files in the CTAN archives, support/chesstools.



Game 1

No.	Your move	No.	My move
3	$\mathcal{Q}$ b1-c3	3	g7-g6

Received on 21.9.96
Replied on 22.9.96
My time taken 1 day(s)
My total time 2 days



Game 2

No.	Your move	No.	My move
2	g7-g6	3	$\mathcal{Q}$ b1-c3

Your postmark date 19.9.96
Your time taken 2 day(s)
Your total time 4 days

Yours sincerely

sender:
TUG Office
TeX User Group
1850 Union Street, #1637
San Francisco CA 94123
USA

postcard

The office will be closed from
30.09.96 until 14.10.96



DANTE
Postfach 10 18 40
69008 Heidelberg
Germany

8-1-3

Figure 8.6: Chess postcards generated by bdfchess.

8.1.5 Using Adobe's cheq font

Cheq is a Type1 PostScript font available free from Adobe. It was originally interfaced to L^AT_EX by Adrian Clark via his cheq package, which set up a number of commands with names like \CheqBKing to access the individual figures but offered no support for drawing a board. With the help of this package and the chess macros developed by Wolfgang Apelt (1988), Sebastian Rahtz wrote an example file chessboard.tex demonstrating one possible way of using this font.

In this section we take a different path: adjusting the font to make it usable with the chess or bdfchess package. This serves both as an example for using the PostScript font tools discussed in Chapter 10 and as a recipe for customizing and integrating other (commercial) chess fonts so that they can be used with these powerful packages.

The Cheq font is distributed as a pfa file with corresponding afm and pl files.² To use it with L^AT_EX we first must generate a tfm file from the pl file:

```
$ pltotf -verbose cheq.pl cheq.tfm
This is PLtoTF, Version 3.4 (C version 6.1)
'055 '057 '102 '104 '113 '114 '116 '120
'121 '122 '123 '124 '134 '137 '142 '144
'153 '154 '156 '160 '161 '162 '163 '164
'170 '240 '247 '250 '266 '270 '271 '272
'265 '302 '316 '317 '345 '346 '352 '353
'360 '361 '365 '366 '373.
```

This is enough to use the font inside L^AT_EX, but we must also inform the PostScript printer driver that this font should be downloaded to the printer instead of being available as a pk file. In case of dvips this can be done by writing a short mapping file cheq.map containing the following statement:

```
cheq Cheq <cheq.pfa
```

Note that in the map file the first name is the font as seen by T_EX, i.e., the name of the tfm file, the second name is the name used inside the PostScript font source (hence the upper-case Cheq, since that is how the font is declared in the file cheq.pfa on the FontName command; see Chapter 10 for more information.)

To inform dvips about this mapping we can either update its master configuration file config.ps or write another one-line configuration file (config.cheq) containing the following line:

```
p +cheq.map
```

If we choose the latter solution we need to use dvips in the following way with any document

²On CTAN it is distributed with file names all in lower case, but you may also find distributions with file names like Cheq.pfa.

accessing this font:

```
$ dvips -Pcheq chessboard # run dvips with the config file
This is dvipsk 5.58e Copyright 1986, 1994 Radical Eye Software
' TeX output 1995.08.25:2245' -> chessboard.ps
<tex.pro><cheq.ps><texps.pro>. [1]
```

To use the font with the chess package we might try something like \font \Chess=cheq at 10pt after loading the package, but we will soon find out this doesn't work because unfortunately the characters in the cheq font and chess10 are in different positions.³ To get the effect we want, we have to re-encode the PostScript font, i.e., move the right symbols into the right places. This is not as complicated as might appear; here we just describe the steps necessary and show the results. More details are given in Chapter 10.

First we produce font tables for both fonts so that we know in which slot each symbol lives. This can be easily done with the standard L^AT_EX file nfssfont.tex.

```
$ latex nfssfont
This is TeX, Version 3.14159 (C version 5.1)
(/cdrom/texmf/tex/latex/base/nfssfont.tex
LaTeX2e <1995/12/01> patch level 2
...
*****
* NFSS font test program version <v2.0d>
*
* Follow the instructions
*****

Name of the font to test = chess10
Now type a test command (\help for help):
*\table

*\init

Name of the font to test = cheq at 10pt
Now type a test command (\help for help):
*\table

*\bye
[1]
Output written on nfssfont.dvi (1 page, 9396 bytes).
Transcript written on nfssfont.log.
```

The result of this L^AT_EX run, shown in Fig. 8.7 gives us information about the actual position of each symbol in the font.

³Because the font \Chess is already declared by the package, we have to use the low-level command \font and not \newfont, as L^AT_EX does not have a \renewfont variant.

Test of chess10 on January 14, 1997 at 2056

	'0	'1	'2	'3	'4	'5	'6	'7	
'06x									"3x
'07x									
'10x									"4x
'11x									
'12x									"5x
'13x									
'14x		♙	♙						"6x
'15x			♚	♚	♚	♚	♚	♚	
'16x	♙	♙	♙	♙					"7x
'17x									
	"8	"9	"A	"B	"C	"D	"E	"F	

Test of cheq at 10pt on January 14, 1997 at 2056

	'0	'1	'2	'3	'4	'5	'6	'7	
'04x									"2x
'05x									
'10x			♙		♙				"4x
'11x			♙	♙	♙		♙		
'12x	♙	♙	♙	♙	♙				"5x
'13x			♙		♙				
'14x			♙		♙				"6x
'15x			♙	♙	♙	♙	♙		
'16x	♙	♙	♙	♙	♙				"7x
'17x	♙								
'24x	♙							♙	"Ax
'25x	♙								
'26x						♙	♙		"Bx
'27x	♙	♙	♙						
'30x			♙						"Cx
'31x							♙	♙	
'34x						♙	♙		"Ex
'35x			♙	♙					
'36x	♙	♙				♙	♙		"Fx
'37x				♙					
	"8	"9	"A	"B	"C	"D	"E	"F	

Figure 8.7: Font encodings for chess10 and cheq.

To re-encode the PostScript font we have to define a new encoding vector that matches the font layout of chess10 and lists the PostScript names of the symbols used in the font. We find those names in the afm file (or alternatively in the pfa file). We must make a list of exactly 256 names, putting dummy /.notdef markers where we do not want a symbol.

```

/ChessEncoding [
/.notdef /.notdef /.notdef /.notdef /.notdef /.notdef
/.notdef /.notdef
% many more such lines
...
% '100 - '107
/.notdef /WBishopOnBlack /WBishop /.notdef /.notdef /.notdef
/.notdef /.notdef
% '110 - '117
/.notdef /.notdef /WKingOnBlack /WKing /WQueenOnBlack
/WKnightOnBlack /WKnight /WPawnOnBlack
% '120 - '127
/WPawn /WQueen /WRook /WRookOnBlack /.notdef /.notdef /.notdef /.notdef
% '130 - '137
/.notdef /.notdef /BSquare /.notdef /.notdef /.notdef /.notdef
% '140 - '147
/.notdef /BBishopOnBlack /BBishop /.notdef /.notdef /.notdef
/.notdef
% and a few similar lines for the remaining Black pieces
...
% followed by a lot of
/.notdef /.notdef /.notdef /.notdef /.notdef /.notdef /.notdef
/.notdef ...
% and finished with
] def

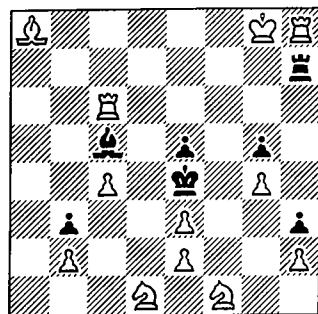
```

Having done this, the only task remaining is producing a re-encoded font (to L^AT_EX with a font having a different name) from the above encoding vector and the original font. This font could, for example, be called xcheq. To get a tfm file for use with L^AT_EX, we can use the program afm2tfm as follows (see Section 10.5.6 on p. 390 for the various other ways to perform this step), assuming the encoding vector was saved in the file xchess.enc:

```
$ afm2tfm cheq -T xchess.enc xcheq
```

This produces the file xcheq.tfm, so that from now on we can use the font in L^AT_EX. As before, we must ensure that the PostScript driver will download the font to the printer (this time re-encoded); with dvips this can be achieved by updating the map file, adding the following additional line:

```
xcheq Cheq " ChessEncoding ReEncodeFont " <xchess.enc <cheq.pfa
```



```
\usepackage {chess}
\font\Chess=xcheq at 18pt
\setlength{\fontdimen2\Chess}{0pt}
\board{B* * *KR}
{ * * * *r}
{ *R* * *}
{ * b p p }
{ *P*k*P*}
{ *p* P *p}
{ P *P* P}
{ * *N*N* }
\[\showboard \]
```

8-1-4

Figure 8.8: Solution — White to move without mating Black.

Done! We can now present you with the solution to the chess question posed a couple of pages ago; the answer is given in Fig. 8.8, which also shows the use of the Adobe chess font with the chess package.

8.2 Xiangqi — Chinese chess

The Chinese chess game, xiangqi, has the same roots as the European chess game but shows extensive differences. For example, the two armies are separated by a river that some pieces cannot cross. Also interesting is the fact that the two armies have different pieces: the white elephants are replaced by ministers on the opposite side, and while the white general is accompanied by advisors the black commander has officers at his side. In European versions of this game in which the Chinese characters are replaced by pictograms, these differences are often ignored.

Jacques Richer has written METAFONT code for the Chinese chess pieces that produces stones with traditional Chinese characters. As shown in Table 8.2 this font, `cchess46`, has a set of black and white stones (upper-case letters denote black, lower-case letters denote white). The letters chosen correspond to the common names for comparable pieces of European chess, e.g., the general is accessible as the letter “k” for king. In addition, the font contains a third set of stones: an alternate set for black in which the characters also have a white background. This third set is useful in producing colored game diagrams—traditionally all stones are white with green and red characters.

The distribution originally did not offer a package with environments and commands to typeset games. But it did contain a L^AT_EX document called `cchessboard.tex` to produce a board with the stones in their initial positions, as shown in Fig. 8.9 (except that we used the alternate set of stones for the black player).

Table 8.2: Coding for xiangqi pieces in the `cchess46` font.

	white		black		alternate black	common names
k	帥	K	將	S	將	King, General;
g	仕	G	士	T	士	Guard, Assistant;
b	相	B	象	U	象	Bishop, Elephant;
n	馬	N	馬	V	馬	Knight, Horse;
r	車	R	車	W	車	Rook, Chariot, Car;
c	炮	C	炮	X	炮	Cannon, Gun, Gunner;
p	兵	P	卒	Y	卒	Pawn, Foot-soldier.

Rearranging the code in this document, Frank Mittelbach produced a small package called `cchess` that displays arbitrary positions with the help of a position environment and the commands `\piece` and `\textpiece`.

`\textpiece{<name>}`

Within running text the command `\textpiece` can be used. Possible values for `name` are the characters shown in Table 8.2. With this and, for example, the tabbing environment, one can easily document and annotate games.

The following listing, a mate situation after four moves, gives an example of the use of this command. The board situation after these four moves is shown in Fig. 8.10.

```
\usepackage {cchess}
\newcommand\x{\times} % a shortcut to denote capture
\begin{tabbing}
1. \> \textpiece{c}h3--e3 \quad \> \textpiece{N}b0--a8 \\
2. \> \textpiece{c}e3xe7 \> \textpiece{R}a0--a9 \\
3. \> \textpiece{c}b3-b5 \> \textpiece{N}h0--g8 \\
4. \> \textpiece{c}b5--e5 mates!
\end{tabbing}
```

8-2-2

The position environment draws a complete board. Within its body the `\piece` command is used to place the individual pieces.

`\piece{<file>}{<rank>}{<name>}`

The `\piece` command places a piece `name` at a certain file and rank (the horizontal and

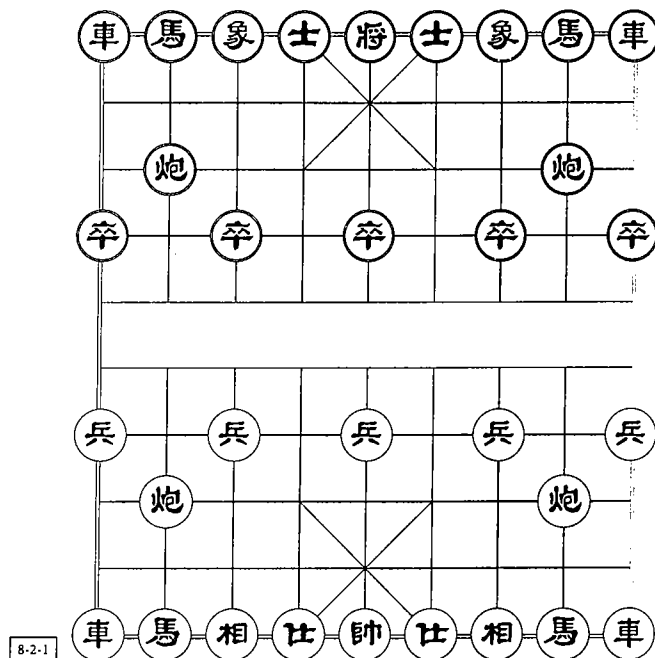
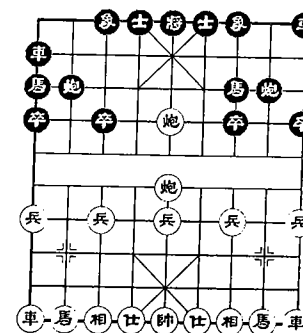


Figure 8.9: Initial setup of Chinese chess game (xiangqi).

vertical position). The *file* argument can take a letter in the range from a to i (left to right) while the *rank* is specified by numbers between 1 and 10 (bottom to top). An example is shown in Fig. 8.10.

In addition, the package offers the commands `\largeboard`, `\normalboard`, and `\smallboard` that specify the size of the board and pieces. Figure 8.9 shows a normal-sized board while Fig. 8.10 uses `\smallboard`.

In the current implementation the `cchess` package uses `\special` commands for PostScript drivers to place the pieces on the board without reconstructing the picture. But this is not the only way to program this, as we see in Section 8.3 below. Perhaps the future will bring us a new and extended implementation of this package that includes concepts used in the chess package to allow documenting tournaments, annotated games and analysis of this interesting Eastern game.



8-2-3

Figure 8.10: A mate situation after four moves.

```
\usepackage {cchess}
\smallboard
\begin{position}
\piece{a}{1}{r}\piece{i}{1}{r}
\piece{b}{1}{n}\piece{h}{1}{n}
\piece{c}{1}{b}\piece{g}{1}{b}
\piece{d}{1}{g}\piece{f}{1}{g}
\piece{e}{5}{c}\piece{e}{7}{c}
\piece{a}{4}{p}\piece{c}{4}{p}
\piece{e}{4}{p}\piece{g}{4}{p}
\piece{i}{4}{p}\piece{e}{1}{k}
\piece{a}{9}{R}\piece{i}{10}{R}
\piece{a}{8}{N}\piece{g}{8}{N}
\piece{c}{10}{B}\piece{g}{10}{B}
\piece{d}{10}{G}\piece{f}{10}{G}
\piece{b}{8}{C}\piece{h}{8}{C}
\piece{a}{7}{P}\piece{c}{7}{P}
\piece{g}{7}{P}
\piece{i}{7}{P}\piece{e}{10}{K}
\end{position}
```

8.3 Go

Go, perhaps the most popular Asian game, is played by two players using black and white stones on the intersections of a 19×19 grid. Stones, after being placed, are never moved unless they are captured, so in documenting the course of a game one usually shows a board with numbered stones. An algebraic notation such as we saw for chess is normally not used.

Hanna Kołodziejewska has developed a number of fonts and an accompanying package `go` to typeset Go diagrams and document full or partial games. The fonts available in the three sizes 10pt, 15pt, and 20pt contain numbered stones in the range from ❶ to ㊯ as well as unnumbered stones, various symbols, and intersection and border lines to produce a display of the board. In addition, a font with a single character shows the traditional symbol for the Go game, as in Fig. 8.11.

It is in principle possible for more than 250 stones to be used in a game, in which case it could not be represented in a single diagram with the currently available fonts. However, since for readability it is usually best to put no more than 100 moves on a single diagram (and then restart the numbering), this theoretical limitation is not very important in practice.

Like the chess package, the `go` package has commands for board initialization, placement of stones (which in this case are equivalent to moves), and commands to display the board or parts of the board. The actual syntax, however, is quite different.



```
\newfont\gosign{gosign50}
\begin{center}
\fbbox{\gosign\symbol{0}}
\end{center}
```

8-3-1

Figure 8.11: The Go symbol.

To place stones onto the board using the commands of the go package, each line intersection is identified by a row label (a lower-case letter in the set a, b, c, d, e, f, g, h, i, k, l, m, n, o, p, q, r, s, t, note the absence of j!) and a column number (in the range 1 to 19).

```
\inifulldiagram
\inidiagram row1-row2:col1-col2
```

The first command `\inifulldiagram` clears the complete board, while the command `\inidiagram` clears the squares delimited by the rows identified by the letters *row₁* and *row₂* and the columns identified by the numbers *col₁* and *col₂*. Note that a blank or a new line *must* follow the last digit.

When the go package is loaded, the board is automatically initialized with no stones on it. Therefore these commands are necessary only if more than one diagram is to be produced.

```
\gofontsize{size}
```

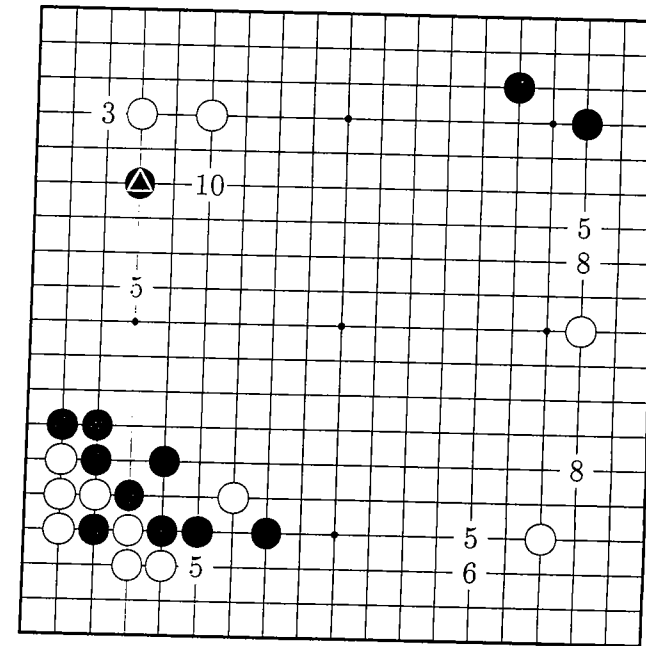
The `\gofontsize` command is available to specify the size of the stones used; the argument *size* can be 10, 15, or 20 and denotes the font size in points. Figure 8.12 shows a 15-point diagram while Fig. 8.13 shows a partial 20-point diagram.

The command `\pos` positions stones on the board. It has the following general syntax:

```
\pos{row}{col}=\player{display}
```

The argument *row* is the letter identifying the row and *col* is the number identifying the column. The command `\player` is one of the commands `\black`, `\white`, and `\letter` (or `\symbol4`). The argument *display* specifies the number to typeset inside the circle—a dot indicates that just the circle is to be typeset without embellishments. It is common to mark especially important stones with an embedded triangle or square. This can be achieved by using `\triangle` or `\square` in the *display* argument. If the `\letter` command is used, *display* denotes the letter or digit to place at the intersection, a method commonly used when discussing Go problems. An example showing all these variations of the `\pos` command is given in Fig. 8.13.

⁴Note that the package unfortunately overwrites L^AT_EX's standard `\symbol` command. Suggestions are given



8-3-2

Best moves for Black are near

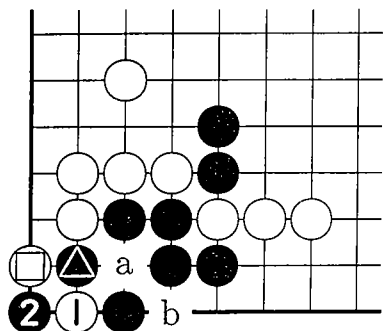
Figure 8.12: A Go game at an early stage.

To produce a display of the current board situation, the following two commands are available:

```
\showfulldiagram
\showdiagram row1-row2:col1-col2
```

The syntax for `\showdiagram` is the same as for `\inidiagram` discussed above. Partial diagrams are often used when commenting on Go problems and showing their solutions with different moves. Thus the use of `\showdiagram` in Fig. 8.13 is typical.

at the end of this section to overcome any problems this may cause.



```
\usepackage {go}
\inifulldiagram \gofontsize{20}
\pos{p}{5}=\black{.} \pos{q}{2}=\white{.}
\pos{o}{3}=\white{.} \pos{q}{3}=\white{.}
\pos{q}{4}=\white{.} \pos{q}{5}=\black{.}
\pos{r}{2}=\white{.} \pos{r}{3}=\black{.}
\pos{r}{4}=\black{.} \pos{r}{5}=\white{.}
\pos{r}{6}=\white{.} \pos{r}{7}=\white{.}
\pos{s}{1}=\white{\square}
\pos{s}{2}=\black{\triangle}
\pos{s}{3}=\letter{a} \pos{s}{4}=\black{.}
\pos{s}{5}=\black{.} \pos{t}{1}=\black{2}
\pos{t}{2}=\white{1} \pos{t}{3}=\black{1}
\pos{t}{4}=\letter{b}
\begin{center}
\mbox{\showdiagram n-t:1-8 }
\end{center}
```

8-3-3

Figure 8.13: Two-stage ko (favorable to Black).

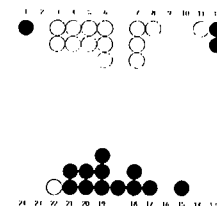
```
\textblack{display} \textwhite{display}
```

Stones can be typeset in running text with the commands `\textblack` and `\textwhite`, which have one argument, the number or special symbol (`\triangle` or `\square`) to appear on the stone. Thus `\textblack{5}` produces 5 in a commentary.

8.3.1 Possible problems

The go package was originally developed for use with plain TeX, and thus the coding can sometimes conflict with other applications when used in L^AT_EX. Here are the most likely problems and possible cures.

- As already mentioned, the L^AT_EX command `\symbol` is overwritten. Since you don't need the new definition (it is just an alternate name for `\letter`), you can restore the original definition if necessary after loading the package by specifying `\renewcommand{\symbol[1]{\char#1\relax}`.
- For similar reasons you cannot use the go package together with the letter class.
- The package internally redefines the plain TeX command `\empty`. This can have strange effects in apparently random places in your output, because the command is sometimes used by other packages.
- Do not use this package together with others in the same `\usepackage` declaration, since it may not work.



8-4-1

White to play 3-2

```
\usepackage {bg}
\begin{position}
\smallboard
\whitepoint{3}{2} \whitepoint{4}{2}
\whitepoint{5}{2} \whitepoint{6}{3}
\whitepoint{7}{3} \whitepoint{8}{1}
\whitepoint{11}{1} \whitepoint{22}{1}
\blackpoint{24}{1} \blackpoint{13}{2}
\blackpoint{10}{1} \blackpoint{8}{1}
\blackpoint{7}{2} \blackpoint{6}{3}
\blackpoint{5}{2} \blackpoint{4}{2}
\blackbar{1}
\shownumbers \middlecube{1} \showcube
\whiteonmove
\boardcaption{White to play 3--2}
\end{position}
```

Figure 8.14: A backgammon example.

8.4 Backgammon

Jörg Richter's package `bg` defines two L^AT_EX environments, `position`, and `game`, to display backgammon games. The `position` environment draws a single board and is thus convenient for discussing a problem, while with the `game` environment you can enter each move individually. In the latter case the board positions are stored internally, allowing the "current" status to be drawn at any time.

By convention, the homes of both players are on the left-hand side, with white's home at the top and black's at the bottom. Unlike the packages discussed so far, positions on the board are not denoted with absolute coordinates but are numbered as viewed by the party whose move is being placed, e.g., White's 24 corresponds to Black's 1, and so on. Moves are always performed from high to low numbers, and the cube is always on the right-hand side of the board.

```
\begin{position}... \end{position}
```

The `position` initializes an empty board into which stones are placed by the commands described below. This list also contains commands to customize some aspects of the board's layout. The board is printed when the `\end{position}` command is encountered. An example that uses various commands of the `position` environment is shown in Fig. 8.14.

```
\blackpoint{n}{p} \whitepoint{n}{p}
```

The two commands are used to place stones onto the board; *n* denotes the number of stones to place and *p* denotes the point where they are positioned. It is important to remember that

these points are numbered downwards from 24 relative to the home position of each player.

```
\blackbar{n}    \whitebar{n}
```

Captured stones are placed on the middle bar. This is done using the commands `\blackbar` and `\whitebar`, where n gives the number of stones to put there.

```
\blackcube{n}    \whitecube{n}    \middlecube{n}
\showcube       \dontshowcube
```

The position and state of the cube are defined by the above commands. By default the cube is in the middle position, i.e., not owned by any player. It can be moved to either side by using `\whitecube` or `\blackcube` with n as its current number. If `\dontshowcube` is used, the cube is not displayed.

```
\smallboard      \normalboard      \bigboard
\fullboard       \halfboard
\blackonmove      \whiteonmove
\shownumbers      \dontshownumbers  \togglenumbers
\boardcaption{text}
```

To denote the size of the board three commands are available; the default is `\normalboard`. Normally a full board is produced but when discussing problems it is sometimes useful to draw only half of it, and this can be achieved using `\halfboard`. The next player to move can be specified, which changes the numbering of the board. Numbering of the board can be suppressed (`\dontshownumbers`) or toggled, i.e., shown from the view of the other player. In all cases you can give the board a caption with the `\boardcaption` command.

```
\begin{game}{Black}{White} ... \end{game}
```

The game environment is used to document the progress of a game. In contrast to the position environment, it is initialized not with an empty board but with the stones in their starting positions. Its two arguments can be used to specify information about the two players, e.g., their names or current scores. Figure 8.15 shows the first moves of a game.

```
\move{die}{moves}
\printboard      \rawboard
```

Moves are entered using the `\move` command. The values of the *die* are entered as a two-digit number. The *moves* are specified in the form $x-y$ and separated by commas with x and y denoting start and target point numbered as viewed by the current player.

• Black

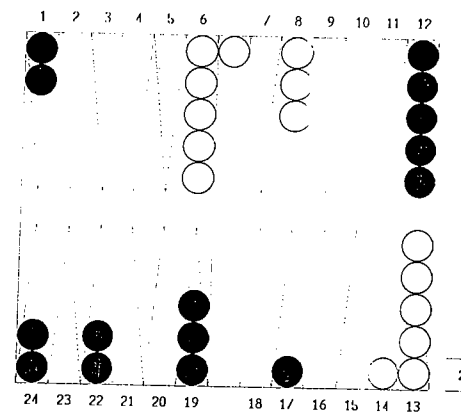
○ White

1. • 55 : 6-1*, 6-1, 8-3, 8-3

1. ○ 64 : 24-18, 18-14

2. • Takes.

2. ○ Doubles.



8-4-2

Black's turn 31

Figure 8.15: First moves of a backgammon game.

```
\usepackage {bg}
\begin{game}{Black}{White}
\normalboard
\whiteonmove
\move{64}{24-18, 18-14}
\move{55}{6-1,6-1,8-3,8-3}
\textmove{Doubles.}
\textmove{Takes.}
\takecube
\boardcaption
{Black's turn 31}
\printboard
\end{game}
```

All moves are recorded and output is preceded by a small checker of the right color. At any time during the game the `\printboard` command shows the current state of the board in a diagram centered with a caption, while `\rawboard` produces the board encapsulated in a `\mbox` without a caption—a version that can be used even in the middle of a text line.

```
\textmove{text}
\takecube
```

The `\textmove` command outputs *text* instead of a move and then switches to the opposite player. This can, for example, be used for double/pass actions or in cases where one player is unable to move at all. The `\takecube` command passes the cube to the opposite player and doubles its value.

<code>\fullincr</code>	<code>\indentwhite</code>	<code>\showmoves</code>
<code>\halfincr</code>	<code>\dontindentwhite</code>	<code>\dontshowmoves</code>

The output of the `\move` commands can be customized in several ways. By default the move number is incremented after every move; if `\halfincr` is used it is updated after every second move. The indentation of the moves by White can be suppressed if desired, and if `\dontshowmoves` is used the output is fully suppressed.

In addition to these commands, the customization commands of the position environment, such as `\dontshowcube`, etc., are available as well.

8.5 Card games

The symbols ♣, ♦, ♥, and ♠ are already part of standard (L^AT_EX): for some reason Donald Knuth included them into his mathematical fonts and made them accessible via the commands `\clubsuit`, `\spadesuit`, `\heartsuit`, and `\diamondsuit`. As these commands have long names and are available only in mathematical formulas, using them repeatedly in normal text is rather awkward. In such a case it might be best to define new commands to reduce the amount of typing, e.g.,

```
\newcommand\club{\ensuremath{\clubsuit}}
\newcommand\diam{\ensuremath{\diamondsuit}}
\newcommand\heart{\ensuremath{\heartsuit}}
\newcommand\spade{\ensuremath{\spadesuit}}
```

The above definitions can be used both in formulas and normal text, so `\heart_A` produces ♥A. Adding a few more definitions like the one below enables you to document rules and annotate games for any game that uses standard playing cards.

```
\newcommand\hand[4]{%
\begin{minipage}[t]{8em}
\begin{tabbing}
\spade{} \> #1\ \heart \> #2\
\diam \> #3\ \club \> #4
\end{tabbing}
\end{minipage}}
\hand{A}{A}{A}{Q}
```

The above definition shows the cards in the order suitable for poker or bridge games. In other card games ♣ is usually the highest color, in which case you should modify the definition accordingly.

8-5-1

8.6 Bridge

Using the ideas outlined in the previous section, Kees van der Laan developed L^AT_EX macros and a bidding environment for typesetting annotated bridge games in the fashion often found in bridge literature (van der Laan, 1989). In the following sections we show the commands that can be found in the file `bridge.tex` and develop some additional commands for special situations.

8.6.1 Card deals

The distribution of cards among the players is often shown in bridge literature as a diagram that shows the position and the hand of each player. Players are traditionally designated by the four points of the compass N, E, S, and W in the center of the graphic that symbolizes a play table. Such a graphic can be produced by the `\crdima` command.

```
\crdima{dealer}{info}{north}{west}{east}{south}
```

The first parameter *dealer* provides information about dealer and vulnerability, e.g., N/None for “North” dealer and vulnerability “none”. The second parameter *info* is text describing the game. The final four parameters describe the hands of each player in the order N, W, E, S.⁵ Usually they all contain a call to the `\hand` command, but it is also possible to put other or additional information into them.

Both commands together then produce diagrams like the following—a start situation showing the hands of all players.

S/None	♠ A 6 ♥ 7 4 2 ♦ J 6 3 ♣ A 8 6 4 3	Start distribution	\input {bridge.tex} \crdima{S/None} {\begin{tabular}{t}{l} Start\\distribution \end{tabular}} \hand{A 6}{7 4 2}{J 6 3} {A 8 6 4 3} \hand{Q J 10 9 7}{10 6} {Q 9 7 2}{J 5} \hand{8 5 3 2}{Q J 8 3} {K 4}{Q 10 9} \hand{K 4}{A K 9 5} {A 10 8 5}{K 7 2}
♠ Q J 10 9 7 ♥ 10 6 ♦ Q 9 7 2 ♣ J 5	N W E S	♠ 8 5 3 2 ♥ Q J 8 3 ♦ K 4 ♣ Q 10 9	

8-6-1

When discussing defense play two hands, your own and the hand of the dummy, are often shown. This can be displayed with the `\crdima` command by using empty arguments

⁵In the original article (van der Laan, 1989) the order is described as clockwise, i.e., N, E, S, W, but the macros as stored on CTAN in the file `bridge.tex` use a different sequence.

for the hands to be omitted. The example below shows an early situation in the above game.

S/None ♠ A E plays ♠3

♥ 7 4 2

♦ J 6 3

♣ A 8 6 4

♠ J 10 9 7

♥ 10 6

♦ Q 9 7 2

♣ J

N
W E
S

```
\input {bridge.tex}
\crdima{S/None}
{E plays \spade 3}
{\hand{A}{7 4 2}{J 6 3}
{A 8 6 4}}
{\hand{J 10 9 7}{10 6}
{Q 9 7 2}{J}}
{}{}

```

8-6-2

An en-dash (i.e., -- which produces –) is customarily used to denote a void. This is shown in the next example, which documents the same game after some more cards have been played.

♠ –

♥ 7 4 2

♦ J 6 3

♣ 8 6

♠ J 10 9

♥ 10 6

♦ Q 9 7 2

♣ –

N
W E
S

♠ –

♥ A K 9 5

♦ A 10 8 5

♣ –

♠ 8 5
♥ Q J 8 3
♦ K 4
♣ –

```
\input {bridge.tex}
\crdima{}{}
{\hand{--}{7 4 2}{J 6 3}
{8 6}}
{\hand{J 10 9}{10 6}
{Q 9 7 2}{--}}
{\hand{8 5}{Q J 8 3}
{K 4}{--}}
{\hand{--}{A K 9 5}
{A 10 8 5}{--}}

```

8-6-3

In discussing certain techniques of play often only the card distribution in a single suit is shown. In that case it would be nice not to use the `\hand` command in the arguments of `\crdima`, but unfortunately the result is not quite what we would expect.

♣ A Q ♣ K 6

♣ J 5

N
W E
S

♣ 7 4

```
\input {bridge.tex}
\crdima{}{}{\club{} A Q}
{\club{} J 5}{\club{} K 6}
{\club{} 7 4}

```

8-6-4

In this case a solution using the tabular environment gives better results. The first argument specifies the suit of interest, the others correspond to the four players (with the same order as in the `\crdima` command).

```
\newcommand{\crdexa}[5]{\renewcommand{\arraystretch}{1.2}%
\begin{tabular}{l|@{}c@{}|l}
```

```
\multicolumn{1}{c}{} & \multicolumn{1}{c}{\#1 \#2} \\ \cline{2-2}
& N & \\
\#1 \#3 & W\hfill E & \#1 \#4 \\
& S & \\ \cline{2-2}
\multicolumn{1}{c}{} & \multicolumn{1}{c}{\#1 \#5}
\end{tabular}}}
```

Note the use of the `\multicolumn` command to suppress the vertical lines in the first and last row. This definition produces the following output:

♣ A Q

N
W E
S

♣ J 5 ♣ K 6

♣ 7 4

8-6-5

```
\crdexa{\club}{A Q}{J 5}{K 6}{7 4}
```

8.6.2 Bidding

An important part of the bridge game is the initial bidding phase in which the players decide who plays the contract. To document such a bidding sequence Kees van der Laan introduced a bidding environment as an application of L^AT_EX's standard tabbing environment.

West	North	East	South
–	1♣	no	1♠
no	2♠	no	4♠
a.p.			

8-6-6

In discussing the theory of bidding, the bridge literature often shows such a bidding sequence together with the hand of one player. This can be achieved with the above commands as follows:

```
\input {bridge.tex}
\hand{3 2}{K J 10 8 5 2}
{Q 6 3}{K 3} \qqquad
\begin{minipage}[t]{8em}
\begin{bidding}
-- \> 1\diam\> 1\heart \> no \\
2\heart
\end{bidding}
\end{minipage}
```

8-6-7

An alternate form in which the bidding of only two partners is shown can be produced by defining a simple command `\bid` as follows. Here the `\hand` command is used to show

one hand and the bidding sequence is maintained with a tabular environment, so that this time we must use & characters in the fifth argument.

```

\input {bridge.tex}
\newcommand\bid[5]{\hand{#1}{#2}{#3}{#4}%
  \quad \begin{tabular}{t}{11}%
    self & partner \\
    \end{tabular}}
\bid{Q J 10 5}{A 10 9 6 3}{A 5 2}{3}
{ & 1\club \ 1\heart & 1\spade \ 4\spade }

```

8-6-8

8.7 Crosswords

Brian Hamilton Kelly introduced in an article in *TUGboat* (1990) a set of $\text{\LaTeX}$ macros to draw crosswords. His system is conceived so as to ensure that the “grid” comes together correctly. For example, Fig. 8.16 on the facing page shows a crossword grid⁶ to be completed for the clues given (the solution is found in Fig. 8.17 on p. 309).

```
\begin{crossword}{size}{show}
```

The crossword environment encloses the material used to typeset the crossword. Its first parameter *size* specifies the number of rows and columns to be used (thus crosswords produced with this package are always square).

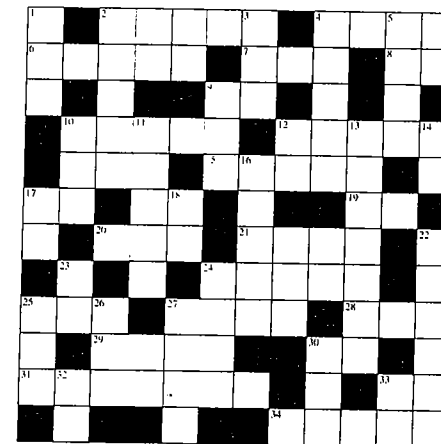
The value of the *show* parameter specifies whether or not the solution is shown in the grid. The usual case when publishing a crossword would be *n* or *N* in which case the grid is left empty. If *y* or *Y* is used the solution appears, something that is helpful during the development of the crossword. All other values bring up an interactive dialogue to determine whether or not the solution should be shown. Within the crossword environment clues are specified using the `\clue` command.

```
\clue{no}{dir}{col}{row}{answer}{text}{help}
```

The first argument *no* specifies the number of the clue, as shown in the first square on the grid and in the list of textual clues. The direction of the word in the grid is specified with the *dir* parameter, either *A* for *across* or *D* for *down*. The next two parameters, *col* and *row*, denote the position of the first square on the grid in which the solution should be entered. Columns are numbered left to right, rows top to bottom.

The *answer* argument holds the answer to the clue specified as *text*. This answer must be given completely in upper-case letters, while the clue *text* itself can contain arbitrary $\text{\LaTeX}$ text, including commands, and is printed below the grid prefixed by the clue number in

⁶ We thank Gerd Neugebauer for generating the puzzle from the index of this book. The clues (added by the authors) sometimes deliberately disguise this origin to increase the challenge.



ACROSS

- 2 Gap between tree node labels and the node in PSTricks (5)
- 4 Modern replacement for scissors and glue (4)
- 6 A Unicode $\text{\TeX}$ variant (5)
- 7 ... you always wanted to know but never dared to ask (3)
- 8 A graphics key that needs four numbers (2)
- 10 Called *bb* in Karl Berry's font-naming schemes (5)
- 12 A way to make your pages into thumbnails (5)
- 15 You can do it to a box but it isn't proper $\text{\LaTeX}$ (5)
- 19 In $\text{\LaTeX}$ denotes ρ ; in other circumstances might mean a word processor (2)
- 20 Result of a $\text{\TeX}$ run (3)
- 21 A language whose name should probably have five letters, but then it was developed for Unix (4)
- 24 It's not Intel (5)
- 25 A pointer misspelled (3)
- 27 Testing your $\text{\LaTeX}$ knowledge: $\prec$ (4)
- 28 Label for a signal line (3)
- 29 Another name for the $\text{\LaTeX}$ 3 project team on c.t.t. (4)
- 30 One way to get a sharp in $\text{\LaTeX}$ (2)
- 31 A figure or plan intended to explain rather than represent actual appearance (7)
- 33 72.27 to an inch (2)
- 34 see 1d (5)

DOWN

- 1 & a34 Grand wizard of $\text{\TeX}$ (3,5)
- 2 A ready-to-run $\text{\TeX}$ for Unix (5)
- 3 A novice golfer's dream (3)
- 4 $\text{\LaTeX}$ 2 ϵ name for document style (5)
- 5 Double beam above notes in $\text{\MusixTeX}$ (4)
- 9 Either or — mathematically speaking (3)
- 10 German beer (3)
- 11 Save your coordinates ($\text{\PSTricks}$) (5)
- 12 Approximation of $\text{\TeX}$'s version number (2)
- 13 A PostScript operator (7)
- 14 Probability function (2)
- 16 A divine messenger misspelled (5)
- 17 How do you get an $\hat{A}$? (2)
- 18 ξ (2)
- 22 $\text{\LaTeX}$ has rigid and rubber ones (6)
- 23 Amor uses them and $\text{\Xypic}$ calls them (2)
- 24 Length of the line segment where the connector joins the first node (4)
- 25 Files containing $\text{\LaTeX}$ font-definition documentation (3)
- 26 η — don't say this is all Greek to you (3)
- 27 $\perp$, also the first letters of everlasting (4)
- 30 We plot it in Chapter 4 (3)
- 32 $\text{\TeX}$'s name for inch (2)
- 33 Lula is chief of (2)

Figure 8.16: A sample crossword for you to fill in.

two lists, one for “across” and one for “down” clues. The final *help* parameter gives supplementary information printed in parentheses. It is commonly used to give the number of characters in the answer or, if the answer to a clue consists of more than one word, it might say something like see_2d.

As a practical example, here is the input for one of the clues of the crossword in Fig. 8.16 (except for the answer argument; that contains question marks to give you a chance to solve the crossword for yourself).

```
\clue{29}{A}{3}{10}{????}{Another name for the \LaTeX3
project team on c.t.t.}{4}
```

Several of the above arguments can contain quite complex hints. For example, if the answer to a clue consists of more than one word, you can put information about the other starting squares into the *no* argument as long as you set off the first number by a space or some other non-digit. In that case you should give the length of all words in the solution as a comma-separated list in the *help* argument. The other words of the solution can then be given with an empty *text* argument and refer back to the first clue in the *help* text.

```
\clue{1 \& a34}{D}{1}{1}{???}{Grand wizard of \TeX}{3,5}
\clue{34}{A}{8}{12}{?????}{see \textbf{1d}}{5}
```

If the *help* argument is completely empty, the clue is not shown in the list of clues beneath the crossword; only the clue number is displayed in the grid (the solution is recorded and checked internally). This facility is helpful for multiword solutions that occupy consecutive “lights” where such additional clues are usually omitted.

In the adaption for L^AT_EX 2_ε by Frank Mittelbach the column headings ACROSS and DOWN above the clues are no longer hardwired but can be customized by redefining the commands \ACROSStext and \DOWNtext, e.g., for a German crossword you could specify:

```
\renewcommand\ACROSStext{Waagrecht}
\renewcommand\DOWNtext{Senkrecht}
```

Using the second argument of the crossword environment to show the solution is normally useful only during development, as the list of clues below the crossword is still typeset. If you want to omit this list and just typeset a solution to “last month’s crossword,” use the starred version crossword* instead, which generates a smaller square to show the solution.

```
\begin{crossword*}{size}{title}
```

This environment has a second parameter that lets you put a title on top of the grid, say, to tell the reader which solution this is. In the body of the environment just use a copy of the \clue commands of the original crossword—doing this produces a solution grid like the one in Fig. 8.17.

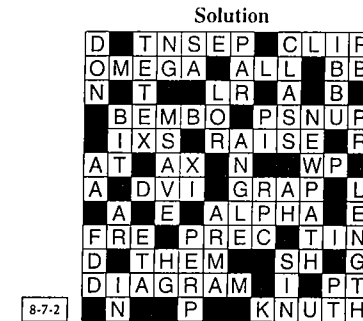


Figure 8.17: Solution to crossword in Fig. 8.16.

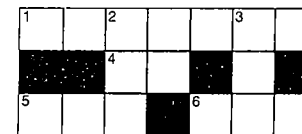
8.7.1 Recent developments

Just before this book was due at the printer, a new L^AT_EX crossword package appeared on CTAN. The cwpuzzle package by Gerd Neugebauer supports classical crossword puzzles, number puzzles, and fill-in puzzles. (In number puzzles all letters are replaced by numbers, the task being to figure out which number represents which letter; in fill-in puzzles the list of clues is replaced by a list of all words in the puzzle and the task is to find the correct places for the words).

In all cases the puzzle is constructed using the Puzzle environment:

```
\begin{Puzzle}{columns}{rows}
```

The arguments *columns* and *rows* specify the horizontal and vertical number of cells. Each line in the body of the environment describes the contents of the cells in one row. For instance,



```
\usepackage {cwpuzzle}
\begin{Puzzle}{7}{3}
|[1]T|A|[2]B|U|L|[3]A|R|.
|*|*|[4]I|M|*|X|*|.
|[5]D|E|G|*|[6]D|E|T|.
\end{Puzzle}
```

Cells are surrounded by vertical bars and the end of the line is marked with a dot. The contents of a cell can be a * denoting a black box or a single character, possibly prefixed by a number in brackets, to denote the start of a clue. For number puzzles you would put such a number in each cell, while for fill-in puzzles numbers are not needed at all. As a final possibility, you can put {} in a cell, in which case this cell is left completely empty and not even a frame is produced. This lets you produce non-rectangular puzzles or leave space for an ad or a picture in the middle of the puzzle.

```
\begin{PuzzleClues}{heading}
```

For a classical crossword puzzle, use this environment to typeset the list of clues. The individual clues are entered in the body using the command `\Clue`, as in the following example:

```
\Clue{1}{TABULAR}{Environment to produce tables}
```

Unlike the crossword environment, your input is *not* checked for correctness but used only to typeset the clue list—in fact the second argument is currently not used at all.

```
\begin{PuzzleWords}{number}
```

To produce a fill-in puzzle, use this environment as often as necessary. The *number* specifies the word length in the current list and in the body the `\Word` command is used to enumerate all words of that length, e.g., for the words of length two we have

```
\begin{PuzzleWords}{2}\Word{IM}\Word{UM}\end{PuzzleWords}
```

In puzzles like this one usually gives all “words” even if they are not meaningful.

Finally, the following commands can be used to produce the clues for a number puzzle:

```
\PuzzleLetters{letters} \PuzzleNumbers{letters}
```

`\PuzzleLetters` shows the list of all *letters* appearing in the puzzle (preferably in alphabetic order), while `\PuzzleNumbers` produces a list of boxes so that the reader can fill in the solution.

```
\begin{Puzzle}{7}{3} ... \end{Puzzle}
\PuzzleLetters{ABDEGILMRTUX} \medskip\par
\PuzzleNumbers{TABULRIMXDEG}
```

1	2	3	4	5	2	6
		7	8		9	
10	11	12		10	11	1

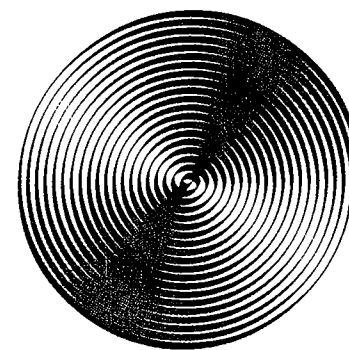
8-7-4

The following letters are used: ABDEGILMRTUX

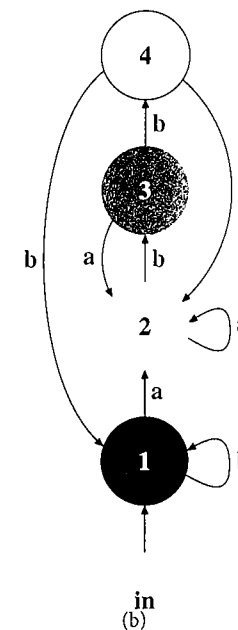
1	2	3	4	5	6	7	8	9	10	11	12
---	---	---	---	---	---	---	---	---	----	----	----

```
\PuzzleSolution \PuzzleUnsolved
```

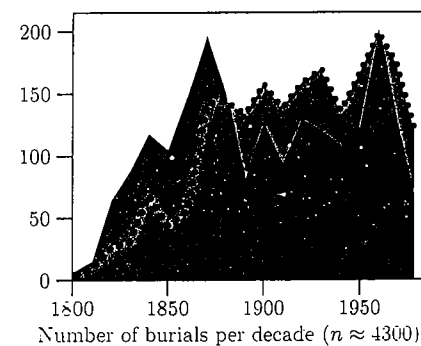
These two commands determine how any following puzzles are typeset. The command `\PuzzleSolution` shows the puzzle with the words filled in, while `\PuzzleUnsolved` (the default) presents the empty grid together with the clues. The package documentation describes several additional parameters that influence the layout and modify the default strings produced.



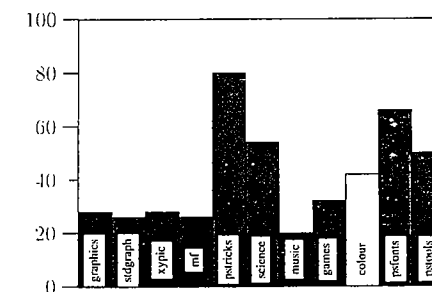
(a)



(b)

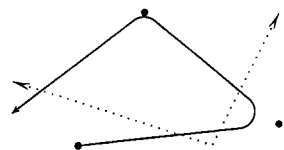


(c)

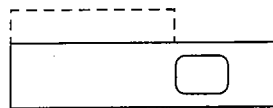


(d)

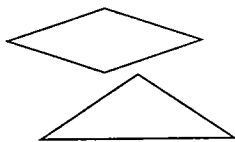
Color plate I: METAPOST examples.



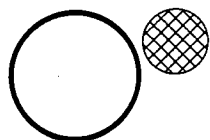
(a)



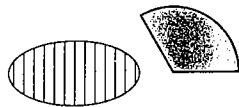
(b)



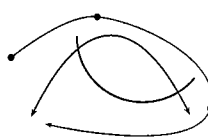
(c)



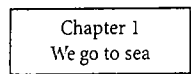
(d)



(e)



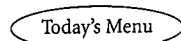
(f)



(g)



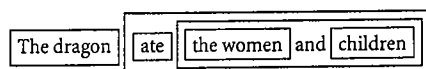
(h)



(i)

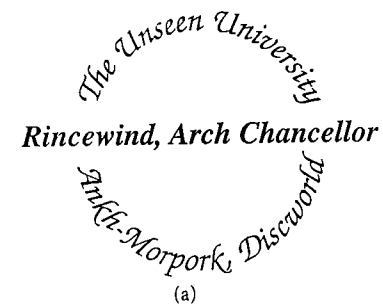


(j)

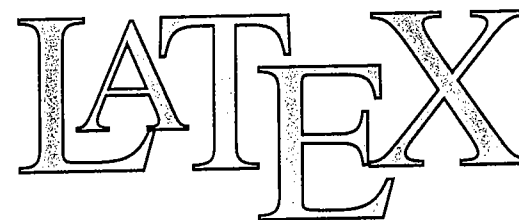


(k)

Color plate II: PSTricks examples 1.



(a)



(b)

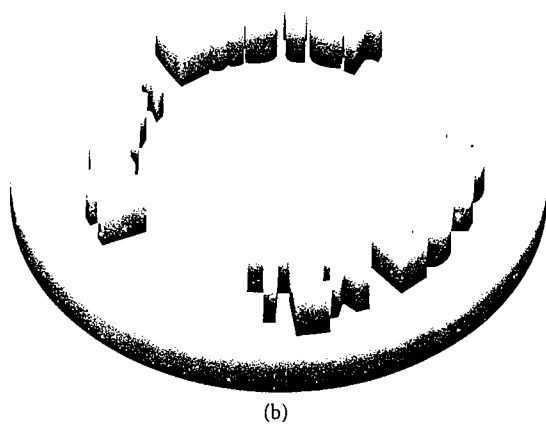
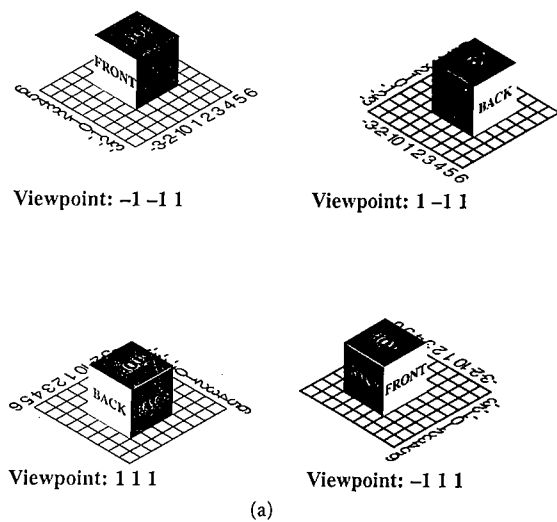


(c)

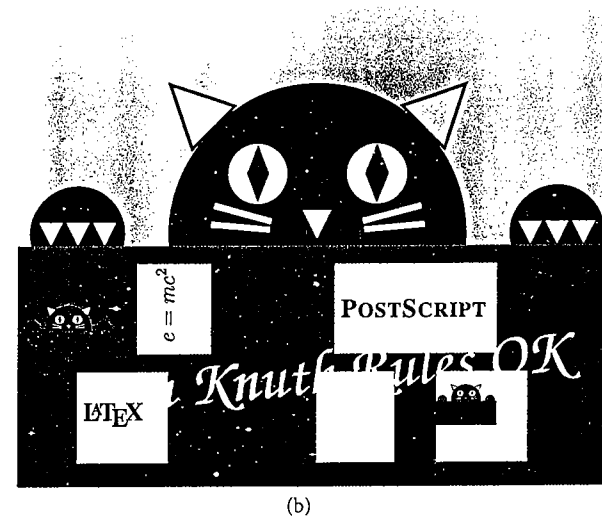
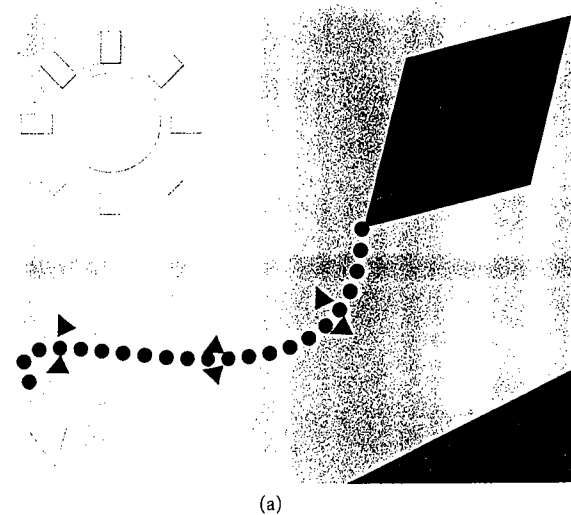


(d)

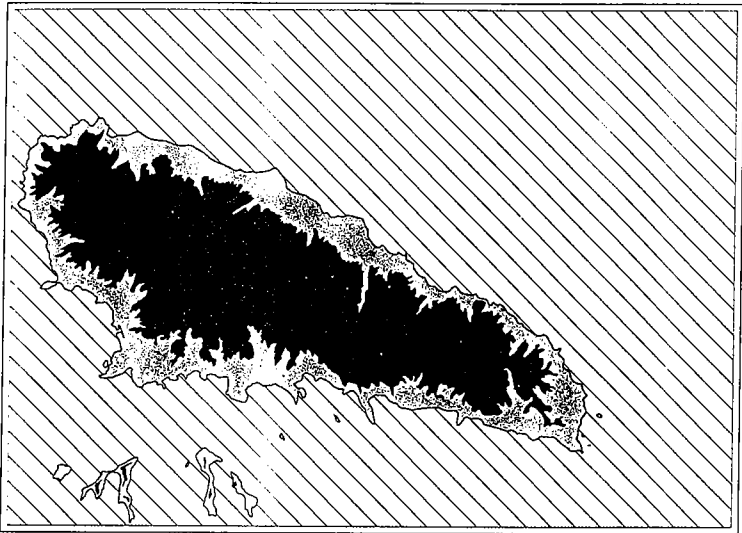
Color plate III: PSTricks examples 2.



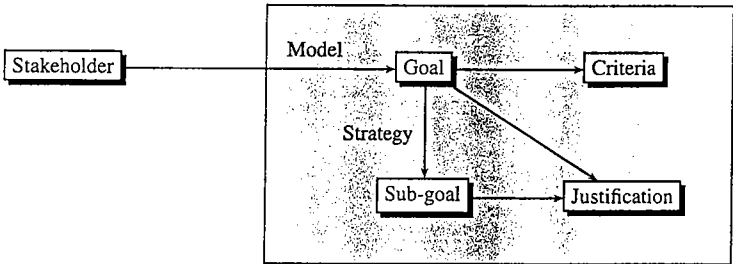
Color plate IV: PSTricks examples 3.



Color plate V: PSTricks examples 4.

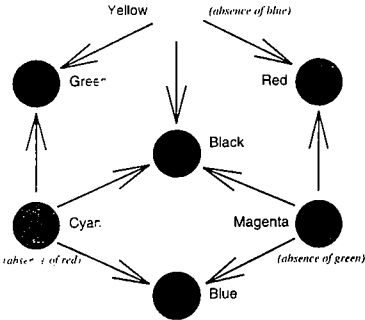


(a)

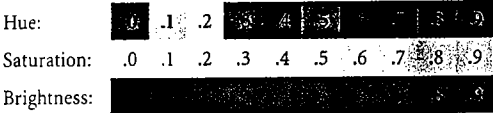


(b)

Color plate VI: PSTricks examples 5.



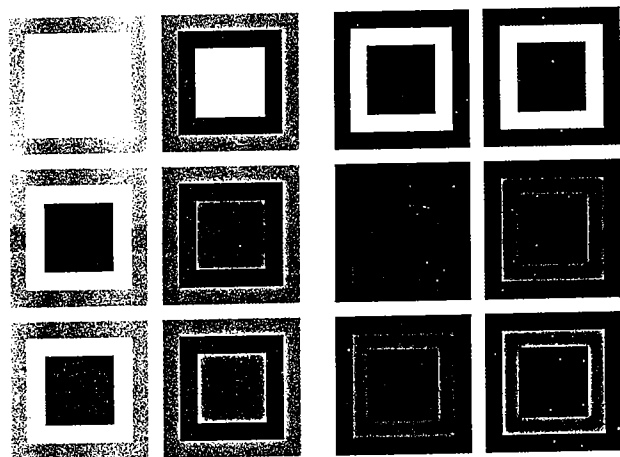
Color plate VII: The relation between the RGB and CMYK color models.



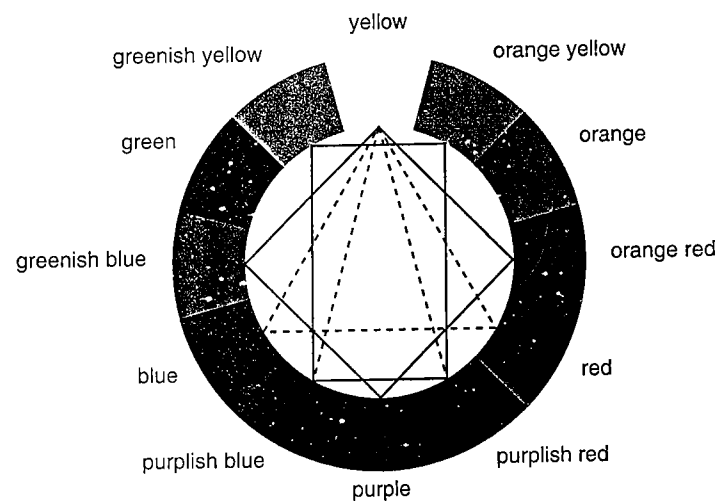
Color plate VIII: The HSB model.

	red	blue	green	yellow	white
China	happiness	sky clouds	Ming dynasty sky clouds	birth riches power	death purity
Egypt	death	virtue faith truth	fertility force	happiness prosperity	joie
France	aristocracy	freedom peace	criminality	worldliness	neutrality
India	life creativity		prosperity fertility	success	death purity
Japan	anger danger	infamy	future youth energy	grace nobility	death
U.S.A.	danger	masculinity	security	cowardice	purity

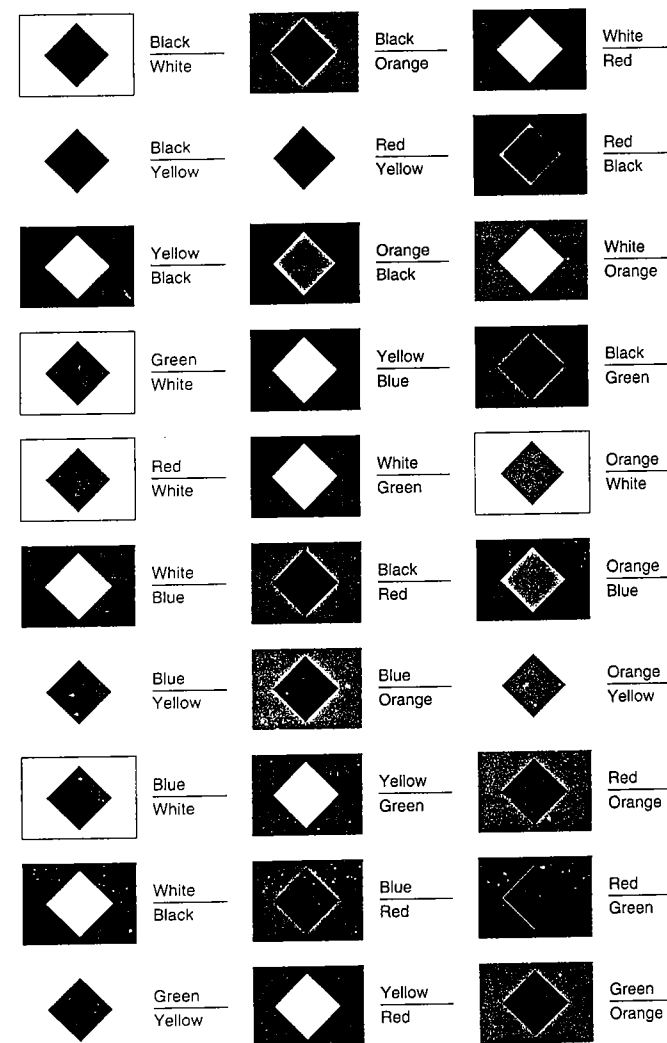
Color plate IX: Symbolic connotation of colors in different countries (see Greenwood, 1993).



Color plate X: Color harmony and the primary colors.



Color plate XI: Color harmonies and the chromatic circle (after Itten, 1974).



Color plate XII: Color contrasts for optimizing visibility and readability (after Duplan and Jauneau, 1986, p. 192).

GreenYellow	Yellow	Goldenrod	Dandelion
Apricot	Peach	Melon	YellowOrange
Orange	BurntOrange	Bittersweet	RedOrange
Mahogany	Maroon	BrickRed	Red
OrangeRed	RubineRed	WildStrawberry	Salmon
CarnationPink	Magenta	VioletRed	Rhodamine
Mulberry	RedViolet	Fuchsia	Lavender
Thistle	Orchid	DarkOrchid	Purple
Plum	Violet	RoyalPurple	BlueViolet
Periwinkle	CadetBlue	CornflowerBlue	MidnightBlue
NavyBlue	RoyalBlue	Blue	Cerulean
Cyan	ProcessBlue	SkyBlue	Turquoise
TealBlue	Aquamarine	BlueGreen	Emerald
JungleGreen	SeaGreen	Green	ForestGreen
PineGreen	LimeGreen	YellowGreen	SpringGreen
OliveGreen	RawSienna	Sepia	Brown
Tan	Gray	Black	White

Color plate XIII: Predefined "Crayola" colors in the *named* color model for the dvips driver.

Text starts off in green a little red nested blue text returning to green

1. magenta cmyk black
2. predefined blue gray text

Start with [black text] and [orange text], and return to black

Start in green, see [black text] and [orange text] and once again green

Black text on red background

Black text on blue background with blue frame

White text, blue background, red frame

White background Red background
Light background Dark background

Fun with color

Fun with color

Fun with color

Fun with color

❑ Farallon's Replica

- runs only on Windows;
- transforms PostScript type 1 fonts into bitmaps;
- well integrated in the environment.

❑ Common Ground of No Hands Software

- Mac and Windows;
- limited to four fixed resolutions;
- well integrated in the environment.

❑ Adobe's Acrobat

- Mac, Windows and Unix (Sun, HP, Irix);
- handles any PostScript document with Distiller;
- includes free Reader.

Color plate XIV: Examples of L^AT_EX color package.

LONDON					Price
Sydney	OG4G	Thu Oct 10	Mon Oct 21 or 28	11 or 18 days	10000000
		Thu Oct 17	Mon Oct 21 or 28	4 or 11 days	10000000
	OG7A	Sun Oct 13	Mon Oct 21 or 28	8 or 15 days	10000000
		Sun Oct 20	Mon Oct 28	8 days	10000000

Color plate XV: Simple multicolor table.

United Kingdom	London	Thames
France	Paris	Seine
Russia	Moscow	Moskva

Color plate XVI: Coloring table rules.

Table title

Description	Column 1	Column 2
Row one	mmmmmm	mmmmmm
Row two	mmmmmm	mmmmmm
Row three	mmmmmm	mmmmmm
Row four	mmmmmm	mmmmmm
Totals	mmmmmm	mmmmmm

(a)

Table title		
Description	Column 1	Column 2
Row one	mmmmmm	mmmmmm
Row two	mmmmmm	mmmmmm
Row three	mmmmmm	mmmmmm
Row four	mmmmmm	mmmmmm
Totals	mmmmmm	mmmmmm

(b)

Table title		
Description	Column 1	Column 2
Row one	mmmmmm	mmmmmm
Row two	mmmmmm	mmmmmm
Row three	mmmmmm	mmmmmm
Row four	mmmmmm	mmmmmm
Totals	mmmmmm	mmmmmm

(c)

Table title		
Description	Column 1	Column 2
Row one	mmmmmm	mmmmmm
Row two	mmmmmm	mmmmmm
Row three	mmmmmm	mmmmmm
Row four	mmmmmm	mmmmmm
Totals	mmmmmm	mmmmmm

(d)

Color plate XVII: Two-color tables (part I).

Table title

Description	Column 1	Column 2
Row one	mmmmmm	mmmmmm
Row two	mmmmmm	mmmmmm
Row three	mmmmmm	mmmmmm
Row four	mmmmmm	mmmmmm
Totals	mmmmmm	mmmmmm

(a)

Table title

Description	Column 1	Column 2
Row one	mmmmmm	mmmmmm
Row two	mmmmmm	mmmmmm
Row three	mmmmmm	mmmmmm
Row four	mmmmmm	mmmmmm
Totals	mmmmmm	mmmmmm

(b)

Table title

Description	Column 1	Column 2
Row one	mmmmmm	mmmmmm
Row two	mmmmmm	mmmmmm
Row three	mmmmmm	mmmmmm
Row four	mmmmmm	mmmmmm
Totals	mmmmmm	mmmmmm

(c)

Table title		
Description	Column 1	Column 2
Row one	mmmmmm	mmmmmm
Row two	mmmmmm	mmmmmm
Row three	mmmmmm	mmmmmm
Row four	mmmmmm	mmmmmm
Totals	mmmmmm	mmmmmm

(d)

Table title

Description	Column 1	Column 2
Row one	mmmmmm	mmmmmm
Row two	mmmmmm	mmmmmm
Row three	mmmmmm	mmmmmm
Row four	mmmmmm	mmmmmm
Totals	mmmmmm	mmmmmm

(e)

Table title		
Description	Column 1	Column 2
Row one	mmmmmm	mmmmmm
Row two	mmmmmm	mmmmmm
Row three	mmmmmm	mmmmmm
Row four	mmmmmm	mmmmmm
Totals	mmmmmm	mmmmmm

(f)

Color plate XVIII: Two-color tables (part II).

Table title		
Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

(a)

Table title		
Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

(c)

Table title		
Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

(e)

Table title		
Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

(b)

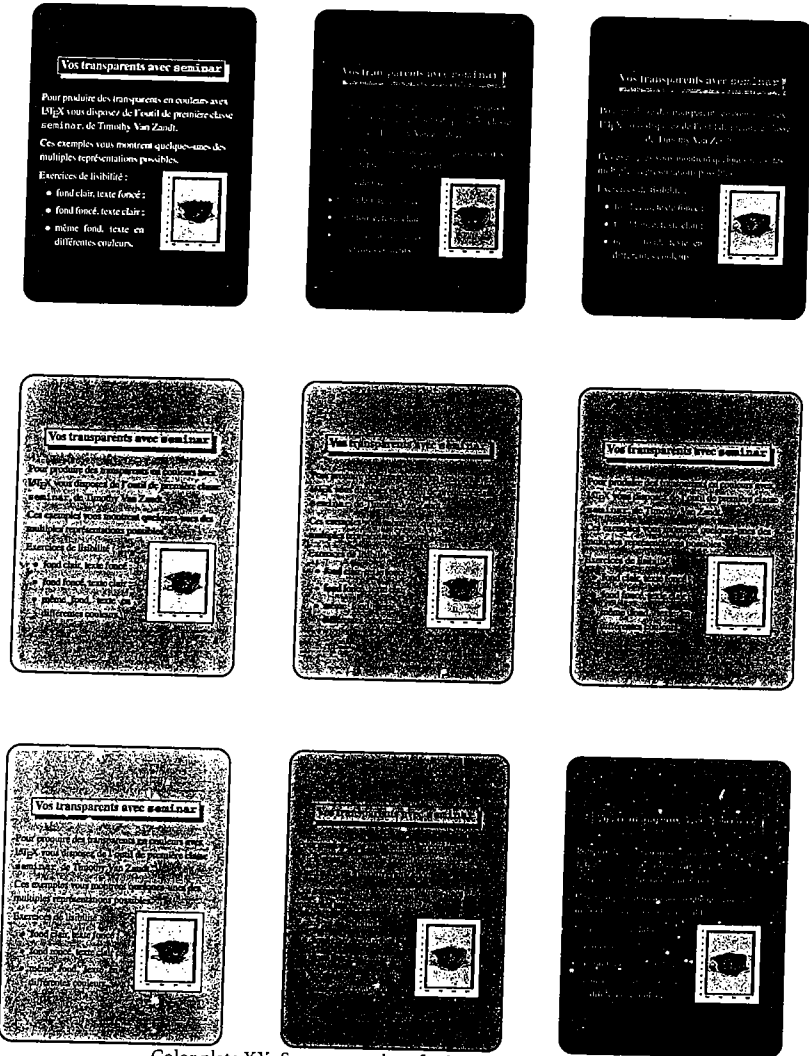
Table title		
Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

(d)

Table title		
Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

(f)

Color plate XIX: Two-color tables (part III).

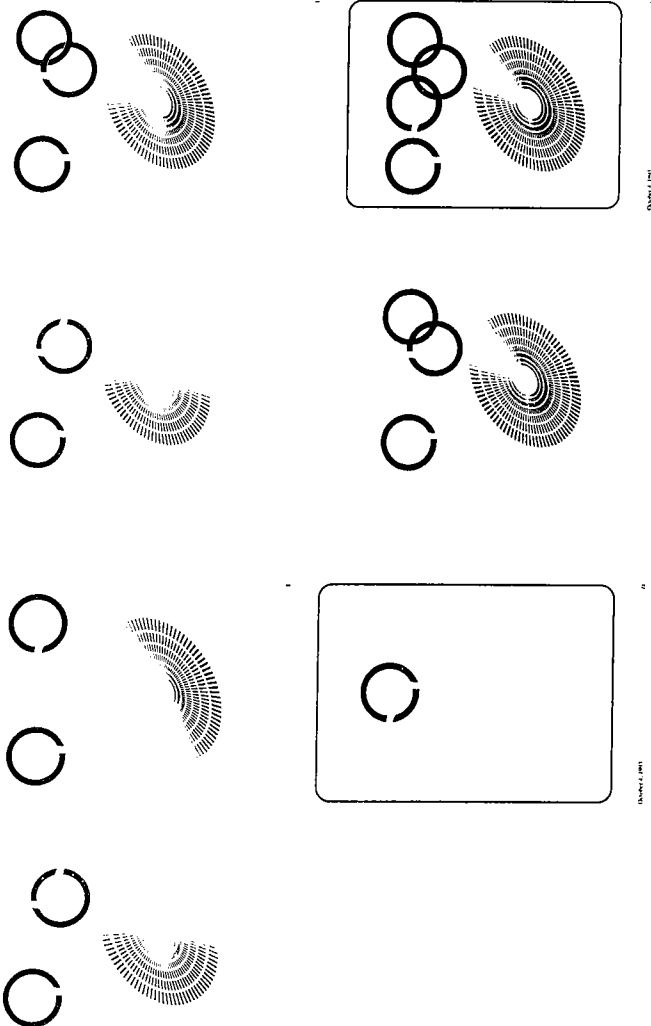


Color plate XX: Some examples of using color with seminar.

The four CMYK separations, cumulatively

CMYK separations

Color plate XXI: The separation of colors in the CMYK model.



CHAPTER 9

The world of color

For many people, color is indispensable for effective graphics. Modern interactive drawing packages all support coloring of lines, filling objects with color, etc., and all the standard bitmap file formats such as GIF (Graphics Interchange Format), JPEG (Joint Photographic Experts Group), PBM (Portable Bitmap), TIFF (Tagged Image File Format) and BMP (Windows Bitmap), as well as Encapsulated PostScript, support color. Thus, if you generate a picture with a drawing package, and import it into your $\text{\LaTeX}$ document using the packages described in Chapter 2, you should have no problems if your printing or viewing device supports color. However, you do have to know something about how color is represented and what color model you are using. We discuss this in the first part of this chapter.

If you prepare your graphics using $\text{\LaTeX}$ itself or simply want colored text, you need some special support from both $\text{\LaTeX}$ and your driver. The main body of this chapter describes the standard $\text{\LaTeX}$ color package, which we believe is powerful enough to meet almost all needs and is capable of working with most other packages. One area the package does not address, however, is coloring tables, and for this we describe a special package, `colortbl`.

$\text{\LaTeX}$ users often request color for use on transparencies for presentations. The standard color package can, of course, be used with the standard slides class, but we devote some space to explaining a more sophisticated class, seminar, and giving examples of its facilities.

At the end of the chapter, we consider the practicalities of serious color printing and color separation. This aims to introduce you to the concepts you will have to discuss with printing people, rather than trying to teach you the whole art of color separation.

Because color printing is still quite expensive, we are limited to a single group of color plates for this book. We print most of the pictures anyway in the main text, using grayscale to show color effects, and then repeat selected examples in the color plates. We indicate when the reader should refer to the full-color version.

9.1 An introduction to color

You should think about color in a document as a tool, not a gadget to merely make the pages look "prettier." The French painter Eugène Delacroix wrote (Duplan and Jauneau, 1986):

Color is above all that part of art which bestows the gift of magic; while subject, form, and line are concerned firstly with reasoning, color has no relationship with intelligence; it has power over feeling, and invades your senses.

This sentence summarizes perfectly the rôle that color plays in the construction of the visual image. By choosing the right color, the typographer or painter can add an affective value to the message, thus making it understood more clearly.

9.1.1 Color theories

Since prehistoric times color has played an important rôle in visual communication, as the colored depictions of animals in the caves of Altamira or Lascaux clearly indicate. The Egyptians, Assyrians, and other ancient Middle East cultures, and later the Phoenicians, knew how to produce paint and dyes, and they loved colored stones. But it was the Greek philosopher Aristotle who first, in the 4th century B.C., tried to actually "understand" color by studying the mixing of colors in different colored glass. To explain his observations he postulated that each color was a mixture of white and black, the dark colors being produced by the reflection of light by the medium. This interpretation, in which colors were ordered on a straight line from the lightest to the darkest, starting with white and ending with black, lasted for almost 2200 years. It was only in the 17th century, thanks to Isaac Newton's experiments with glass prisms that the spectral theory of light was discovered. Newton ordered the colors on a closed circular ring, a representation still in use today.

Goethe at the beginning of the 19th century dedicated much of his life to the study of physics and chemistry, and he was particularly fascinated by color. Even though his color theory is no longer accepted today, his discussion of the matter was at the basis of some significant progress in the field. Thomas Young, an English doctor, introduced in 1801 the three-color theory of light; this was further developed by Hermann von Helmholtz, and their theory of three-color vision is called the *Young-Helmholtz Law*. It was the work of Maxwell, Helmholtz and Grassmann that formed the basis of the science of colorimetry. The laws of mixing colored light were first formulated by Grassmann (1853), who was the first to show clearly the relationship between light and color. He summarized his findings in three theorems, *Grassmann's Laws*:

- (a) Colors obtained by *additive* color mixing are completely determined by their three-color components and do not depend on their spectral composition (*color mixing theorem*);
- (b) Three numbers are necessary and sufficient to fully describe a color (*additivity theorem*);

- (c) In daylight conditions, color sensation does not depend on light intensity (*proportionality theorem*).

Recent studies and experiments have consolidated our knowledge in the field of color (see Dérivé, 1993; Gerritsen, 1988; Nemcsics, 1993, for more details).

9.1.2 Color systems

Today we know that several representations may be used to describe colors, any of which can be chosen depending on the targeted application domain, for instance additive or subtractive color mixing, color perception theory, television, artwork, dyes, gray levels. In particular, PostScript subdivides color spaces into three categories:

- *Device color spaces*: colors or gray levels are directly expressed in units the output device understands. Examples of such models are RGB (*Red, Green, Blue*) and a variant for television YIQ, CMYK (*Cyan, Magenta, Yellow, Black*) and HSB, also called HSV (*Hue, Saturation, Brightness* or *Value*). See Color Plate VII and Color Plate VIII for demonstrations of the RGB and HSB models.
- *CIE color spaces*: colors are specified in a device-independent way. In 1931 the *Commission Internationale de l'Eclairage* (CIE) established a colorimetric system defining all colors in an unambiguous and objective way. In 1964 (CIEUCS) and 1976 (CIELAB/CIELUV), extensions were introduced in the CIE model to correct certain shortcomings of the original version.
- *Special color spaces*: these are used for special applications, such as patterns, color maps, and separations.

An introduction to the use of color with PostScript can be found in Kunkel (1990) and McGilton and Campione (1992). A detailed description of all the models and of the algorithms to transform from one space to another is beyond the scope of this book. More details can be found in Adobe Systems (1990), Foley et al. (1990), Jackson et al. (1994) and Salmon and Slater (1987); Gerritsen (1988) classifies all the color models that have been proposed into groups (see also Nemcsics, 1993). We deal simply with the RGB and CMYK models, since these are widely used in the printing and computer industries.

The additive RGB color space

In RGB space all colors are obtained by the superposition (addition) of three *primary* components, defined by the CIE as red (700 nm), green (546.1 nm), and blue (435.8 nm). In this model the electrons of the cathode ray tube hit phosphorous elements that emit light of the right wavelength, and combine (add) to give the desired color (see Color Plate VII).

The subtractive CMYK color space

The printing industry does not use the RGB primary colors, but rather their complements: cyan, magenta, and yellow. This is because inks "subtract" their complementary colors from the white light that falls on the surface; e.g., cyan ink absorbs the red component of white light and thus, in terms of the additive primaries, cyan is white minus red, i.e., blue plus green. Similarly, magenta absorbs the green component and corresponds to red plus blue, while yellow, which absorbs blue, is red plus green (see Color Plate VII). In practice the printing industry uses a process called "undercolor removal" in which a fourth "color", black, is added so as to create a darker black than is available by mixing the three colored inks. This color model is called the CMYK model, where the final "K" stands for the black component. The effect of the black component is handled as follows:

- *calculate black component*: it is the minimum of the components C, Y, and M, because this number corresponds to the value of which all three inks are superimposed to yield black: $K = \min(C, Y, M)$;
- *undercolor removal*: subtract this black component K from the three components C, M, and Y to compensate for the amount of black introduced by the use of black ink: $C = C - K$; $Y = Y - K$; $M = M - K$.

9.1.3 Symbolic values of color

The significance of color, like symbols, varies greatly across cultures. Color Plate IX shows some cultural connotations for various colors (following Greenwood (1993)). For example, red means danger in the United States, as in most Western cultures, but it is a symbol for life and creativity in India. It is important to be aware of these differences when designing a poster, book cover or computer interface.

To minimize ambiguity, international standards groups have chosen colors with generally understood significances. In the field of traffic signs, for instance, red is a sign for danger, while green indicates health services, or says that the road is clear. Blue, white, and black are only secondary colors in such contexts, used for information on road surfaces, rest areas, etc. Similar standards have been developed for plumbing and electrical wiring.

A color with a generally-recognized significance can help to reinforce an idea and can play an important psychological rôle in creating an atmosphere or conveying a cultural, religious, or political message. It should be noted, however, that every human activity, profession, and interest group, even within the same culture, has its own "color" jargon. You should thus always be aware of possible side effects of using a color in any given context.

9.1.4 Color harmonies

Color harmonies are arrangements of color pleasing to the eye. Scores of books have been written on color harmony, and the conclusions of many of these works are often contradictory. Reasons for this are not hard to find (Judd and Wyszecki, 1963):

- Color harmony is a matter of individual emotional response, of likes and dislikes, and even the same person can change his or her mind over time, since old combinations can become boring while frequent exposure to new combinations can make us appreciate them.
- Color harmony depends on the absolute size of the areas covered by the colors as well as on the design and the colors themselves. For instance, a nice-looking mosaic pattern can become quite unattractive when magnified by a factor of ten.
- Color harmony depends on the relative sizes of the areas as well as on their colors.
- Color harmony depends on the shape of the elements as well as on their colors.
- Color harmony depends on the meaning or interpretation of the design as well as on the colors; clearly color harmony for a portrait painter is quite a different matter from color harmony in abstract design or typography.

Nonetheless, it is still interesting to try to formulate a few principles for the construction of color harmonies.

- Color harmony results from the juxtaposition of colors selected on an orderly plan that can be recognized and emotionally appreciated.
- When comparing two similar sequences of color, observers choose the one most familiar as the most harmonious.
- Groups of colors that seem to have a common aspect or quality are considered to be harmonious.
- Colors are perceived as harmonious only if the combination of colors has a selection plan which is unambiguously recognizable.

It has been observed experimentally that the eye prefers combinations in which primary colors are in equilibrium with their complementary colors, and that our perception of a color changes in relation to its environment. Color Plate X shows the effects of *saturation* or *absorption* of the three primary colors with respect to white (leftmost column) black (second column) and with respect to its complementary color (third column) and a gray tone of the primary color itself (rightmost column).

Itten (1974) uses a model based on a harmonic color circle subdivided into twelve equal parts to explain his theory of colors (see Color Plate XI). It contains the three primary colors yellow, red, and blue, 120° apart. Their complementary colors, purple, green, and orange, also called the secondary colors, are positioned diametrically opposite their respective primaries. The circle contains six more colors, intermediate between each primary and its adjacent secondaries. The harmonic color circle is only a simplification. Indeed, all possible colors can be represented on the surface of a sphere that has the harmonic colors at its equator, white at the north pole, and black at the south pole. Thus moving from the equator towards the south (north) pole yields darker (lighter) variants of a given color. This also

means that for each point on the color sphere, there exists a diametrically opposed point with complementary characteristics, e.g., light greenish blue is opposed to dark orange red. Centuries of artistic experience have shown that a few simple basic rules let artists construct effective color harmonies in their works. Following Itten, we discuss a few of them below.

Two-color harmonic combinations

Complementary colors lying at diametrically opposite points of the color circle (sphere) define two-color harmonies, like the 2-tuples (red, green), (blue, orange), plus the almost infinite number constructed using possible combinations on the sphere.

Three-color harmonic combinations

When an equilateral triangle is constructed inside the color circle, the colors at each edge form a three-color harmony. The most fundamental 3-tuple (yellow, red, blue) is well known in all forms of art, publishing, and publicity, for its effectiveness and it can be used in a wide variety of patterns and layouts, and in all kinds of light and dark combinations. The secondary color 3-tuple (purple, green, orange) has also a strong character, and is frequently used. Other 3-tuples are also possible and you can construct harmonic 3-tuples by replacing the equilateral triangle by an isosceles one, or by working on the color sphere and combining light and dark variants. As a special case, you can put one edge of the triangle at the white point (north pole) to create the harmony (white, dark greenish blue, dark orange red), or on the black point (south pole) to create the harmony (black, light greenish blue, light orange red).

Four-color harmonic combinations

You can construct a four-color harmony by taking the colors lying on the edges of a square in the color circle, e.g., the 4-tuple (yellow, orange-red, purple, greenish blue). It is also possible to use a rectangle combining two pairs of complementary colors.

Higher order harmonies (like six-color) are equally easy to obtain using similar geometric models, on the color circle or the color sphere. Note, however, that each combination has its own character and set of basic laws, and only long experience can show which of the various sets of harmonies is most efficient for a given application.

9.1.5 Color and readability

The readability of a message or sign is closely linked to how our visual system processes the information presented to it. Factors influencing the visibility of colors are:

- *intensity*: pure colors of the spectrum have the highest intensity;
- *contrast*: between the different colors;
- *purity*: pure colors are more visible than graded variants, in which white has been added, making them fainter, or black, making them darker.

Color Plate XII shows some of the most effective color contrasts for maximum readability or visibility, e.g., on slides, road signs, or publicity leaflets.

9.2 Colors and L^AT_EX — the color package

L^AT_EX's color support is built around the idea of a system of *color models*. The color models supported by a driver may vary, but typically include:

rgb *red green blue*. A comma-separated list of three real numbers between 0 and 1 (the color components in function of the additive RGB model.)

cmyk *cyan magenta yellow black*. A comma-separated list of four real numbers between 0 and 1 gives the components of the color according to the subtractive CMYK model used in most printers.

gray *grayscale*. A single real number between 0 (black) and 1 (white).

named A name selected from a list of predefined colors is used.

The named color model is not supported by all drivers. When it is available then the driver may predefine the color for each name according to one of the other color models, or refer to color names supported by the printer or page-description language like PostScript. The dvips driver offers 64 "Crayola" colors, as originally proposed by Jim Hafner in his *colordvi* and *foitex* packages; these names and colors are shown in Color Plate XIII. It is important to realize that these colors are defined not according to CMYK values, but at the PostScript level in a replaceable header file that can be customized for different printers. This allows some independence from fixed CMYK or RGB values that may not look right on your device.

Users can set up color name definitions with the `\definecolor` command described in Section 9.2.2 on p. 319, which associates the names with numeric values from one of the color models. Other color models can also be used (such as Pantone, X11, HSB, etc.). In each case the relevant code for the new models must be defined in the driver configuration files of the color package. Some drivers even allow the color intensity to be specified, e.g., `ForestGreen .4` indicates a particular green at 40% intensity.

9.2.1 Supported options

As discussed in Section 2.1.1 with reference to the graphics commands, it is ultimately the *dvi* driver for your particular printer that paints the colors on the output medium. Therefore the driver must be declared as an option on the `\documentclass` or `\usepackage` commands or specified in the configuration file `color.cfg`, using, for instance, the command

```
\ExecuteOptions{dvips}
```

It is possible to turn off the effect of all color commands by specifying the `monochrome` option. This can be useful when previewing.

Three options control the behavior of the named model. By default the named color model has no predeclared names; the `dvipsnames` option defines all the names in the color prologue of `dvips`. The `dvips` option automatically implies `dvipsnames`, unless you explicitly overrule this choice by specifying the `nodvipsnames` option. Conversely, if you want to use these names with another driver, e.g., `Textures`, then you have to specify the `dvipsnames` option. Finally, the `usenames` option makes all the names of the named color model available as $\LaTeX$ color names (described in Section 9.2.2).

Comparatively few drivers support color fully: `dvips`, `dvipsone`, `pctexp`s, and `Textures` offer the best support at the time of writing.

The pstcol package

The `PSTricks` package provides a full featured interface between PostScript and $\LaTeX$, discussed in Chapter 4. Unfortunately, the color support in that package is slightly incompatible with the color mechanism defined in the color package. To correct this (temporary) incompatibility the `pstcol` package modifies a small number of internal `PSTricks` functions. Since the `PSTricks` package is loaded by `pstcol`, you can use any colors defined by color package commands within `PSTricks` commands, and vice versa.

9.2.2 Using colors

The syntax for color changes is similar to that of font changes and has two forms, one declarative and the other with arguments for local changes.

Using predefined colors

This is the basic way of using a simple range of colors defined by the color package. The user need not know the color model used. All drivers define the colors `black` and `white`; if they support the RGB model, they define `red`, `green` and `blue`, and if they support the CMYK model, they define `cyan`, `magenta`, and `yellow`.

```
\color{name}      \textcolor{name}{text}
```

The first form, `\color`, is a state changing command that sets colors to remain “active” until the end of the current (implicit or explicit) group. The second form, `\textcolor`, is a command suitable for short colored text fragments — `\textcolor{name}{text}` is equivalent to `{\color{name}text}`. This example is also printed in Color Plate XIV.

Text starts off in green a little red nested blue text
returning to green

```
\usepackage {color}
\color{green}
Text starts off in green
\textcolor{red}{ a little red}
{\color{blue}nested blue text}
returning to green}
```

9-2-1

Specifying values for color models

If the predefined colors are not sufficient, you can specify a color explicitly with respect to a given color model.

```
\color[model]{specification}
\textcolor[model]{specification}{text}
```

Note that the meaning of the first mandatory argument (*specification*) to the `\color` and `\textcolor` commands changes when the option *model* is given; instead of a predefined color name, you supply the appropriate values for the model.

The example in the previous section (with predefined colors) can be rewritten in terms of the RGB model as follows (see also Color Plate XIV):

```
\usepackage {color}
{\color[rgb]{0,1,0}
Text starts off in green
\textcolor[rgb]{1,0,0}{ a little red}
{\color[rgb]{0,0,1}nested blue text}
returning to green}
```

9-2-2

Another example (also in Color Plate XIV) that mixes predefined and local color specifications is the following list:

```
\usepackage {color}
\begin{enumerate}
\item \textcolor{cmyk}
      {0,1,0,0}{magenta cmyk} black
\item \color[gray]{0.5}
      \textcolor{blue}{predefined blue}
      gray text
\end{enumerate}
```

9-2-3

As the use of local color specifications throughout a document reduces its portability and makes updating it cumbersome, it is usually best to avoid using this feature. A better approach is to define your own colors in the preamble and then refer to them by name, as discussed in the next section.

Defining colors

The colors `black`, `white`, `red`, `green`, `blue`, `cyan`, `magenta`, and `yellow` should be predefined by any driver. Packages can define other colors. To define colors yourself, you can use the `\definecolor` command.

```
\definecolor{name}{model}{color specification}
```

This defines *name* as a color that can be used in later color commands, for example

```
\definecolor{Myorange}{cmyk}{0,0.42,1,0}
\definecolor{Myblue}{rgb}{0.8,0.85,1}
\definecolor{Mygrey}{gray}{0.75}
```

After these definitions, Myorange, Myblue, and Mygrey can be used in addition to the built-in colors.

Using named colors

To use the “named” colors supported by the driver, you specify the color name and the model, as described in the previous section; thus `\color[named]{SpringGreen}` selects the color SpringGreen.

Alternatively, rather than using the `named` option each time, it might be preferable to define an alias for that color with the `\definecolor` command, for instance:

```
\definecolor{MyGreen}{named}{SpringGreen}
```

and then refer to SpringGreen via `\color{MyGreen}`.

If you just want to make available all the existing names from the named model as if you had done a `\definecolor` for each, you can specify the `usenames` package option. This effectively calls `\definecolor` for every color of the named model and allows `\color{SpringGreen}` to be used in addition to `\color[named]{SpringGreen}`.

Using the named color model has certain advantages over using other color models. First, since the `dvi` file contains a request for a color by *name*, the actual mix of primary colors used to obtain the color requested can be tuned to the characteristics of a particular printer. In particular, the `dvips` driver uses an external header file `color.pro` that contains the definitions of the color names. In order to ensure constant colors when printing on different devices, you should ideally produce a different version of this file for each output device that is to be used.

Second, apart from the so-called “process colors” that are produced by mixing primary colors during the print process, you may want to use “spot” or “custom” colors for which a particular color name refers not to a mix of primaries, but to a particular ink. The parts of the document using this color are printed separately using this named ink color.

Special concerns with color in L^AT_EX

You need to be aware of some special situations relating to stored boxes in L^AT_EX. The following example (also printed in Color Plate XIV) shows that color is defined when the box

is *created* with the `\sbox` command, not when it is *used* (i.e., the color characteristics are stored with the box and the surrounding color does not influence that color).

	<code>\usepackage{color}</code>
	<code>\newsavebox{X}</code>
	<code>\sbox{X}{[black text] and</code>
	<code>\color{cmyk}{0,0.6,0.8,0}[orange text]}</code>
Start with [black text] and [orange text], and return to black	Start with <code>\usebox{X}</code> , and return to black
Start in green, see [black text] and [orange text] and once again green	<code>\color{green}</code> Start in green, see <code>\usebox{X}</code> and once again green

This parallels L^AT_EX's handling of font attributes, which are also fixed when the box is created. However, the internal mechanisms used are quite different and this can lead to unpleasant surprises if new commands are not carefully designed.

When text is stored inside a box, e.g., with the `\sbox` or the low-level `\hbox` command, each character carries with it a note about the font in which it is set. In contrast, color is handled by putting “start color” and “end color” messages into the output, and thus if such a constructed box is decomposed with low-level T_EX commands the correspondence between color and text can be lost. As an example of the problems this can cause, consider a situation in which “red” starts at the bottom of one page and continues on the next; when a T_EX format that is not aware of “color”, such as plain T_EX (but not L^AT_EX 2_ε), comes to format these pages, the footer and header on the page break in question are also printed in red. Similar situations arise when such a format handles lists (e.g., the item labels might be printed in the wrong color).

Standard L^AT_EX goes a long way to support color and many of the potential problems are circumvented in the main L^AT_EX code (you do not in practice have to worry about coloring headers and footers by accident), but you may still get unexpected color effects if packages are used that are not written with the restrictions of color support in mind. These problems are discussed in detail in the document ‘L^AT_EX 2_ε for class and package writers’ distributed with L^AT_EX, and in Rokicki (1994). Unless you are a confident L^AT_EX programmer, it is sensible to confine your use of color to simple situations using official L^AT_EX rather than low-level T_EX commands (e.g., `\savebox` instead of `\setbox`, `\mbox` rather than `\hbox`, etc.), so you can be sure to obtain the expected result.

9.2.3 Page color

```
\pagecolor{name}
```

The background color of the whole page can be set by using `\pagecolor`, which takes the same argument forms as `\color` but sets the background color for the current and all subsequent pages. Since it is a global declaration, you must use `\pagecolor{white}` to return to a white background (the default setting).

9.2.4 Colored box backgrounds

Two commands similar to `\fbox` produce boxes with backgrounds shaded in a given color.

`\colorbox{background color}{text}`

The `\colorbox` command puts the *text* into a box and colors the background. This background extends `\fboxsep` in all four directions.



```
\usepackage {color}
\definecolor{Light}{gray}{.80}
\definecolor{Dark}{gray}{.20}
\colorbox{red}{Black text on red background}
\par\colorbox{Light}{%
  \textcolor{Dark}{Light background}}
\par\colorbox{Dark}{%
  \textcolor{white}{Dark background}}
```

9-2-5

`\fcolorbox{frame color}{background color}{text}`

The `\fcolorbox` commands additionally puts a frame (in the color specified) around the colored box (this example is also in Color Plate XIV):



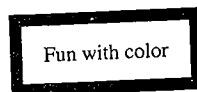
```
\usepackage {color}
\fcolorbox{red}{blue}{Black text,
  blue background, red frame}

\fcolorbox{red}{blue}{\color{white}%
  White text, blue background, red frame}
```

9-2-6

Some further examples (also in Color Plate XIV) show how to control the exact form of the box with the `\fbox` parameters `\fboxrule` and `\fboxsep`, which specify the thickness of the rule and the size of the shaded area.

Fun with color



```
\usepackage {color}
\setlength{\fboxrule}{6pt}%
\setlength{\fboxsep}{10pt}%
\colorbox{yellow}{Fun with color}\quad
\fcolorbox{red}{yellow}{Fun with color}
\par\bigskip\par
\setlength{\fboxrule}{1pt}%
\colorbox{green}{Fun with color}\quad
\fcolorbox{red}{green}{Fun with color}
```

9-2-7

Combining the use of PostScript fonts and color, you can construct lists with colorful elements (also printed in Color Plate XIV); the `\ding` command comes from the `pifont` package described in Section 10.1.1:

```
\usepackage{pifont,color}
\newenvironment{coldinglist}[2]
  {\begin{list}{\textcolor{#2}{\ding{#1}}}{}}
  {\end{list}}

\begin{coldinglist}{113}{red}
  \item Farallon's \textcolor{blue}{Replica}
  \begin{coldinglist}{42}{green}
    \item runs only on Windows;
    \item transforms PostScript type 1 fonts
      into bitmaps;
    \item well integrated in the environment.
  \end{coldinglist}
  \item \textcolor{blue}{Common Ground} of No
    Hands Software
  ....
```

□ Farallon's Replica

- runs only on Windows;
- transforms PostScript type 1 fonts into bitmaps;
- well integrated in the environment.

□ Common Ground from No Hands Software

- Mac and Windows;
- limited to four fixed resolutions;
- well integrated in the environment.

□ Adobe's Acrobat

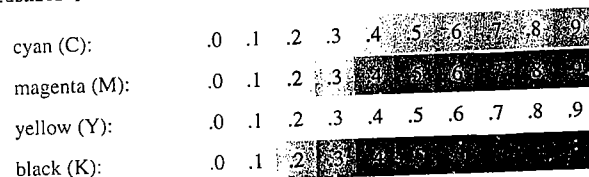
- Mac, Windows and Unix (Sun, HP, Irix);
- handles any PostScript document with Distiller;
- includes free Reader.

More complicated color support can be obtained in the framework of the `colortbl` package, which allows you to produce colored tables (see Section 9.3) or the seminar class, which makes color slides (see Section 9.4).

9.2.5 Calculating colors

Using the facilities of the color package, you can easily generate shades of a color by varying the cyan, magenta, yellow, and black components between 0 and 90%. This example uses the `\whiledo` control construct defined in the `ifthen` package.

```
\usepackage{ifthen,color}
\newcounter{Colr}
\setlength{\fboxsep}{2mm}
\begin{flushleft}
\newcommand{\CBox}[1]{\colorbox[cmk]{. #1,0.,0.,0.}{. #1}}
\makebox[30mm][l]{cyan (C):}%
\whiledo{\value{Colr}<10}{\CBox{\theColr}\stepcounter{Colr}}\
\renewcommand{\CBox}[1]{\colorbox[cmk]{0.,. #1,0.,0.}{. #1}}
\setcounter{Colr}{0}\makebox[30mm][l]{magenta (M):}%
\whiledo{\value{Colr}<10}{\CBox{\theColr}\stepcounter{Colr}}\
\renewcommand{\CBox}[1]{\colorbox[cmk]{0.,0.,. #1,0.}{. #1}}
\setcounter{Colr}{0}\makebox[30mm][l]{yellow (Y):}%
\whiledo{\value{Colr}<10}{\CBox{\theColr}\stepcounter{Colr}}\
\renewcommand{\CBox}[1]{\colorbox[cmk]{0.,0.,0.,. #1}{. #1}}
\setcounter{Colr}{0}\makebox[30mm][l]{black (K):}%
\whiledo{\value{Colr}<10}{\CBox{\theColr}\stepcounter{Colr}}
\end{flushleft}
```



The technique is well suited for working with the "gray" color model, which can be used to good effect on ordinary laser printers. The following picture shows an automatically generated data display in which the value for each grid cell is printed and the cell is shaded with the appropriate shade of gray. The data values for the shade are percentages, but are transformed to the 0–1 range of the gray color model simply by inserting "0" in front (luckily there are no 100% values). The number is set in black or white depending on the shade of background gray.

```
\usepackage{ifthen,color}

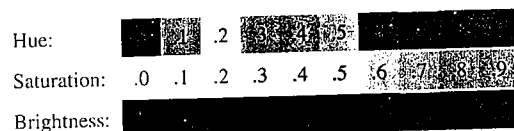
\newcommand{\sq}[2]{%
\definecolor{thiscol}{gray}{. #2}%
\ifthenelse{#2<50}%
{\color{white}}%
{\color{black}}%
}
```

```
\colorbox{thiscol}{\makebox[2em]{#1}}
}
\setlength{\tabcolsep}{0pt}
\begin{tabular}{*{10}{c}}
\sq{102}{85}&\sq{10}{98}&\sq{1}{1}{84}&
\sq{112}{53}&\sq{114}{83}&\sq{115}{82}&
\sq{119}{32}&\sq{11}{98}&\sq{111}{82}&
\sq{128}{31}&\sq{128}{31}&\sq{12}{98}&
\end{tabular}
```

102	10	110	112	114	118	119	11	122	128
12	130	135	138	14	141	142	144	14	159
15	160	162	167	16	175	176	17	181	187
193	197	199	19	1	20	215	22	23	24
24	25	27	28	29	29	2	32	33	
34	35	39	3	40	41	433	44	45	474
47	48	506	50	53	55	57	60	618	62
634	636	64	67	68	693	6	73	74	78
79	7	80	81	85	8	92	93	98	9

Since the L^AT_EX color package itself is largely independent of specific color models, you can use any of the color models supported by a given driver. For instance, with `dvips`, which generates PostScript output, you can specify colors using the HSB (Hue, Saturation, Brightness) model, which is well suited for graphics artists, since colors are given by one number (the hue), and saturation and brightness can also easily be varied, independently of the color. This model is also well suited to color calculations. In the following example (also printed in Color Plate V III) we vary each of the three HSB parameters in turn, in ten steps of 0 to 1; again, it uses the `\whiledo` control construct defined in the `ifthen` package.

```
\usepackage{ifthen,color}
\begin{flushleft}
\newcounter{Col}
\setlength{\fboxsep}{2mm}
\newcommand{\ColBox}[1]{\colorbox[hsb]{. #1,1.,1.}{. #1}}%
\setcounter{Col}{0}\makebox[18mm][l]{Hue:}%
\whiledo{\value{Col}<10}{\ColBox{\theCol}\stepcounter{Col}}\
\renewcommand{\ColBox}[1]{\colorbox[hsb]{.1,. #1,1.}{. #1}}%
\setcounter{Col}{0}\makebox[18mm][l]{Saturation:}%
\whiledo{\value{Col}<10}{\ColBox{\theCol}\stepcounter{Col}}\
\renewcommand{\ColBox}[1]{\colorbox[hsb]{1.,1.,. #1}{. #1}}%
\setcounter{Col}{0}\makebox[18mm][l]{Brightness:}%
\whiledo{\value{Col}<10}{\ColBox{\theCol}\stepcounter{Col}}\
\end{flushleft}
```



9-2-9

9.3 Coloring tables

Adding color to tables is not as straightforward as one might like. The basic problem is that color support for lines and cells cannot be provided at the TeX level but must be added in the code of the various packages. Table packages are complex and varied and the standard ones do not directly provide color. L^AT_EX's color package can provide a standard interface to the color facilities of drivers, but a good deal of work must still be done on the internal commands of the table-building environment (tabular or array).

Timothy van Zandt's colortab package (1993c) provides an interface to plain TeX's table-building primitive \halign. It handles coloring of both cells and rules but since it uses PSTricks to do its job it works only if your PostScript driver is supported by PSTricks. The colortab package also provides some functionality for coloring cells with L^AT_EX's array and longtable environments, but the syntax is somewhat awkward and does not work with all table packages. We therefore describe a L^AT_EX-specific package, colortbl.

9.3.1 The colortbl package

colortbl was written by David Carlisle to provide basic facilities for coloring table cells and lines; it can be used with other L^AT_EX table add-on packages like longtable, dcolumn, and hhline,¹ and loads the standard color and array packages to provide basic facilities.

The basic command of the package is:

```
\columncolor [color model] {color} [left overhang] [right overhang]
```

This can *only* be used in the argument of a > column specifier, either in a table preamble or the column specification of a \multicolumn. It provides a background color for that column or cell. The *color model* and *color* specifications are just as in normal use of the color package. The "overhang" arguments determine how far the color spreads beyond the column contents; they default to the values of \tabcolsep (in tabular) or \arraycolsep (in array), i.e., the color fills the entire column.

¹ You can also use colortbl with tabular*, but see the documentation for examples of border conditions where this does not yield the expected results.

As our first example we show a multicolor table with interesting proposals for a few days in Australia, also printed as Color Plate XV:

```
\usepackage {colortbl}
\setlength{\extrarowheight}{2mm}
\setlength{\tabcolsep}{2mm}
\begin{tabular}{|l|l|}
>{\columncolor{yellow}}c|c|>{\columncolor{yellow}}c|c|
>{\columncolor{red}}\bfseries c<{\textsc{GBP}}|}
\hline
\multicolumn{3}{>{\columncolor{red}}l}{\color{white}\textsf{LONDON}}
&\multicolumn{3}{>{\columncolor{red}}r}{\color{white}\textsf{Price}}
\\[1pt]
\hline
Sydney & OG4G & Thu Oct 10 & Mon Oct 21 or 28 & 11 or 18 days & 999\\
& & Thu Oct 17 & Mon Oct 21 or 28 & 4 or 11 days & 999\\
& OG7A & Sun Oct 13 & Mon Oct 21 or 28 & 8 or 15 days & 999\\
& & Sun Oct 20 & Mon Oct 28 & 8 days & 999\\
\hline
\end{tabular}
```

LONDON					Price
Sydney	OG4G	Thu Oct 10	Mon Oct 21 or 28	11 or 18 days	999
		Thu Oct 17	Mon Oct 21 or 28	4 or 11 days	999
	OG7A	Sun Oct 13	Mon Oct 21 or 28	8 or 15 days	999
		Sun Oct 20	Mon Oct 28	8 days	999

9-3-1

Contrast this with the following simpler version, in which the \newcolumntype command defines a column with gray background and the colors cover only the text in the cells, not the white space on either side:

```
\usepackage {colortbl}
\setlength{\extrarowheight}{2mm}
\setlength{\tabcolsep}{2mm}
\newcolumntype{G}{>{\columncolor[gray]{0.8}[0pt][0pt]}c}

\begin{tabular}{|l|G|G|G|G|G|c|}
\hline
\multicolumn{6}{>{\columncolor[gray]{0.5}}l}{\textsf{LONDON}}
&\multicolumn{6}{>{\columncolor[gray]{0.5}}r}{\textsf{Price}}
\\[1pt]
\hline
Sydney & OG4G & Thu Oct 10 & Mon Oct 21 or 28 & 11 or 18 days & 999\\
& & Thu Oct 17 & Mon Oct 21 or 28 & 4 or 11 days & 999\\
& OG7A & Sun Oct 13 & Mon Oct 21 or 28 & 8 or 15 days & 999\\
& & Sun Oct 20 & Mon Oct 28 & 8 days & 999\\
\hline
\end{tabular}
```

```

& &Thu Oct 17 &Mon Oct 21 or 28 & 4 or 11 days &999\\
& OG7A &Sun Oct 13 &Mon Oct 21 or 28 & 8 or 15 days &999\\
& &Sun Oct 20 &Mon Oct 28 & 8 days &999\\
\hline
\end{tabular}

```

LONDON					Price
Sydney	OG4G	Thu Oct 10	Mon Oct 21 or 28	11 or 18 days	999
		Thu Oct 17	Mon Oct 21 or 28	4 or 11 days	999
	OG7A	Sun Oct 13	Mon Oct 21 or 28	8 or 15 days	999
		Sun Oct 20	Mon Oct 28	8 days	999

9-3-2

This system is fine for vertically colored tables; but how do we handle tables such as Table A.3 on p. 474 and Table A.4 on p. 482, where rows have alternate colors? Using `\multicolumn` would be very clumsy, so `colortbl` has an extra command:

```
\rowcolor[color model]{color}[left overhang][right overhang]
```

This can only be used at the start of a row; it overrides any current `\columncolor` specifications, but its scope is limited to the current row:

```

\usepackage {colortbl}
\setlength{\extrarowheight}{2mm}
\begin{tabular}{|l|c|c|c|c|c|c|}
\hline
Sydney & OG4G & Thu Oct 10 & Mon Oct 21 or 28 & 11 or 18 days & 999\\
\rowcolor[gray]{0.5}
& & Thu Oct 17 & Mon Oct 21 or 28 & 4 or 11 days & 999\\
& OG7A & Sun Oct 13 & Mon Oct 21 or 28 & 8 or 15 days & 999\\
\rowcolor[gray]{0.5}
& & Sun Oct 20 & Mon Oct 28 & 8 days & 999\\
\hline
\end{tabular}

```

Sydney	OG4G	Thu Oct 10	Mon Oct 21 or 28	11 or 18 days	999
		Thu Oct 17	Mon Oct 21 or 28	4 or 11 days	999
	OG7A	Sun Oct 13	Mon Oct 21 or 28	8 or 15 days	999
		Sun Oct 20	Mon Oct 28	8 days	999

9-3-3

To color the *lines* in a tabular environment, including `\vline`, `\cline`, and `\hline`, the `colortbl` package provides the declaration

```
\arrayrulecolor[color model]{color}
```

which has arguments like `\color` and colors all subsequent horizontal and vertical rules in tables. It can be set outside the table, at the start of a row, or in a `>` specification in a table preamble. Its effects are *not* limited to the current scope, but do not override specifications in the table preamble. Thus in the following table (also printed in Color Plate XVI) the vertical rules are set to green at the start and are unaffected by the subsequent `\arrayrulecolor` commands:

```

\usepackage {colortbl}
\setlength{\arrayrulewidth}{2pt}
\arrayrulecolor{green}
\begin{tabular}{|l|c|c|}
\arrayrulecolor{black}\hline
United Kingdom & London & Thames\\
\arrayrulecolor{blue}\hline
France & Paris & Seine \\
\arrayrulecolor{black}\cline{1-1}
\arrayrulecolor{red}\cline{2-3}
Russia & Moscow & Moskva \\
\hline
\end{tabular}

```

United Kingdom	London	Thames
France	Paris	Seine
Russia	Moscow	Moskva

9-3-4

There is one problem: if colored rules are combined with colored cells, partial `\cline` rules are covered by the cell shading and thus are invisible. If necessary, you can use the more complicated `hhline` package with `colortbl` to achieve the desired effect.

Only one more area in tables can be colored—the gap between rules created by using `||` in a table preamble or `\hline\hline` in a table body. This can be set with

```
\doublerulesepcolor[color model]{color}
```

which works like `\arrayrulecolor`. Note, however, that when you color the space between two `\hline` commands in the `longtable` package, $\text{\LaTeX}$ no longer allows a page break at this point. This is one of the few situations in which adding color to your document may produce different page breaks.

9.3.2 Examples

Here we show how to color elements of tables by using various coloring commands described above. Support of color with $\text{\LaTeX}$ is not yet optimal, so that some complex constructs must sometimes be used to obtain a particular effect. All these tables are reproduced in color in Color Plates XVII to XIX.

We invite our readers to study *Color for the Electronic Age* (White, 1990), an excellent book on many aspects of color, for ideas on how colors can add value to tabular material. In the chapter on charts and graphs Jan White shows how to introduce variety in table presentation by using a single color in different ways. In the following variations, we implement some of his suggestions.

Let us start by looking at the generic table that will be the basis of our discussion (Color Plate XVIIa):

Table title

Description	Column 1	Column 2
Row one	mmmmmmmm	mmmmmmmm
Row two	mmmmmmmm	mmmmmmmm
Row three	mmmmmmmm	mmmmmmmm
Row four	mmmmmmmm	mmmmmmmm
Totals	mmmmmmmm	mmmmmmmm

```
\usepackage {colortbl}
\begin{tabular}{lrr}
\large\textbf{Table title}\\[2mm]
\textbf{Description}
& \textbf{Column 1}
& \textbf{Column 2}\\[1mm]
Row one & mmmmmmm & mmmmmmm \\
Row two & mmmmmmm & mmmmmmm \\
Row three& mmmmmmm & mmmmmmm \\
Row four & mmmmmmm & mmmmmmm \\
\cline{2-3}
Totals & mmmmmmm & mmmmmmm
\end{tabular}
```

9-3-5

First, let us put the table completely inside a colored box—orange, say, by defining that color with `\definecolor` and then using the simple `\colorbox` command (see Color Plate XVIIb):

```
\usepackage {colortbl}
\definecolor{orange}{cmyk}{0,0.61,0.87,0}
\colorbox{orange}{%
\begin{tabular}{lrr}
\large\textbf{Table title}\\[2mm]
\textbf{Description}
& \textbf{Column 1}
& \textbf{Column 2}\\[1mm]
Row one & mmmmmmm & mmmmmmm \\
Row two & mmmmmmm & mmmmmmm \\
Row three& mmmmmmm & mmmmmmm \\
Row four & mmmmmmm & mmmmmmm \\
\cline{2-3}
Totals & mmmmmmm & mmmmmmm
\end{tabular}%
}
```

9-3-6

Not only does this not add much information to the table, but you may also feel that the text is too dark to be easily readable. We can therefore print the text in white (Color Plate XVIIc):

9-3-7

```
\usepackage {colortbl}
\definecolor{orange}{cmyk}{0,0.61,0.87,0}
\colorbox{orange}{%
\color{white}%
\begin{tabular}{lrr}
\large\textbf{Table title}\\[2mm]
\textbf{Description}
& \textbf{Column 1}
& \textbf{Column 2}\\[1mm]
Row one & mmmmmmm & mmmmmmm \\
Row two & mmmmmmm & mmmmmmm \\
Row three& mmmmmmm & mmmmmmm \\
Row four & mmmmmmm & mmmmmmm \\
\cline{2-3}
Totals & mmmmmmm & mmmmmmm
\end{tabular}%
}
```

This does not, of course, affect the color of the horizontal rule drawn with `\cline`.

Depending on the circumstances, you might or might not consider this an improvement; if you do want to set light text like this on a dark background then it is usually better to render the text in a heavier font, or the same font in bold (Color Plate XVIIId):

9-3-8

```
\usepackage {colortbl}
\definecolor{orange}{cmyk}{0,0.61,0.87,0}
\colorbox{orange}{%
\color{white}\bfseries
\begin{tabular}{lrr}
\large\textbf{Table title}\\[2mm]
\textbf{Description}
& \textbf{Column 1}
& \textbf{Column 2}\\[1mm]
Row one & mmmmmmm & mmmmmmm \\
Row two & mmmmmmm & mmmmmmm \\
Row three& mmmmmmm & mmmmmmm \\
Row four & mmmmmmm & mmmmmmm \\
\cline{2-3}
Totals & mmmmmmm & mmmmmmm
\end{tabular}%
}
```

So far, however, no elements of the table are highlighted, so the color is only an embellishment, not a functional tool. Let us therefore see how by coloring some elements we can focus the reader's attention (Color Plate XVIIIa):

Table title

Description	Column 1	Column 2
Row one	mmmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmmm	mmmmmm
Row four	mmmmmm	mmmm
Totals	mmmmmm	mmmmmm

```
\usepackage {colortbl}
\begin{tabular}{lrr}
\color{green}
\large\textbf{Table title}\\[3mm]
\textbf{Description}
& \textbf{Column 1}
& \textbf{Column 2}\\[1mm]
Row one & mmmmm & mmm \\
Row two & mmm & mm \\
Row three& mmmmm & mmmmm \\
Row four & mmmmm & mmm \\
\cline{2-3}
\color{green} Totals
& \color{green} mmmmm
& \color{green} mmmmm
\end{tabular}
```

9-3-9

The title and bottom line have been set in green for emphasis. We can also invert the procedure by coloring the text you do *not* want to stress, so bringing out the black text elements (Color Plate XVIIIb):

Table title

Description	Column 1	Column 2
Row one	mmmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmmm	mmmmmm
Row four	mmmmmm	mmmm
Totals	mmmmmm	mmmmmm

```
\usepackage {colortbl}
{\color{green}%
\begin{tabular}{lrr}
\color{black}
\large\textbf{Table title}\\[3mm]
\textbf{Description}
& \textbf{Column 1}
& \textbf{Column 2}\\[1mm]
Row one & mmmmm & mmm \\
Row two & mmm & mm \\
Row three& mmmmm & mmmmm \\
Row four & mmmmm & mmm \\
\arrayrulecolor{black}\cline{2-3}
\color{black} Totals
& \color{black} mmmmm
& \color{black} mmmmm
\end{tabular}%
}
```

9-3-10

Note the `\arrayrulecolor` command that colors in black the horizontal line separating the body from the last line.

To draw attention to individual rows of a table, one can put a band of color behind them (Color Plate XVIIIc):

Table title

Description	Column 1	Column 2
Row one	mmmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmmm	mmmmmm
Row four	mmmmmm	mmmm
Totals	mmmmmm	mmmmmm

9-3-11

```
\usepackage {colortbl}
\newcommand\panel[1]{\multicolumn{1}%
{>\columncolor{magenta}}#1}}
\begin{tabular}{lrr}
\large\textbf{Table title}\\[2mm]
\textbf{Description}
& \textbf{Column 1}
& \textbf{Column 2}\\[1mm]
Row one & mmmmm & mmm \\
Row two & mmm & mm \\
\panel{1}{Row three}
& \panel{r}{mmmmmm}
& \panel{r}{mmmmmm} \\
Row four& mmmmm & mmm \\
\cline{2-3}
Totals & mmmmm & mmmmm
\end{tabular}
```

But we can do even better: color the whole table, and leave the row to be emphasized with a white background (Color Plate XVIIId):

Description	Column 1	Column 2
Row one	mmmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmmm	mmmmmm
Row four	mmmmmm	mmmm
Totals	mmmmmm	mmmmmm

9-3-12

```
\usepackage {colortbl}
\newcommand\panel[1]{\multicolumn{1}%
{>\columncolor{white}}#1}}
\colorbox{magenta}{%
\arrayrulecolor{black}
\begin{tabular}{lrr}
\large\textbf{Table title}\\[2mm]
\textbf{Description}
& \textbf{Column 1}
& \textbf{Column 2}\\[1mm]
Row one & mmmmm & mmm \\
Row two & mmm & mm \\
\panel{1}{Row three}
& \panel{r}{mmmmmm}
& \panel{r}{mmmmmm} \\
Row four& mmmmm & mmm \\
\cline{2-3}
Totals & mmmmm & mmmmm
\end{tabular}}
\end{tabular}}
```

This is completely analogous to the previous example except that the `\columncolor` command now use the color white, while the `\colorbox` at the beginning makes the whole table magenta.

Now we look at ways to highlight columns rather than rows. We use the `\columncolor` command to specify the color of the columns (Color Plate XVIIIe):

Table title

Description	Column 1	Column 2
Row one	mmmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmmm	mmmmmm
Row four	mmmmmm	mmmm
Totals	mmmmmm	mmmmmm

```
\usepackage {colortbl}
\definecolor{Bluec}{cmyk}{.60,0,0,0}
\begin{tabular}{l>{\columncolor{Bluec}}rr}
\large\textbf{Table title}\\[2mm]
\textbf{Description} & \textbf{Column 1}
& \textbf{Column 2} \\[1mm]
Row one & mmmmm & mmm \\
Row two & mmm & mmm \\
Row three& mmmmm & mmmmm \\
Row four & mmmmm & mmm \\
Totals & mmmmm & mmmmm
\end{tabular}
```

9-3-13

Colored panels of this type are often used to highlight connected regions in a table. The blue shade (Bluec) is defined at the beginning with the standard `\definecolor` command, although we could also have combined it with `\arrayrulecolor` as

```
\arrayrulecolor{cmyk}{.60,0,0,0}
```

Another feature often encountered in color work is the color gradient (Color Plate XVIIIIf):

Description	Column 1	Column 2
Row one	mmmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmmm	mmmmmm
Row four	mmmmmm	mmmm
Totals	mmmmmm	mmmmmm

```
\usepackage {colortbl}
\definecolor{Bluea}{cmyk}{.20,0,0,0}
\definecolor{Blueb}{cmyk}{.40,0,0,0}
\definecolor{Bluec}{cmyk}{.60,0,0,0}
\definecolor{Bluee}{cmyk}{.80,0,0,0}
\definecolor{Bluee}{cmyk}{1.0,0,0,0}
\begin{tabular}{l>{\columncolor{Bluee}}l%
>{\columncolor{Bluee}}r>{\columncolor{Bluee}}r}
\multicolumn{3}{\large\textbf{Table title}}\\[2mm]
\textbf{Description} & \textbf{Column 1}
& \textbf{Column 2} \\[1mm]
\rowcolor{Bluea}Row one & mmmmm & mmm \\
\rowcolor{Blueb}Row two & mmm & mmm \\
\rowcolor{Bluec}Row three& mmmmm & mmmmm \\
\rowcolor{Bluee}Row four & mmmmm & mmm \\
\rowcolor{Bluee}Totals & mmmmm & mmmmm
\end{tabular}
```

9-3-14

Here we use various levels of blue (Bluea through Bluee) defined at the start for successive rows. Although the mechanics look a bit frightening at first sight, the result can be

quite pleasing, and this technique is certainly one of those most often used to produce attractive and easily readable tabular material.

To emphasize the titles and headings, we can color fields we consider less important and to further guide the eye we can add some vertical and horizontal lines (Color Plate XIXa):

Table title

Description	Column 1	Column 2
Row one	mmmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmmm	mmmmmm
Row four	mmmmmm	mmmm
Totals	mmmmmm	mmmmmm

9-3-15

```
\usepackage {colortbl}
\newcommand{\panel}[1]{\multicolumn{1}
{>{\columncolor{cmyk}{.40,0,0,0}}r}{#1}}
\begin{tabular}{l|l|r}
\multicolumn{1}{l}{}
{\large\textbf{Table title}}\\[2mm]
\textbf{Description} & \textbf{Column 1}
& \textbf{Column 2} \\[1mm]\hline
Row one & \panel{mmmmmm} & \panel{mmmm} \\
Row two & \panel{mmmm} & \panel{mmm} \\
Row three& \panel{mmmmmm} & \panel{mmmmmm} \\
Row four & \panel{mmmmmm} & \panel{mmmm} \\
Totals & mmmmm & mmmmm
\end{tabular}
```

As already remarked, we can also color the region containing the important text and leave the less important text region white. As we want the surrounding region to extend a bit beyond the table limits, we reset the `\fboxsep` width to 3mm, and choose a lightish blue tint as the background color. The rest is similar to the previous example. It is interesting to compare the examples to see which is more efficient in getting across the message (Color Plate XIXb):

Table title

Description	Column 1	Column 2
Row one	mmmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmmm	mmmmmm
Row four	mmmmmm	mmmm
Totals	mmmmmm	mmmmmm

9-3-16

```
\usepackage {colortbl}
\newcommand{\panel}[1]{%
\multicolumn{1}{>{\columncolor{white}}r}{#1}}
\setlength{\fboxsep}{3mm}
\colorbox{cmyk}{.40,0,0,0}{%
\begin{tabular}{l|l|r}
\multicolumn{1}{l}{}
{\large\textbf{Table title}}\\[2mm]
\textbf{Description} & \textbf{Column 1}
& \textbf{Column 2} \\[1mm]\hline
Row one & \panel{mmmmmm} & \panel{mmmm} \\
Row two & \panel{mmmm} & \panel{mmm} \\
Row three& \panel{mmmmmm} & \panel{mmmmmm} \\
Row four & \panel{mmmmmm} & \panel{mmmm} \\
Totals & mmmmm & mmmmm
\end{tabular}}
```

One need not always color the column cells themselves; coloring the rules inside the table can be enough to highlight the relevant information. We can do this with `colortbl` facilities. First we replace the default black rules with colored ones; this is done with the

`\arrayrulecolor` command and affects the vertical rules defined in the table preamble. After the preamble we switch to a slightly different blue for all horizontal rules, but the right-most part of the last rule is switched back to black with a new `\arrayrulecolor` declaration before the `\cline` (Color Plate XIXc):

Table title

Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

```
\usepackage {colortbl}
\arrayrulecolor{cmyk}{.80,0,0,0}
\begin{tabular}{l|l|l}
\multicolumn{3}{l}{\large\textbf{Table title}}\\[2mm]
\textbf{Description} & \textbf{Column 1}
& \textbf{Column 2} \\[1mm]\hline
\arrayrulecolor{cmyk}{1.0,0,0,0}
Row one & mmmm & mmm \\ \hline
Row two & mmm & mm \\ \hline
Row three& mmm & mmm \\ \hline
Row four & mmm & mm \\ \cline{1-1}
\arrayrulecolor{black}\cline{2-3}
Totals & mmm & mmm
\end{tabular}
```

9-3-17

Alternatively we can go back to a blue colored background for some rows, and draw only a few of the rules to guide the reader through the information (Color Plate XIXd):

Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

```
\usepackage {colortbl}
\arrayrulecolor{black}
\begin{tabular}{l|l|l}
\multicolumn{3}{l}{\large\textbf{Table title}}\\[2mm]
\rowcolor{cmyk}{.20,0,0,0}
\textbf{Description} & \textbf{Column 1}
& \textbf{Column 2} \\[1mm]\hline
Row one & mmm & mm \\
Row two & mmm & mm \\
Row three& mmm & mm \\
Row four & mmm & mm \\ \cline{2-3}
Totals & mmm & mm
\end{tabular}
```

9-3-18

To construct a more complicated arrangement that subdivides the table into different regions, we can use a combination of row colors; as we noted on p. 329, the standard method for getting partial horizontal rules does not work correctly with `colortbl`, so `hhline` is used in the following example (Color Plate XIXe):

```
\usepackage {colortbl, hhline}
\definecolor{Blueb}{cmyk}{.40,0,0,0} \definecolor{Blue}{cmyk}{.80,0,0,0}
\arrayrulecolor{white}
\begin{tabular}{>\columncolor{Blueb}}1>\columncolor{Blue}}r|l%
>\columncolor{Blue}}r}
```

```
\multicolumn{3}{>\columncolor{Blueb}}1>\large\textbf{Table title}}\\[2mm]
\rowcolor{white}
\textbf{Description} & \textbf{Column 1} & \textbf{Column 2} \\[1mm]
\arrayrulecolor{black}
\rowcolor{Blueb}
Row one & mmm & mm \\ \hhline{--}
Row two & mmm & mm \\ \hhline{--}
Row three& mmm & mm \\ \hhline{--}
Row four & mmm & mm \\ \hhline{--}
\rowcolor{white} Totals & mmm & mm
\end{tabular}
```

Table title

Description	Column 1	Column 2
Row one	mmmm	mmmm
Row two	mmmm	mmmm
Row three	mmmm	mmmm
Row four	mmmm	mmmm
Totals	mmmm	mmmm

9-3-19

The interplay among various styles of typesetting titles, coloring rules, and the various shades of the background color gives you a rich palette of possibilities for presenting complex data. As a last effect possibility, we can consider altering the *width* of the rules; in the table below, a thick horizontal rule highlights the title and header lines. The quite complex code shows most of the commands used in this section. Note the supplementary columns in the table to extend the color a `\quad` to the left and right (Color Plate XIXf):

```
\usepackage {colortbl}
\definecolor{Blueb}{cmyk}{.40,0,0,0}
\definecolor{Blue}{cmyk}{.80,0,0,0}
\definecolor{Bluee}{cmyk}{1.0,0,0,0}
\arrayrulecolor{black}
\setlength\arrayrulewidth{1mm}
\begin{tabular}{l|l|l}
\quad&\multicolumn{3}{l}{\large\textbf{Table title}}&\quad\\[2mm]
\rowcolor{Blueb}&\textbf{Description}
&\textbf{Column 1}
&\textbf{Column 2} \\[2mm]
\hline
\rowcolor{Blueb}&Row one & mmm & mm \\
\rowcolor{Blueb}&Row two & mmm & mm \\
\rowcolor{Blueb}&Row three& mmm & mm \\
\rowcolor{Blueb}&Row four & mmm & mm \\
\cline{2-3}
\rowcolor{Bluee}&Totals & mmm & mm \\
\end{tabular}
```

Table title

Description	Column 1	Column 2
Row one	mmmm	mmmm
Row two	mmmm	mmmm
Row three	mmmm	mmmm
Row four	mmmm	mmmm
Totals	mmmm	mmmm

9-3-20

We hope these examples have shown how flexible the color commands are in creating almost any possible layout. The only caveat is that you should not get carried away by the wealth of possibilities. The purpose of color is to help the reader understand material better—don't let your use of color stand in the way of comprehension.

9.4 Color slides with L^AT_EX: the seminar class

Many L^AT_EX users want to take advantage of T_EX's high-quality typesetting to produce slides for a presentation. This facility was originally provided by a separate package, Sl_TE_X, but it had a number of disadvantages:

- it was limited to a set of specially scaled Computer Modern fonts and was difficult to adapt to other fonts;
- the user had to have two separate files, one for control information and the other for the actual slides;
- the control of color and overlays was limited and crude;
- there was only one “style” for slides, and writing a different layout (say, to put a logo on each slide) was not documented.

L^AT_EX 2_ε, released at the end of 1993, included a simple L^AT_EX class, slides, to emulate Sl_TE_X without the overhead of a separate macro package. However, better L^AT_EX packages have been available for some time, one of which is Jim Hafner's foiltex (Hafner, 1992). Here, however, we look in detail at Timothy van Zandt's seminar (van Zandt, 1993b); when used in conjunction with a PostScript printer and the PSTricks macros described in Chapter 4, seminar offers almost every facility imaginable, including:

- fancy frames, headers, and footers;
- landscape and portrait slides in the same document;
- colored text and tables;
- interleaving of annotations and slides;
- slide “chapters” and list of slides;
- overlays.

seminar is a normal L^AT_EX class that can be used with almost all other L^AT_EX packages (such as those to change font, include graphics, or add mathematical facilities, like A_MS-L^AT_EX). Its main job is to produce transparencies, but it can also generate accompanying notes from the same source file.

9.4.1 Using the seminar class

Usage is simple; begin your document in the normal way with seminar as the class name and then put whatever you want on a transparency inside a series of “slide” environments.

```
\begin{slide} ... \end{slide}
\begin{slide*} ... \end{slide*}
```

slide is for landscape slides and slide* is for portrait slides. The portrait option makes the document come out in portrait orientation, rather than the default landscape; commands to rotate either the portrait or landscape slides are inserted as appropriate to the option chosen. Because rotation is not directly supported by L^AT_EX, you have to work out the appropriate \special commands to make your driver turn pages through 90 degrees—seminar is not integrated with the standard L^AT_EX graphics package. For a PostScript driver like dvips, the rotation commands can be defined in the local control file described below (Section 9.4.6). If the driver cannot handle rotation at all, you may still be able to print landscape slides if you can persuade your printer itself to print in that mode.

The default output is shown in the following example:

```
\documentclass{seminar}
\usepackage{times}
\begin{document}
\begin{slide}
  My talk is about:
  \begin{description}
    \item[Cats] Nice furry creatures that
      should have a place in every good home
    \item[Dogs] Nasty barking things that bite you;
    \item[Snakes] They come slithering through the
      grass and \emph{have no feet}; this is most
      disturbing;
    \item[Rhinoceroses] \textbf{Never} be rude to
      a rhino; they are bigger than you, and meaner.
  \end{description}
\end{slide}
\end{document}
```

My talk is about:

Cats: Nice furry creatures which should have a place in every good home

Dogs: Nasty barking things which bite you;

Snakes: They come slithering through the grass and *have no feet*; this is most disturbing;

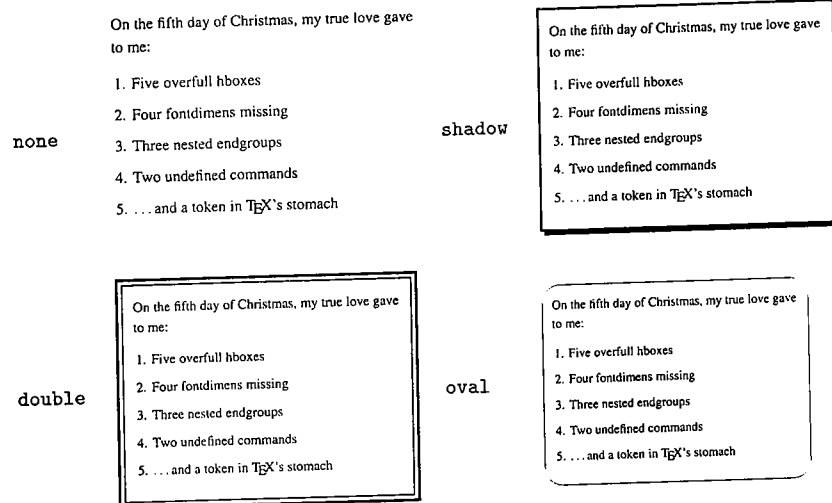
Rhinoceroses: **Never** be rude to a rhino; they are bigger than you, and meaner.

Most slides are no more complicated than that and can be produced using standard L^AT_EX environments like itemize, enumerate and tabular.

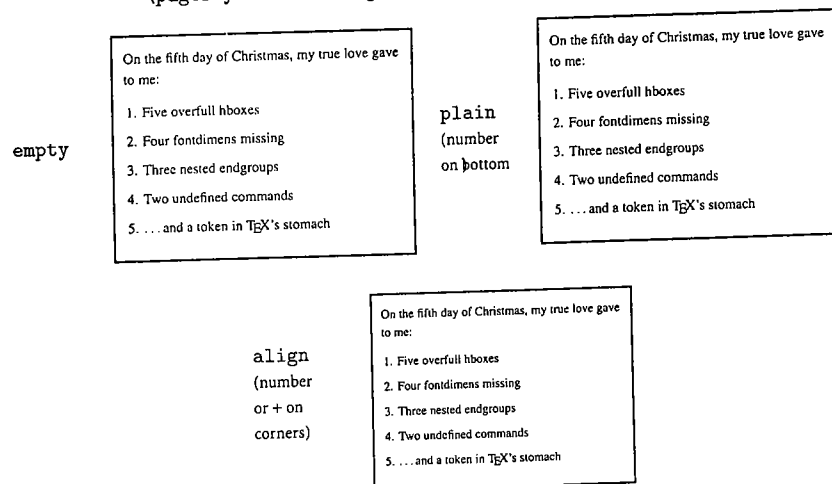
9.4.2 Frame styles

seminar distinguishes between the *pagestyle* and the *slideframe*. The former, as in normal L^AT_EX, describes the running head and foot text; the latter describes the box or frame of some kind around the text of the slide. A variety of slide-framing styles are available, set with the \slideframe command, e.g., \slideframe{shadow}. Below are some of the

predefined frame styles (some assume you have a PostScript printer and load the fancybox package for you):



Similarly, a variety of page styles (headers and footers) are available with the `\pagestyle` command, e.g., `\pagestyle{plain}`. The predefined ones are:



New slide frames styles and page styles can be defined; the examples in this chapter (e.g., Color Plate XX) were created for use at the European Particle Physics Laboratory (CERN).

9.4.3 Interleaving notes and selecting subsets

It is easy to intersperse your slides with notes to yourself; you can simply place them between the `slide` environments or enclosed in a specific `note` environment. You can use any $\text{\LaTeX}$ commands in these notes and include your whole article here if desired. A variety of class options can be used to print the slides:

`slidesonly` only the slides are printed;

`notesonly` only the notes are printed;

`notes` the slides are interleaved with the notes;

`article` the document notes are typeset as a normal $\text{\LaTeX}$ paper and the slides are put in as figures (reduced by 50%).

`\slideplacement{type}`

The `\slideplacement` command affects how slides are placed in the `article` format; the possible *types* are:

`float` slides are floated (the default);

`float*` slides are floated, but in two-column format they span both columns;

`here` slides are placed where they occur in the notes.

Further details on controlling the interaction between slides and notes are given in the *User's Manual* accompanying the seminar distribution.

Selected slides can be included or excluded with the `\onlyslides` or `\noteslides` commands, which take as a parameter a comma-separated list of slides; this list can contain numbers, ranges (e.g., 5–10) or $\text{\LaTeX}$ `\ref` commands referring to `\label` commands in the slides.

9.4.4 Controlling slide size, fonts, and magnification

The user can change any of the following either on a slide-by-slide basis or for the whole document:

- slide height and width;
- top, bottom, left, and right margins;
- text justification (default is ragged right);

- page breaking by varying tolerance of overrun material;
- interline spacing;
- point size, and choice of fonts.

Changing the default settings is explained in detail in the *User's Manual* (van Zandt, 1993b).

Because seminar works by magnifying pages, sophisticated users should read the manual to see how to set and change TeX dimensions. Most users need not worry about this—in commands like `\includegraphics` you should always express your “width” and “height” requests in fractions of the line size (`\linewidth` or `\columnwidth`) anyway.

9.4.5 Fonts

Because the font size on slides is larger than that in normal text (articles, manuals, etc.), some fonts (especially for mathematics) are often not available at the size needed. Programs like the dvips or emTeX drivers try to generate these fonts but do not always find the METAFONT input files necessary to do so. It makes sense, therefore, to use scaleable PostScript fonts; the complete Computer Modern family is available in PostScript Type1 format, either in the commercial set from Blue Sky or Y&Y or in the (slightly less polished) public-domain bakoma set by Basil Malyshev.

9.4.6 The local control file

The seminar package always starts by trying to find a file called `seminar.con` on the TeX search path; this gives the user or site a convenient opportunity to overwrite the defaults. The `seminar.con` file can contain any L^AT_EX commands, including inputting package files.

Color Plate XX was typeset using a `seminar.con` set up for CERN; let us examine its contents. First, we define a landscape macro for the dvips driver. Without this, by default the command `\printlandscape` simply warns the user that the file should be printed in landscape mode:

```
1 \renewcommand{\printlandscape}{\special{papersize=297mm,210mm}}
```

We intend to include graphics and use color; thus we need the following packages (using pstcol, which also loads PSTricks):

```
2 \RequirePackage[usenames]{pstcol}
3 \RequirePackage{graphicx}
```

As this control file is for a PostScript printer and we want to make use of PostScript effects like rounded box corners, we load some extra packages:

```
4 \RequirePackage{semcolor,fancybox}
```

9.4.6 The local control file

Since this setup is designed to be used in Europe, where A4 paper is standard, we need to set up some sizes and re-invoke the code to set internal parameters:

```
5 \setlength\paperwidth{210mm}
6 \setlength\paperheight{297mm}
7 \input{sem-page.sty}
```

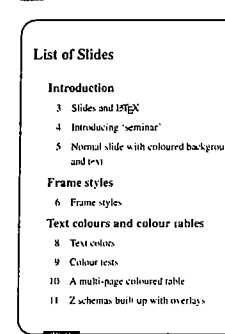
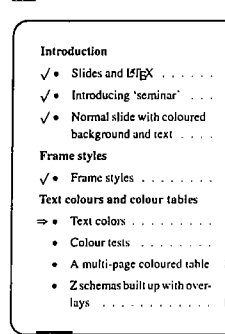
Next size and position the slide area; note that the margin sizes are commands, not L^AT_EX lengths, and so have to be set using `\renewcommand`.

```
8 \setlength\slidewidth{205mm}
9 \setlength\slideheight{150mm}
10 \renewcommand\slideleftmargin{.8in}
11 \renewcommand\sliderightmargin{.8in}
```

For slide “sections,” list of contents, etc. we use another package which comes with seminar:

```
12 \RequirePackage{slidesec}
```

The slidesec package gives us access to various internal commands, some of which are used below. We can, for instance, produce a list of slides, in two different styles:

<pre>\begin{slide*} \listofslides \end{slide*}</pre>		<pre>\begin{slide*} \Slidecontents \end{slide*}</pre>	
--	--	---	--

For slide titles, there is a predefined `\slideheading` command; we will amend this to be typeset with a “shadow” by redefining the command `\makeslideheading`:

```
13 \newsavebox\headbox
14 \renewcommand{\makeslideheading}[1]{%
15 \gdef\theslideheading{#1}%
```

```

16 \begin{lrbox}{\headbox}\large\bfseries#1\end{lrbox}%
17 \begin{center}\shadowbox{\usebox\headbox}\end{center}}

```

A `\slidechapter` command is also defined:

```

18 \newcounter{slchapter}
19 \newcommand\l@chapslide[1]{\slide@undottedcline{#1}{}}
20 \newcommand\slidechapter[1]{%
21   \begin{slide}%
22     \addtocontents{los}{\protect\l@chapslide{\protect\large\bfseries #1}}%
23   \end{slide}%
24   \renewcommand\thechapterheading{#1}%
25   \setcounter{slchapter}{0}}
26 \newcommand\thechapterheading{}

```

This allows the user to break the slides into groups; with the “cernsections” page style, the slide chapter title (stored in `\thechapterheading`) is printed in the bottom right corner.

Now we set up the CERN page styles using `\newpagestyle`; the plain “cern” style just places registration “+” marks, the date, and the page number.

```

27 \newpagestyle{cern}
28 {{\color{black}\small \textbf{+}\hfil \today \hfil\textbf{+}}}
29 {{\color{black}\small \textbf{+}\hfil \thepage \hfil\textbf{+}}}

```

The “cernsections” style has a logo (the PostScript file `cernlogo.eps`) in the top left, the slide title in the top center, and the slide number in the top right; in the bottom left we print the current date, and in the bottom right the current section heading:

```

30 \newpagestyle{cernsections}
31 {{\color{black}\small
32   \raisebox{-.5cm}{0cm}[0cm]{\includegraphics[height=.8cm]{cernlogo.eps}}}
33   \hfil\textbf{\theslideheading}\hfil\textbf{\thepage}}}
34 {{\small\color{black}\today \hfil\thechapterheading/\arabic{slchapter}}}

```

To rotate header texts properly for portrait slides, we also have to issue the following command:

```

35 \rotateheaderstrue

```

For the slide frames, we define a frame with the word “CERN” set in a colored box on the lower left edge of the normal slide box; this is done using PSTricks macros (automatically included by the `pstcol` package). The color commands are defined using the `usenames` option of the `pstcol` package; `\SlideFront` is the current foreground color (this is defined later on):

```

36 \newslideframe{cern}{\SlideFront
37   \boxput(-0.7,-1.11){%

```

```

38   \psframebox[linecolor=black,fillcolor=ForestGreen,fillstyle=solid]{%
39     \mbox{\normalsize\sffamily\color{black}CERN}}}{#1}%
40   \color{black}}

```

Finally, we make sure that each slide starts with the current foreground color and that the `slchapter` counter is incremented properly:

```

41 \renewcommand\everyslide{%
42   \SlideFront
43   \stepcounter{slchapter}%
44 }

```

We set up slides using the command `\SlideColors`, with two parameters, color names for foreground and background. A synonym is defined for black on white.

```

45 \newcommand\SlideColors[2]{\gdef\SlideFront{\color{#1}}%
46   \slideframe{\Frameddefault}%
47   \slideframe*[\psset{fillcolor=#2,fillstyle=solid}]{blueframe}}
48 \newcommand\blackandwhite{\SlideColors{black}{white}}

```

By default frame borders are colored using the PostScript `setgray` operator; to change it, we have to use an optional parameter to `\newslideframe`. For example, we can define a simple blue frame as follows:

```

49 \newslideframe{blueframe}
50 [\psset{linecolor=NavyBlue,linewidth=\slideframewidth,%
51   framesep=\slideframesep,cornersize=absolute,linearc=.5cm}]%
52 {\psframebox{#1}}

```

The slide defaults are for a detailed layout and the CERN logo, with yellow writing on a blue background:

```

53 \pagestyle{cernsections}
54 \newcommand\Frameddefault{cern}
55 \SlideColors{Yellow}{RoyalBlue}

```

This just colors the area inside the frame, so to color the entire slide you would need an additional `\pagecolor` command.

To allow further customization by the user, an extra file `seminar.cfg` is loaded if it can be found:

```

56 \InputIfFileExists{seminar.cfg}
57 {\typeout{Loading user configuration of seminar}}{ }

```

An example of using this setup is shown in Color Plate XX.

To change the default setup to black on white you can add this code in front of the command `\begin{document}`:

```
\documentstyle{seminar}
\renewcommand{\SlideFront}{\color{black}}
\begin{document}
\begin{slide}
...
```

If you want to keep the “SlideColors” interface so as to get color backgrounds for your slides, but want a plain border and no logos, etc. you can use the following setup:

```
\documentstyle{seminar}
\pagestyle{plain}
\renewcommand{\SlideColors}[2]{%
\renewcommand{\SlideFront}{\color{#1}}%
\slideframe%
[\psset{fillcolor=#2,fillstyle=solid}]%
{scplain}%
}
\begin{document}
\SlideColors{Red}{Yellow}
\begin{slide}
...
```

If a `seminar.con` such as this is part of a $\text{\LaTeX}$ setup, users who do not like the system can “roll their own” to various degrees of sophistication. In the simplest case they can just write a `seminar.con` file of their own in the directory which contains their slide files:

```
% A do-nothing file seminar.con
\endinput
```

The complete syntax of the many other commands at your disposal is described in the seminar *User Guide*; you should probably also read Chapter 4 on PSTricks!

9.5 Color in the printing industry and separation

The printing industry generally uses either the CMYK color model for full-color printing and spot colors for one or two-color work. The latter are normally specified according to the Munsell and Pantone charts; more recently, the Focoltone and Trumatch systems have become common for color matching.

PostScript is almost universally used for high quality typeset output, and Level 2 of the PostScript language offers full support for color, with not only the simple RGB, CMYK, and HSB models, but also the CIE system and various special color spaces. The details of these systems, and algorithms for converting between color models, are discussed exhaustively in

Adobe Systems (1990), pp. 176–199. There are very useful discussions of color in PostScript in Kunkel (1990) and McGilton and Campione (1992); Reid (1988) also has useful material. If you read the older books, you should note that full Level 2 PostScript provides a number of important new commands that considerably ease preparation of color separations.

Issues encountered when working with $\text{\LaTeX}$ in the professional printing industry are discussed in a useful article by Michael Sofka (Sofka, 1994). Duggan (1994) also treats color separation from a $\text{\LaTeX}$ perspective, while Sowa (1994) describes how to do separations at the level of bitmap output from $\text{\LaTeX}$.

The $\text{\LaTeX}$ user can be largely shielded from the complications of the actual printing. A document containing color material can be typeset and run through a driver like `dvips` to create a color PostScript document that can be previewed on screen or printed on a color printer. The main problem is that the color rendering and precision of the common inkjet printers is not good enough to prepare copy for professional printing; however, they are very well suited to transparencies, handouts, posters etc., and it is well worth the effort to develop your use of $\text{\LaTeX}$ color for this type of work.

9.5.1 Color separation

To produce a “real” book or journal using offset printing, the printer will require four versions of each page that contain, respectively, the gray levels corresponding to the proportions of cyan, magenta, yellow, and black. Each page is overprinted four times with each of the colors. Color Plate XXI, containing the five Olympic rings at the top and a multi-colored ellipse at the bottom, shows how adding the various color inks gives a colored picture its final form. We started by applying the cyan ink (top left), and then added the magenta (top right), yellow (bottom left), and finally the black inks (bottom right) to obtain the picture in full color. The four separate stages of the process, and the cumulative effect, are shown.

Color work is usually typeset on special film and to high tolerances, since each page is overprinted four times and registration must be exact. Some typesetting systems can produce the four separations automatically, but more commonly this is done by PostScript manipulation. Professional-quality typesetting packages and sophisticated graphics manipulation packages like Adobe Illustrator, Corel Draw and Adobe Photoshop handle color separation directly, while Adobe’s Separator program can work on arbitrary color PostScript. The effectiveness of programs like this depends on how color is specified in the PostScript file produced, and PostScript Level 2 provides a great deal of support for this.

When preparing material for separation, the user needs to understand some specific issues:

- custom or “spot” colors;
- knockout and overprint;
- trapping.

Print jobs commonly differentiate between work that can be handled with the CMYK model (process color) and work specifying a precise color (custom color). While process

color pages are printed four times with proportions of the appropriate color, pages with a custom color are printed separately, with the ink specifically requested (commonly from standard sets like Pantone). The L^AT_EX color package does not directly support spot color.

Let us consider the simple example of a drawing of a yellow circle on a blue background (like Color Plate V). It must be decided at some point whether to create this by getting the right values of C, M, Y and K² and then printing them superimposed on one another (this is called overprinting), or by leaving a circular hole in the blue background where the yellow circle will go (this is knockout). The CMYK system (and the default PostScript color model) assumes overprinting (since the whole system works on an additive principle), but custom colors naturally require a knockout, since they specify a precise color that should not be mixed with anything else. In PostScript, the `setoverprint` operator can be used to turn overprinting on or off; this could be used in more sophisticated color handling in a dvi-to-PostScript driver.

Whether we use knockout or overprinting, we will encounter another problem, registration of successive printings on a piece of paper. Even the most precise machinery has difficulty in aligning material that is specified in scaled points at a resolution of over 2500 dpi. In practice, therefore, careful color work uses *trapping*. Returning to the example of a yellow circle on a blue background with trapping, we either slightly decrease the size of the hole in the blue (a choke trap) or increase the size of the circle (a spread trap), and thus create a slight overlap to avoid registration problems.

It is beyond the scope of this book to deal with the full ramifications of high-quality color printing (such as the specification of screens), and we strongly suggest that readers take professional advice before embarking on large-scale full color projects with L^AT_EX. While Mike Sofka (1994) with a private dvi driver, and others using Textures on a Macintosh, have shown that L^AT_EX can do full-scale color work, users who need relatively simple color printing (such as a two-color book in which all headings are blue and all tables have a blue tinted background) can do very well without recourse to commercial software. CMYK color separations can be made with the color package and dvips by using only PostScript Level 1 operators; we describe this in the following section.

9.5.2 Color separation using L^AT_EX and dvips

The principle of attaining simple dvips separations from T_EX output is that each output page is produced four times, using the `-b 4` command-line switch, or `b 4` in a configuration file, and a header file that redefines the color operators for each of the four pages. The header file `colorsep.pro` (distributed with dvips, maintained by Sebastian Rahtz, and largely derived from Kunkel (1990) and Reid (1988); printed in Appendix A.4) uses the `bop-hook` handle (see Section 11.2.5 on p. 429) to increment a counter at the beginning of each page, thus checking whether a C, M, Y or K page is being produced. The `setcmykcolor` operator is then redefined to produce just one of the four colors, in gray, and RGB colors are converted to CMYK before going through the same process. The `setgray` operator is activated only on the black (K) page. Unfortunately, color bitmap images are not handled by this system.

²The normal printing order is in fact first yellow, then cyan, magenta, and finally black.

Cyan	Warning!	Yellow	Warning!
Magenta		black	Warning!

Figure 9.1: Example of simple color separations.

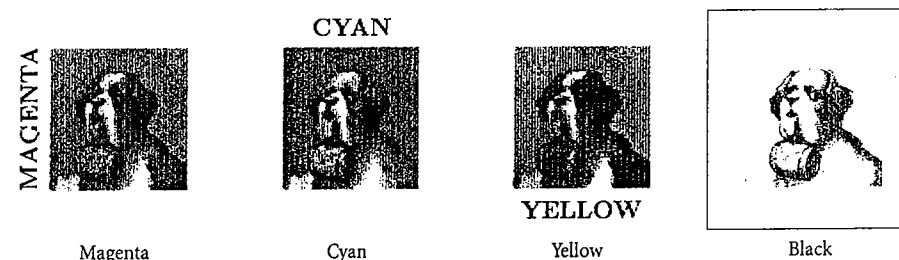


Figure 9.2: Color separation of a bitmap image using aurora.

The output from the separation for the example of `Warning!` is simulated in Fig. 9.1; here the box is set in "ForestGreen", whose CMYK value is "0.91 0 0.88 0.12". Notice that the "M" page is blank, as neither the green box nor the black text needs any magenta.

For greater control over the separations Graham Freeman's *aurora* package can also be used with dvips or any other dvi-to-PostScript program that supports the downloading of header files. This also works by redefining PostScript color commands, but the pages for each different C, M, Y, and K color are produced with separate header files. One main header file (`aurora.pro`) is downloaded together with one of `cyan.pro`, `magenta.pro`, `yellow.pro`, or `black.pro`. Thus the dvips command line to put the magenta portions of a file `text.dvi` into a PostScript file called `text-magenta.ps` would be

```
dvips text -h aurora.pro -h magenta.pro -o text-magenta.ps
```

Many forms of color bitmap images are handled by *aurora*, but not the full range of possibilities supported by the PostScript Level 2 `colorimage` command; specifically, compound color does not work. Figure 9.2 shows what the separations for an image look like using *aurora*. The familiar canine is shown with a gray shade for the quantities of cyan, magenta, yellow or black. The effect is even clearer in Fig. 9.3, a small portion of the same picture (part of the barrel at the dog's neck).

A custom color can be produced by precisely defining the C, M, Y, and K components and providing a matching header file for the color that specifies a point in CMYK space of the separation you want. If the current color in the document fits the color, then the object is printed. This method of selecting custom colors for output requires that the numbers

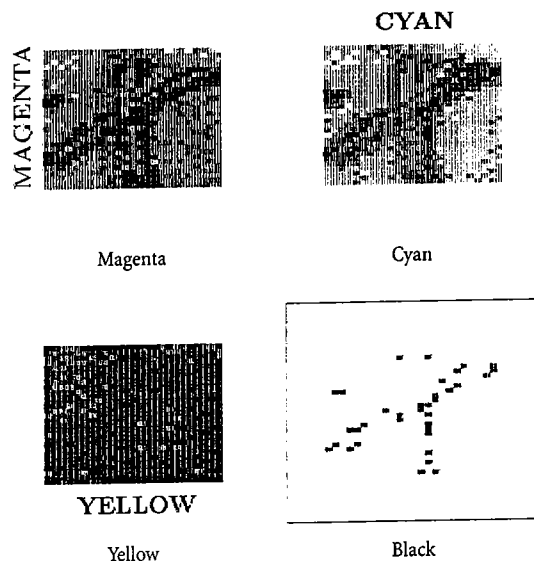


Figure 9.3: Detail of color separation of a bitmap image.

specifying the custom color precisely match the color used within the document. If the color does not match, no output is produced, while with the normal system the component of the current process color determines the *amount* of output to produce.

CHAPTER 10

Using PostScript fonts

In this chapter we look at the most basic type of graphical object in documents: the characters that form the words. Character shapes are not a direct part of the TeX system; all TeX wants to know about them is some metric information, such as their width or height. It is the task of the device driver to take the TeX output (the dvi file) and produce from it a graphical representation of the page. For this the driver needs information about the actual shapes of the characters. This information is stored in so-called fonts (collections of characters) for which many different storage formats exist. Thus in principle any existing font can be used with TeX provided that:

- the metric information TeX needs is available or can be generated;
- a driver program exists that can interpret dvi files and in addition understands the format in which the fonts are stored.

However, for a long time the number of fonts available in practice to TeX users was smaller than imposed by the above restrictions.

Donald Knuth developed a companion program to TeX, METAFONT, for generating fonts to be used with TeX (Chapter 3 looked briefly at METAFONT's drawing capabilities). For some time only those fonts designed using this program were available to TeX users, with the result that TeX or L^ATeX documents had an easily identified look and feel—mainly a result of the use of the Computer Modern fonts. In recent years designers have produced a few more fonts with METAFONT, but the number of METAFONT-generated fonts offered to the TeX community was and is very small compared to other typesetting systems which can access the literally thousands of fonts commercially available in the PostScript world.

While it was in fact possible to use PostScript fonts within TeX at an early stage, their integration into macro packages like L^ATeX was difficult, as font access was hard-wired into

the macro code and changing it thus required a deep understanding of the inner workings of $\TeX$. This situation changed in 1989 with the introduction of a new font selection scheme (NFSS), of which a later release (NFSS2) was integrated into the current $\LaTeX$ standard.

With this release of $\LaTeX$ it became possible to access the large set of PostScript fonts without difficulty—and, most importantly, without modifying $\LaTeX$ internal code and building private format files. While with $\LaTeX$ 2.09 producing portable documents required using Computer Modern Fonts, $\LaTeX$ documents nowadays can use other fonts without compromising portability, provided, of course, the fonts are available at both sites. And even if they are not, documents can still be processed, since NFSS automatically selects substitution fonts for anything missing.

With NFSS, there is no limit on the number of fonts that can be integrated into the $\LaTeX$ system, and once this is done they all can be used with the same $\LaTeX$ format (although perhaps not all in the same document since most $\TeX$ programs have an upper limit on the number of fonts that can be used in one document). Thus you can immediately typeset your documents with classical or modern typefaces and combine the superior typesetting quality of $\TeX$ with all the professionally designed typefaces of this century, most of which are available as PostScript fonts. However, integrating new PostScript fonts into $\TeX$ requires some initial work by the user, and this chapter lays the foundation for this work.

We start by looking at the simplest case, where the PostScript fonts are already set up for use with $\TeX$, concentrating on the PSNFSS system (a collection of small packages and accompanying files for $\LaTeX$), which makes it easy to use a large number of common PostScript fonts out of the box. We then show what steps need to be taken to install PSNFSS components for extra fonts you may acquire, and explain how you can access them in your class file or document.

The remainder of the chapter provides information necessary to prepare additional PostScript fonts for use with $\TeX$ or to modify an existing setup. But even if you don't intend to undertake any such steps yourself, this information is valuable in helping you understand how this technology works, where its benefits and its limitations lie and how it all fits together.

This part of the chapter starts with a short introduction to the font concepts of $\TeX$ and discusses the type of files typically encountered in a $\TeX$ system and a number of useful tools for manipulating these files. We then introduce important concepts of PostScript font technology such as the different types of PostScript fonts, how metric information is provided for such fonts and how PostScript fonts are encoded, i.e., how one can access individual characters of a font. Then we look at the naming conventions for PostScript fonts developed and maintained by Karl Berry—dry material, perhaps, but important if we want to handle a large number of fonts in parallel on more than one operating system.

Having cleared the fog, we are ready to apply our new knowledge and look at the integration of PostScript fonts into the $\TeX$ system. Here we discuss the tools and techniques for producing font metric information suitable for use with $\TeX$ from the font metric information provided with PostScript fonts, how virtual fonts can be used to encode fonts so that they match standard $\LaTeX$ font encoding, and so on. We end with a look at integrating Multiple Master fonts, using the setup for this book as a case study.

10.1 Using preconfigured PostScript fonts

$\LaTeX$'s font-selection scheme is documented thoroughly in Chapter 7 of Goossens et al. (1994), and a condensed explanation can be found in the file `fontguide.tex`, which is part of the $\LaTeX$ distribution. For the remainder of this chapter we assume that readers have access to the more detailed description given there.

The *default* action associated with any of the NFSS commands can be changed with the `\renewcommand` macro; each of the declarations has a corresponding command with a suffix of `default`. Thus you could change the effect produced by `\ttfamily` by saying (in the document preamble or a package file):

```
\renewcommand{\ttdefault}{courier}
```

if you have a font family called “courier”. (There may be encoding problems, however, as discussed later.) But how do we get to the stage where this `courier` works? We may be in one of three situations:

1. We are content to use the standard setup distributed with the $\LaTeX$ PSNFSS package, discussed below;
2. The metrics and description files have already been created for the font family we want, so we have to install them and create a package file; this is discussed in Section 10.1.3;
3. We have only the Adobe Font Metric files and need to create all the metric files etc. that $\TeX$ needs; this is rather more complicated and is described in Section 10.5.6.

10.1.1 The PSNFSS system

Let us deal first with the simplest case, using the prepackaged system. The standard $\LaTeX$ PSNFSS package, coordinated by Sebastian Rahtz, provides straightforward support for common PostScript fonts. It consists of a simple set of packages that change high-level family defaults and various more complicated packages, such as one to use Lucida Bright with its companion Lucida New Math and one to use Times in math. Table 10.1 lists the packages and summarizes their effects.

Most people simply want to install support for the fonts available in almost all PostScript printers. These are four serif families (Times, Palatino, Bookman, and New Century Schoolbook), two sans-serif families (Helvetica and AvantGarde), one monospaced typewriter family (Courier), a symbol font, and the cursive Zapf Chancery; these were the fonts Apple provided with the first LaserWriter Plus in 1986. Together these families, in their various shapes, make up 35 fonts, and you often see references to “the 35 fonts” (see Table 10.2 for a sample). If you stick to these fonts, you can be sure that everyone can print your documents.

To obtain the basic setup, acquire the PSNFSS package available in CTAN archives and run $\LaTeX$ on `psfonts.ins`. This creates the packages (`times`, `palatino`, `helvet`, `avant`, `newcent`, `bookman`, `chancery`, `pifont`, and `mathptm`) for the standard fonts, which you can

Table 10.1: Effect of PSNFSS package files.

Package	Sans font	Roman font	Typewriter font
times	Helvetica	Times	Courier
palatino	Helvetica	Palatino	Courier
helvet	Helvetica		
avant	AvantGarde		
newcent	AvantGarde	NewCenturySchoolbook	Courier
bookman	AvantGarde	Bookman	Courier
chancery		Zapf Chancery	
mathptm		Times	
basker		Monotype Baskerville	
bembo		Bembo	
charter		Bitstream Charter	
garamond		Adobe Garamond	
mtimes		Monotype Times	
nimbus	URW NimbusSans-Regular	URW NimbusRoman-Regular	
utopia		Utopia	
lucbr	LucidaSans	LucidaBright	LucidaTypewriter

The mathptm package makes as much math as possible using PostScript Times and other standard fonts. The lucbr package uses the font names for Lucida Bright that conform to the *Fontname* scheme. The package has an option `yy` to use the font names supplied by Y&Y which loads Lucida New Math; an option `expert` makes use of additional upright math and small-caps text fonts.

install in your TeX setup. Now you need to add TeX font metric files, virtual font files, and font description files, and inform your dvi driver about the font names. The files you need to install are available from the CTAN archives in `fonts/psfonts/adobe`, but are more conveniently supplied in the PSNFSS distribution ready to install in the subdirectory called `lw35nfss`. If you use the dvips driver, you can add the contents of the file in the `lw3fnfss/dvips` directory called `psnfss.map` to the installed `psfonts.map`.

The small packages of PSNFSS work by redefining the default roman, sans-serif or typewriter family. Thus the times package basically consists of:

```
\renewcommand{\sfdefault}{phv}
\renewcommand{\rmdefault}{ptm}
\renewcommand{\ttdefault}{pcr}
```

The only difficulty here is knowing the names of the font families; this is discussed below in Section 10.4.

Documents containing math

Many L^AT_EX users who start to use fonts such as Times get an unpleasant shock on seeing a page of mixed text and mathematics. The Computer Modern math fonts do not mix very

Table 10.2: Sample texts for standard PostScript fonts.

Times Roman For the price of £45, almost anything can be found floating in fields. ¡THE DAZED BROWN FOX QUICKLY GAVE 12345–67890 JUMPS! — ¿But aren't Kafka's Schloß and Æsop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?

Palatino For the price of £45, almost anything can be found floating in fields. ¡THE DAZED BROWN FOX QUICKLY GAVE 12345–67890 JUMPS! — ¿But aren't Kafka's Schloß and Æsop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?

New Century Schoolbook For the price of £45, almost anything can be found floating in fields. ¡THE DAZED BROWN FOX QUICKLY GAVE 12345–67890 JUMPS! — ¿But aren't Kafka's Schloß and Æsop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?

AvantGarde For the price of £45, almost anything can be found floating in fields. ¡THE DAZED BROWN FOX QUICKLY GAVE 12345–67890 JUMPS! — ¿But aren't Kafka's Schloß and Æsop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?

Bookman For the price of £45, almost anything can be found floating in fields. ¡THE DAZED BROWN FOX QUICKLY GAVE 12345–67890 JUMPS! — ¿But aren't Kafka's Schloß and Æsop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?

Helvetica For the price of £45, almost anything can be found floating in fields. ¡THE DAZED BROWN FOX QUICKLY GAVE 12345–67890 JUMPS! — ¿But aren't Kafka's Schloß and Æsop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?

Courier For the price of £45, almost anything can be found floating in fields. ¡THE DAZED BROWN FOX QUICKLY GAVE 12345–67890 JUMPS! -- ¿But aren't Kafka's Schloß and Æsop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?

Zapf Chancery For the price of £45, almost anything can be found floating in fields. ¡THE DAZED BROWN FOX QUICKLY GAVE 12345–67890 JUMPS! — ¿But aren't Kafka's Schloß and Æsop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?

10-1-1

well with Times in line thickness, average letter height, and overall design. Figure 10.1 shows an example of this mix, and it is far from satisfying. Unfortunately, real math fonts are few and far between—the only choices for TeX users at this time are Computer Modern, Euler Math, MathTime, and Lucida New Math. There are several difficulties with one or all of these:

1. for scientific publications, the dominant typeface is Times; none of Computer Modern, Euler Math, or Lucida works well with Times;


```
\begin{Pilist}{family}{number}
```

Inevitably, the Pilist environment sets up an itemized list with a Pi character instead of a bullet:

- * Do not forget the Symbol font,
- * which can have effects like
 - ▲ ▲ ▲ ▲ ▲ ▲ ▲
- * and individual symbols like ⊗.

```
\begin{Pilist}{psy}'247
\item Do not forget the Symbol font,
\item which can have effects like
  \Piline{psy}'AA
\item and individual symbols like
  \Pisymbol{psy}'C4.
\end{Pilist}
```

10-1-4

```
\begin{Piautolist}{family}{number}
```

The Piautolist is rather more sophisticated; given a starting number, it increments that counter for each item and tags the list item with the corresponding symbol:

- ① text inside list,
 - α text inside list,
 - β text inside list,
 - i. text inside list,
 - Ⓢ text inside list,
 - Ⓣ text inside list,
 - ii. text inside list,
 - χ text inside list,
 - ② text inside list.
- We can also refer to item Ⓢ as usual.

```
\begin{Piautolist}{pzd}'254
\item text inside list,
\begin{Piautolist}{psy}'141
\item text inside list,
\item text inside list,
\begin{enumerate}
\item text inside list,
\begin{Piautolist}{pzd}'56
\item text inside list,
  \label{pilabel}
\item text inside list,
\end{Piautolist}
\item text inside list,
\end{enumerate}
\item text inside list,
\end{Piautolist}
\item text inside list.
\end{Piautolist}
We can also refer to item
\ref{pilabel} as usual.
```

10-1-5

This interface can be used with fonts other than pzd or psy; all that is required is a font with encoding U, series m (medium) and shape n (normal). One application is to provide a simple interface for using math symbols (such as Greek letters) in ordinary text. What is required is either a font description (.fd) file or font description commands in a package

file; thus to use one of the Lucida New Math fonts, we must define a family and a font shape:

```
\DeclareFontFamily{U}{hlcm}{ }
\DeclareFontShape{U}{hlcm}{m}{n}{<->hlcrm}{ }
```

We can now make use of the font with the \Pisymbol command:

```
\newcommand\textGamma{\Pisymbol{hlcm}{0}}
```

Note that the font itself (hlcrm) is loaded only once, whether it is used here or in a more conventional math package file; all we have done is set up a new internal interface in L^AT_EX's NFSS.

10.1.2 Fonts, metric files — the whole lot

One of the commonest sources of confusion for would-be PostScript font users is knowing which files they have to install and what they have to change in their driver setup. Here we describe in more detail what files are involved in using the PSNFSS setup. The precise details may vary from driver to driver and such details are beyond the scope of this chapter; we describe the range of drivers briefly in Chapter 11 and refer the reader to their documentation.

The PSNFSS packages are implemented using *virtual fonts* (see Section 10.2.3); this means that for every font that is used, there are three font files and an entry in a font description file. Let us consider the common case of Times Bold. With the PSNFSS setup the package file times.sty looks something like this:

```
\def\fileversion{5.2}
\def\filedate{1995/08/16}
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{times}[\filedate\space\fileversion\space
Times PSNFSS2e package]
\renewcommand{\sfdefault}{phv}
\renewcommand{\rmdefault}{ptm}
\renewcommand{\ttdefault}{pcr}
```

This means that when L^AT_EX comes to typeset something in Roman, it looks for a *font-encoding file* named by combining the encoding name and the family name defined by \rmdefault. The default encoding is OT1 and the package defines \rmdefault to be ptm, so L^AT_EX opens ot1ptm.fd,¹ which looks like this:

```
\ProvidesFile{ot1ptm.fd}
[1997/01/01 Fontinst v1.504 font definitions for OT1/ptm.]
\DeclareFontFamily{OT1}{ptm}{ }
\DeclareFontShape{OT1}{ptm}{b}{n}{<-> ptmb7t}{ }
\DeclareFontShape{OT1}{ptm}{b}{sc}{<-> ptmbc7t}{ }
```

¹The encoding name is translated to lower-case before the file is looked for.

“Why”, the astute reader asks, “do you not simply use the name Times-Roman in $\TeX$, do all the re-encoding either with some such mechanism as $\LaTeX$ ’s `inputenc` package or in PostScript, and so avoid the whole farrago of virtual fonts, map files etc.?” This was, indeed, how the first generation of $\TeX$ PostScript drivers worked, and for the simplest cases it still works well and fast. The reason that PSNFSS adopts a more complicated approach are twofold. First, the virtual font mechanism provides vastly more flexibility than simple re-encoding and provides a clean layer of functionality between $\TeX$ and PostScript. Other systems depend on direct manipulation of the PostScript fonts themselves, which is not always possible or advisable in a shared environment. Second, font naming is not standardized so that adopting a rational naming scheme helps a great deal in managing large quantities of fonts.

10.1.3 Installing PostScript fonts

For many of the common PostScript fonts marketed by Adobe, Monotype etc., the CTAN archives contain a recommended set of metric and font description files that can be downloaded for immediate use. These utilize virtual fonts to sort out encoding issues. To install a new font family, you need to take the following steps:

1. Locate the font family in the `fonts/psfonts` directory of the CTAN archives, which is organized by foundry (e.g., `adobe`, `monotype`). For instance, Adobe Garamond is in `adobe/garamond`. The names for the directories are from the *Fontname* scheme discussed in Section 10.4 on p. 381.
2. Each family directory has subdirectories containing `tfm`, `vf` and `fd` files; for Adobe Garamond you need all files from these subdirectories, e.g., `tfm/pgmb7t.tfm` and so forth. You should install these where $\LaTeX$ and your driver can find them: $\LaTeX$ needs the `tfm` and `fd` files and the driver needs the `vf` files (and possibly the `tfm` ones too).
3. You now need to tell your driver that you are going to use these new PostScript fonts (to prevent it from looking for METAFONT sources and `pk` files). With `dvips` this is accomplished by taking the `map` file in the family directory and appending it to the standard `psfonts.map` file of `dvips`. Alternatively, you can install the `map` and the `config` file (named `config.Short-Family-Name`, e.g., Adobe Garamond’s short name is `pag`, so the `config` file is `config.pag`) where `dvips` support files live, and use, e.g., `dvips -Ppag` to load the extra `map` file on the fly. If you don’t use `dvips`, check your driver’s manual for how to tell it about new PostScript fonts.
4. If the font is not resident in the printer, you have to download it. You can use software supplied with the font to do this, or have it done on the fly by some drivers (Section 11.2.4 discusses how `dvips` does this). However, note that the names in the `map` files supplied on CTAN assume strict conformance with the *Fontname* scheme—you may have to rename your `pfb` or `pfa` font file.

5. If you want to load merely one font family and have it replace the default Roman, sans-serif or typewriter family in your document, each of the font family directories has a simple $\LaTeX$ package file.

When the font metrics have been downloaded and installed and your driver has been set up accordingly, you can simply load the appropriate package and proceed as normal. Note, however, that changing the text font family has no effect on mathematics. Installing new math fonts is rather more complicated, and we return to it with an example on p. 401.

If you just want to change the default font of your document to Times, this is as far as you need to read. However, if you are preparing a serious book or journal design, you need a fuller understanding of both $\TeX$ and PostScript font technology, so we now backtrack a little and consider the underlying mechanisms.

10.2 $\TeX$ font technology

$\TeX$ has a very general model of using different typefaces; all it needs is a set of metric information for each font, and it leaves the actual setup to the `dvi` driver. For this reason, speaking of $\TeX$ fonts is not actually very meaningful since, as already mentioned in the introduction, any font can be used with $\TeX$ as long as we can obtain metric information for it (a `tfm` file—see Section 10.2.2) and the driver program can use the font if it is referenced in the `dvi` file. We follow the standard usage of calling fonts produced by METAFONT “ $\TeX$ fonts” but you should keep in mind that it is a misnomer. After introducing the $\TeX$ font formats we concentrate on the $\TeX$ font metric format and finally on the concepts of virtual fonts, both of which we need for the successful integration of PostScript fonts.

10.2.1 Types of $\TeX$ fonts

The *gf* font format

The *gf* (generic font) format is produced by METAFONT when generating a font; this is a *bitmap* format, in contrast to the outline format of the METAFONT font source. It is created for a particular device at a particular size (e.g., at 16 points for a Canon laser printer). The *gf* format does not match the format of any other font software; it was designed to be easy to produce conversion programs to any other format, hence the name “generic”. METAFONT also produces a `tfm` ($\TeX$ font metric) file to go with the font—see Section 10.2.2 for a description.

A number of conversion programs are usually part of a $\TeX$ system, the most important being `gftopk`, which produces a `pk` file, a format understood by all modern `dvi` driver programs. Other programs are `gftodvi`, which produces large pictures of every character in the font (such as the samples in Knuth (1986e)), intended primarily for test outputs when developing METAFONT fonts, and the program `gftype` which tests whether or not a *gf* file is corrupted.

The pxl font format

The pxl is an obsolete font format that was used by the early dvi drivers. If your installation still has pxl fonts, it must be really antique and you should seriously consider upgrading. Transformation programs like gftopxl or pxltopk exist but might be difficult to get, since current T_EX distributions rarely support this pxl format.

The pk font format

The normal bitmap format used by drivers is pk. This is simply a more efficient storage of the g_f format, and is usually generated using gftopk immediately after METAFONT. The size or device characteristics cannot be changed at this stage. It is also possible to use pktogf to translate a pk file back to the generic format, and pktype to test its validity.

The user must keep in mind the fact that bitmap fonts cannot be scaled in size without loss of quality, and that METAFONT generates different bitmap patterns for different devices. The very precise tuning that is possible using METAFONT makes for high-quality output, but has the disadvantage that fonts are not portable among different output devices (although in practice many devices, such as 300-dpi laser printers, are similar enough that they can share font files).

10.2.2 T_EX font metric files

The metric information T_EX needs about characters of a font is stored in files with the extension t_fm (for T_EX font metric). For each character in a font four dimensions are recorded: its width, its height above and its depth below the baseline, and its “italic correction”, i.e., the amount of extra space that is added with the \/ command to take into account a switch from a slanted to an upright typeface.² These dimensions are all that T_EX wants to know about the shape of each character—this is why people often talk about T_EX regarding characters as “black boxes.”

In addition, the t_fm format contains data about certain relationships between characters, such as kernings and ligatures. Kerning between two characters means that when these characters appear next to one another, a certain extra space, the “kern” (which can be negative) should be inserted to balance the visual appearance. A ligature is a glyph that is inserted by T_EX in place of two or more characters that are actually in the document. For example, many fonts contain a “fi” glyph (a single character) that is used automatically if “i” follows “f”. Typesetting systems that do not handle ligatures print “fi” instead.

In fact, the ligature mechanism used by T_EX is much more powerful than the example above suggests; one can specify that characters should change if they appear at beginning or end of a word, more than one glyph can be inserted, and the characters triggering the ligaturing process can be replaced (as in the example above) or survive partly or completely. For example a ligature rule in a font could say, if “a” is followed by “b” then replace “a” with “c” and retain “b.”

² T_EX users normally do not need to worry about adding \/ commands in the right places because the high-level commands like \textit and \emph automatically take care of any necessary italic correction.

Finally, the t_fm format contains information about the font as a whole: a number of dimensions denoting the font design size, the width of a normal space, its stretchability, and so on. From within T_EX they are accessible via the \fontdimen command (Goossens et al., 1994, p. 201). Fonts for mathematics have extra dimensions for specialized setting.

The information in the t_fm file is stored in binary format suitable for fast access by T_EX, but is therefore nearly impossible to read or modify by anything other than a computer. However, we can transform the t_fm file into a readable format with the program tftopl. This produces a p1 file (a property list file) that is human-readable and can be modified if necessary. A t_fm file can be produced from a p1 file using the program pltotf.

A p1 file consists of a list of entries of the form

```
(PROPERTY-NAME VALUE)
```

where the property name is one of a finite set of names and the value, depending on the property, can be a string, a number, or even another property list. To explain the concepts better we give an annotated example of portions of cmr10.p1; a full syntax description of the p1 format can be found in the Web program source of tftopl.

The file starts with the properties relevant to the font as a whole (a COMMENT property is ignored). Numbers can be given as real values (R), decimals (D), or octals (O).

```
(FAMILY CMR)
(FACE 0 352)
(CODINGScheme TEX TEXT)
(DESIGNSIZE R 10.0)
(COMMENT DESIGNSIZE IS IN POINTS)
(COMMENT OTHER SIZES ARE MULTIPLES OF DESIGNSIZE)
(CHECKSUM 0 11374260171)
(FONTDIMEN
  (SLANT R 0.0)
  (SPACE R 0.333334)
  (STRETCH R 0.166667)
  (SHRINK R 0.111112)
  (XHEIGHT R 0.430555)
  (QUAD R 1.000003)
  (EXTRASPACE R 0.111112)
)
```

The FONTDIMEN property has itself a property list with its value listing a number of font dimensions. Since these are needed by T_EX for the typesetting process, the tftopl program that generated this listing knows about them and identifies them by name; e.g., SPACE specifies the normal word space to use when this font is selected. If the font specifies further font dimensions, additional entries (here one for \fontdimen8) might look like this:

```
(PARAMETER D 8 R 0.1)
```


The next part of `cmr10.p1` is the `LIGTABLE` property entry, which holds the information about ligatures and kerning. We present only a small part of this table here:

```
(LIGTABLE
...
(LABEL C f)
(LIG C i 0 14)
(LIG C f 0 13)
(LIG C l 0 15)
(KRN 0 47 R 0.077779)
(KRN 0 77 R 0.077779)
(KRN 0 41 R 0.077779)
(KRN 0 51 R 0.077779)
(KRN 0 135 R 0.077779)
(STOP)
(LABEL 0 13)
(LIG C i 0 16)
...
(STOP)
)
```

This is read as follows: if the characters “f” is encountered (C f means the character in ASCII position “f”) and the following character is “i”, replace both characters by the character in position octal 14 (which is “fi” in that font). Similarly, if “f” is followed by another “f” replace it by octal 13, and so on for the other LIG properties. The KRN properties specify that a bit of extra space should be added ($0.077779 \times \text{DESIGNSIZE}$) when “f” is followed by certain characters (the octal numbers correspond to ‘, ?, !, , and] in this font). The STOP property ends the current ligature/kerning rule and LABEL starts a new one. The LIG property for octal 13 is responsible for generating the three-character ligature “ffi”: by the first ligature rule “f” and “f” are transformed to “ff” (octal 13) and then if this is followed by “i” it is further transformed to “ffi” (octal 16).

The LIG property actually has a number of variant forms: it may be immediately preceded or followed by a slash, and then immediately followed by some number of > characters not exceeding the number of slashes. The slashes specify retention of the original characters on the left or right; the > signs specify that the retained (or inserted) characters should not take part in further ligature processing. For example,

```
(LABEL C a) (LIG/> C b C c) (STOP)
(LABEL C c) (KRN C b R 1.0) (STOP)
```

transforms “ab” into “cb” but the second line is not applied; no kern is inserted between the characters “b” and “c” because the > sign specifies that the ligature processing should continue with “b” and not at the inserted “c” (of course, the kern would appear if “cb” is found directly in the input).

Returning to our example: the `p1` file finishes with a large number of `CHARACTER` property entries (one for each character in the font) of which we show here the one for the character “g”:

```
(CHARACTER C g
  (CHARWD R 0.500002)
  (CHARHT R 0.430555)
  (CHARDP R 0.194445)
  (CHARIC R 0.013888)
  (COMMENT
    (KRN C j R 0.027779)
  )
)
```

The first four property entries give values for the width, height, depth, and italic correction for the character; if any of them is zero it might be omitted. This is followed by a `COMMENT` property entry listing any kerning or ligature information for this character. This is automatically produced by `tfptopl` from the `LIGTABLE` property as a guide for the reader. In this case it tells us that somewhere in the `LIGTABLE` there is a rule to add some extra space between “g” and “j”.

10.2.3 TeX virtual fonts

Virtual fonts (Knuth, 1990) are like font metric files in that they contain general information about the font and detailed instructions for each character. But they stand at a higher level: they describe an “imaginary” font that can consist of bits of other fonts, low-level `dvi` code such as line-drawing, or even `\special` commands. Thus character 47 may say “set a capital T from Times Roman”, character 53 may say “set a 20 point = sign from Computer Modern”, and character 101 may say “use a `\special` command to include this logo picture”. Virtual fonts are used to pretend to TeX that another font is arranged like a Computer Modern font, to allow all TeX’s internal macros to work without problem, or to combine specialized characters from an “expert” font with a normal text font.

A virtual font has two parts: a `tfm` metric file that TeX reads, and a `vf` file that drivers read and that tells them how to create each character. Very often this is used simply to rearrange a font (i.e., “to create character 212 in this font, take character 176 from the other font”), but it can also be used to combine two fonts, join arbitrary characters, or even change color, rotate characters or perform other operations.

Just like `tfm` files, virtual fonts have a human-readable form—the `vp1` file. We show some examples of the code by examining one that implements a T1-encoded Baskerville Roman. Conversion between `vf` and `vp1` format is handled by the programs `vftovp` and `vptovf`.

A `vp1` file can contain the same type of property entries as are allowed in `p1` files. In addition, some other property types are available that deal with selecting characters from

one or more fonts (for which `tfm` or `vf` files exist—i.e., the process of selecting characters can be recursive). These are `VTITLE`, which identifies the virtual font (default value is an empty string), the `MAPFONT` property, which declares the fonts from which characters can be selected, and the `MAP` property, which can appear as part of a `CHARACTER` property and specifies where the character should be taken from. The last two properties take as values property lists with properties relevant only for `vf` files.

In our example, after identifying the font, the virtual font for Baskerville Roman declares two base fonts from which it draws characters:

```
(MAPFONT D 0
  (FONTNAME mbvr8x)
  (FONTCHECKSUM 0 33005554065)
  (FONTAT R 1.0)
  (FONTDSIZE R 10.0)
)
(MAPFONT D 1
  (FONTNAME mbvr8r)
  (FONTCHECKSUM 0 7541561315)
  (FONTAT R 1.0)
  (FONTDSIZE R 10.0)
)
```

The `FONTNAME` refers to the relevant `tfm` file, and the `FONTCHECKSUM` should match the checksum in that font so that the processing software can tell if the right file was found. `FONTDSIZE` should match the design size of the font and `FONTAT` specifies a scaling factor; i.e., with this property you can use a font at a size other than its design size.

A simple entry (here for the number “4”) gives the size and an instruction to pick a character from one base font:

```
(CHARACTER C 4
  (CHARWD R 0.5)
  (CHARHT R 0.654999)
  (MAP
    (SELECTFONT D 1)
    (SETCHAR C 4)
  )
)
```

But the `ff` ligature, for example, might be drawn from the other base font, which is the Expert set for the font. A non-standard accented character can be created using `dvi` primitives (`PUSH`, `POP` and `MOVERIGHT`) to put two base glyphs one on top of the other:

```
(CHARACTER 0 271
  (CHARWD R 0.4269905)
  (CHARHT R 0.6299925)
  (MAP
```

```
(PUSH)
(MOVERIGHT R 0.046997)
(SETCHAR 0 264)
(POP)
(SETCHAR C z)
)
)
```

To appreciate how *anything* that is valid in a `dvi` file can be used in a virtual font, look at the following example from a `vp1` file; this creates a dotless-`j` glyph (not present in most PostScript fonts) by inserting explicit PostScript code to take a dotless-`i` and mask off the dot. Note, however, that this example is highly nonportable, since it relies on the `SPECIAL` property executing raw PostScript, whose interpretation depends on the driver.

```
(CHARACTER 0 32 (comment dotlessj)
  (CHARWD R 278.00)
  (CHARHT R 458.00)
  (CHARDP R 217.00)
  (CHARIC R 0.00)
  (MAP (SELECTFONT D 0)
    (SPECIAL ps: gsave newpath 0 0 moveto (\31) true charpath
      flattenpath pathbbox /IHeight exch def pop pop pop grestore gsave
      newpath 0 0 moveto (\152) true charpath flattenpath pathbbox pop
      exch /JDepth exch def /JRight exch def /JLeft exch def grestore
      gsave newpath)
    (PUSH)
    (MOVEDOWN R 217.00)
    (SPECIAL ps: JLeft JDepth rmoveto JLeft neg JRight add 0 rlineto
      0 JDepth neg IHeight add rlineto JLeft neg JRight add neg 0
      rlineto 0 JDepth neg IHeight add neg rlineto closepath clip)
    (POP)
    (SPECIAL ps: (\152) show grestore)
  )
)
```

Virtual fonts can also be used in font manipulation by calculating metrics for stretched or shrunk fonts, letter-spaced fonts, and small capitals. In some cases, the manipulated metrics must be accompanied by some PostScript code to change the font itself. In `dvips` this can be done with code in the `map` file; thus the single line³

```
hlcbot8rn LucidaTypewriterBoldOblique " .850 ExtendFont
TeXBase1Encoding ReEncodeFont " <8r.enc <hlcbot8a.pfb
```

says that the font known to TeX as `hlcbot8rn` is to be created by performing two operations on the PostScript font `LucidaTypewriterBoldOblique`. First it is anamorphically shrunk to 85% of its natural width, and then it is re-encoded to `TeXBase1Encoding`. The

³Due to space limitations in this book the single configuration line is shown as two lines.

transformations `ExtendFont` and `ReEncodeFont` are defined in PostScript, in the standard header files downloaded by dvips. Two resources are supplied, the definition of the encoding, in `8r.enc`, and the Type 1 font itself, in `h1cbot8a.pfb` (Chapter 11 gives more details on how dvips works). The metric file that TeX originally read for `h1cbot8rn` was created by taking the normal metrics for `LucidaTypewriterBoldOblique` and writing a virtual font with the shrunken values.

10.3 PostScript font technology

10.3.1 Types of PostScript fonts

The page description language PostScript can be used to describe complex graphics images in an efficient way. It is beyond the scope of this book to provide a systematic description of PostScript, but we note now that each character in a PostScript font is described by a small PostScript program specifying character *outlines*, i.e., collections of lines, arcs, and curves. To realize the characters on an output device, the PostScript interpreter rasterizes these outlines dynamically and transforms them into a bitmap image, taking into account the resolution of the output device. In contrast, METAFONT performs the rasterization once and for all at the stage at which the `gf` file is created.

We describe below the principal ways of representing fonts in PostScript.

Type 1 fonts

The Type 1 font format is a compact way of describing a font outline using a well defined language that can be quickly interpreted. When a character is described in outline format it has unlimited resolution: if you make it ten times as big, it is just as accurate as if it were ten times as small.

The problem here is that a pixel has a physical size and can be printed only as black or white. Look at a sheet of graph paper: rows and columns of little squares. Draw a large “O” in the middle of the graph paper. Darken all the squares touched by the O. Do the darkened squares look like the O you drew? This is the difficulty with low resolution (less than 300 dots to the inch). Which pixels do you turn on and which do you leave off to most accurately reproduce the character?

To solve this problem, Type 1 fonts use “hints” that suggest to the rasterizer what to do at small or large sizes. All methods of hinting strive to fit the outline of a character onto the pixel grid and produce the most pleasing or recognizable character, no matter how coarse the grid is. TeX’s companion program, METAFONT, also provides this function.

A Type 1 font program is written in a subset of the PostScript language and is organized as a collection of named procedures describing character shapes. When a character code is requested, the Type 1 renderer first uses the character code as an entry in an *encoding* array in the font dictionary to obtain the name of the character to be constructed; by re-encoding in the font, it is thus possible to create various different layouts of the characters. The name of the character is used as a key in the `CharStrings` dictionary; this yields a binary string representing the outline of the character, which is then drawn.

Before the main body of the font is a set of preliminary matter. The following shows the structure of a Type 1 font file converted to readable form (we use Times Roman as an example):

```
%!PS-AdobeFont-1.0: Times-Roman 001.007
%%CreationDate: Tue Mar 20 12:15:38 1990
%%VMusage: 30487 37379
11 dict begin
/FontInfo 10 dict dup begin
/version (001.007) readonly def
/Notice (Copyright (c) 1985, 1987, 1989, 1990 Adobe Systems Incorporated.
/Copyright ( The digitally encoded machine readable outline data for
producing the Typefaces licensed to you is copyrighted
(c) 1981 Linotype AG and/or its subsidiaries. All Rights Reserved.
...
) readonly def
/FullName (Times Roman) readonly def
/FamilyName (Times) readonly def
/Weight (Roman) readonly def
/ItalicAngle 0 def
/isFixedPitch false def
/UnderlinePosition -100 def
/UnderlineThickness 50 def
end readonly def
/FontName /Times-Roman def
/Encoding StandardEncoding def
/PaintType 0 def
/FontType 1 def
/FontMatrix [0.001 0 0 0.001 0 0] readonly def
/UniqueID 28416 def
/FontBBox{-168 -218 1000 898}readonly def
currentdict end
currentfile eexec
```

The important elements here are the font name (`FullName`) and the `Encoding`; we need the name to access the font, and we need to know the encoding. In this font, the encoding is “`StandardEncoding`” but other fonts may provide specific layouts with an encoding vector (as discussed in Section 10.3.3 on p. 378). The general font description is followed by the encrypted information that defines the actual character shapes; here is the decoded description of a single character:

```
/asciicircum {
  24 469 hsbw
  0 422 vstem
  297 365 hstem
  68 297 rmoveto
  143 289 rlineto
```

```

143 -289 rlineto
68 hlineto
-181 365 rlineto
-60 hlineto
-181 -365 rlineto
closepath
endchar
} |-

```

Type 1 fonts are typically supplied by a vendor in a form suitable for a Macintosh or for Windows and Unix systems. The latter are in Printer Font Binary form (pfb); to read or edit them, they must be converted to Printer Font ASCII (pfa) form. Both this conversion and that to and from the special Macintosh format can be done with the public-domain t1utils programs by Lee Hetherington or with the Font Manipulation Package sold by Y&Y (see Section 10.5.5 on p. 389).

Composite fonts

The PostScript language supports two kinds of font programs: base fonts and composite fonts. A *base font*, as described in Chapter 5 of the *PostScript Language Reference Manual* (Adobe Systems, 1990), is a font program containing about 200 characters accessed by single-byte codes.

A *composite font* program is a “font of fonts.” Instead of characters, it contains descendant font programs in a recursive, hierarchical structure, at the leaves of which are ordinary base font programs. Composite font programs can contain several thousand characters, accessed by multi-byte codes. They are mostly useful for non-Roman scripts, such as Kanji characters, which can also have several writing directions. The font program can contain several sets of metrics, and a key in the font dictionary selects the set of metrics used to show a string.

Multiple Master fonts

Multiple Master typefaces are an extension of Type 1 fonts that let you customize the appearance of a typeface for a specific document. With a Multiple Master typeface, you can create a condensed version of a headline that exactly fits the space available for it, or take a table or block of copy that doesn't quite fit a page, set it in a Multiple Master typeface, and try condensed or extended versions until it does.

With Multiple Master technology, an enhanced version of the Adobe Type Manager (ATM) aids in the exchange of formatted electronic documents. The program automatically creates substitute fonts that simulate typefaces missing from your computer, so you can view, edit and print pages that retain the same spacing, line breaks and other formatting details as the original text. This is the technology used in Adobe Acrobat.

A Multiple Master font contains two or more outline typefaces called *master designs* that describe one or more *design axes*. The master designs constituting a design axis represent a dynamic range of one typographic parameter, such as the weight or width. This

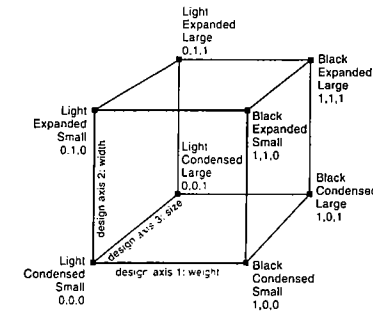


Figure 10.4: Multiple Master typeface design space.

range of styles is defined in a Multiple Master font program by specifying one master design to represent each end of an axis, such as a *light* or *black* weight, as well as any *intermediate master designs* that are required. The maximum number of master designs allowed is sixteen.

It is possible to think of the master designs as being arranged in a one, two, three, or four dimensional space with various font instances corresponding to different locations in that space. The entries in the FontInfo dictionary specify what this space is and where the master designs are within it. This information is necessary for interactive programs that let users create new font instances, and is given in the font's Adobe Multiple Font Metrics (amfm) file.

Figure 10.4 shows the design space of a three axis Multiple Master font in which the axes are *weight*, *width*, and *optical size*. Figures 10.5 and 10.6 show some instances of the two-dimensional Myriad and three-dimensional Minion design spaces.

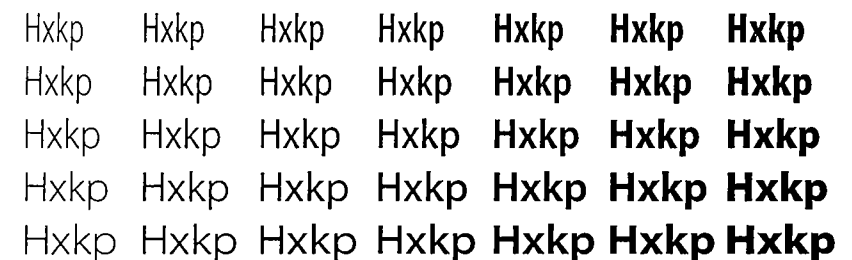


Figure 10.5: Two-dimensional grid showing various instances of the two axis Multiple Master Myriad sans-serif font.

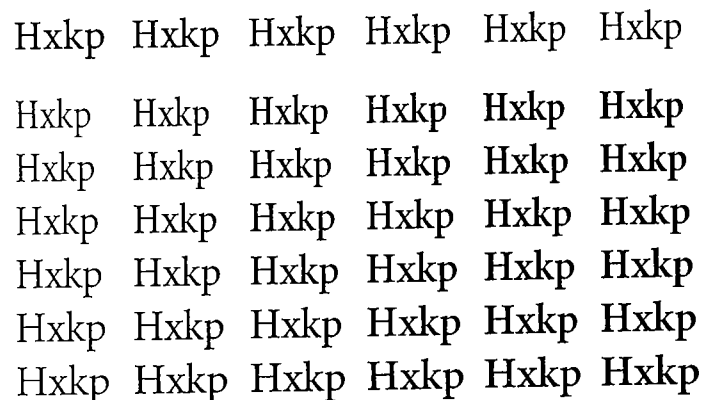


Figure 10.6: The three axis Multiple Master Minion serif font. The top line shows various optical sizes (6pt, 8pt, 11pt, 18pt, 40pt, and 72pt) normalized to 20pt. The bottom matrix shows various weights (increasing from left to right) and widths (increasing from top to bottom).

Using Multiple Master fonts with L^AT_EX, as discussed in Section 10.8, is fairly straightforward. The book you are reading was typeset using Adobe Multiple Master Minion and Myriad.

Type 3 fonts

There is another format for PostScript fonts, the Type 3 format, for which it is easier to develop font code, since you can use all PostScript operators to make your character images. It is, however, impossible to use hints, and character rendering is less efficient than for Type 1 fonts. PostScript dvi drivers often use this type of font when they incorporate output from METAFONT, which generates bitmapped fonts from outline source descriptions. The dvi driver extracts bitmaps from the packed font images (pk files) for all T_EX fonts used in a T_EX job, and transforms them into Type 3 PostScript fonts for inclusion in the output. Since these fonts are generated at a given resolution and for a given printer, they do not look well on a different printer or at a higher resolution than that for which they were originally intended. Therefore, to use higher resolution printers, you have to regenerate all the pk files at that resolution—a huge job, taking up many megabytes of disk space. Alternatively, you can use Type 1 renderings of the T_EX fonts—for Computer Modern fonts commercial and freely available versions exist. The same difficulty also arises if you want to reduce your pages to print two or more pages onto one physical page (using the tools described in Chapter 11), since you then have to scale the fonts.

TrueType fonts

TrueType, developed by Microsoft and Apple, is a later font format than that of PostScript. As in PostScript, fonts are stored as outlines (using second order curves rather than PostScript's third order) and the rendering engine is built into the operating system (Mac System 7 or Microsoft Windows and NT). For the user, this is equivalent to Adobe Type Manager—any application can access the fonts and have them rendered at any size, for any supported device (both screens and printers). The fonts have hints, like PostScript Type 1.

With the very widespread use of Windows, TrueType fonts are extremely common, but are used surprisingly little with T_EX; this is due partly to the fact that only the commercial T_EX implementations currently support them, and partly to the lack of widely known tools to create T_EX metrics for TrueType fonts. Kinch Computer Company's TrueT_EX package works entirely with TrueType (Kinch supplies the Computer Modern family in this format); this T_EX implementation is also supplied with Scientific Word, which provides a friendly front end to L^AT_EX. The V_TE_X variant developed and sold by MicroPress has its own font format, but provides conversion utilities.

The TrueType font format is supported by Adobe Acrobat and the free Ghostscript PostScript interpreter⁴; in this context, PostScript calls encapsulated TrueType Type 42 fonts. Adobe and Microsoft have agreed (as of mid-1996) to collaborate on a fusion of the two font formats to be known as Open Type; one of the advantages Open Type offers over the current Type 1 format (which is regarded as somewhat technically superior to TrueType) is that it is a true 16-bit system, allowing fonts with 65000+ characters to be available at the same time.

10.3.2 Adobe Font Metric files

Adobe Font Metric (AFM) or Adobe Multiple Font Metrics (AMFM) files are a standard means of interchanging font metric information between programs. These ASCII-encoded files are machine-independent and extensible. An afm file provides both global metrics for a font program and information for individual characters. For a Multiple Master (MM) font program, a single amfm file specifies data and global information for the complete MM font, while for each master design in the font program there exists an individual afm file.

An afm file has four parts. The first part has general information about the font, with its name, weight, cap height, x-height, and default encoding:

```
StartFontMetrics 2.0
Comment Copyright (c) 1991 Adobe Systems Incorporated.
Comment Creation Date: Fri Sep 6 17:58:27 1991
Comment UniqueID 301073
Comment VMusage 32532 39424
FontName BaskervilleMT
FullName Baskerville MT
FamilyName Baskerville MT
Weight Regular
```

⁴The release 4.03 available at the time of writing does not, however, read the font hints.

```

ItalicAngle 0
IsFixedPitch false
FontBBox -160 -255 1006 902
UnderlinePosition -100
UnderlineThickness 50
Version 001.003
Notice Copyright (c) 1991 Adobe Systems Incorporated.
EncodingScheme AdobeStandardEncoding
CapHeight 662
XHeight 400
Ascender 672
Descender -246

```

Second comes a set of metric fields for each character; characters are designed on a notional grid of 1000 units square to which the fields relate:

1. a numeric code; this is the position in the default encoding; note that characters may have a code of -1, meaning that they are not available without re-encoding;
2. the width of the character;
3. the name of the character;
4. the character bounding box.

Only portions of this section are listed below:

```

StartCharMetrics 228
C 32 ; WX 250 ; N space ; B 180 0 180 0 ;
C 33 ; WX 250 ; N exclam ; B 73 -16 178 656 ;
C 34 ; WX 408 ; N quotedbl ; B 66 388 342 674 ;
C 35 ; WX 668 ; N numbersign ; B 45 -23 622 669 ;
C 36 ; WX 490 ; N dollar ; B 46 -102 444 676 ;
....
C 118 ; WX 458 ; N v ; B -2 -16 460 400 ;
C 119 ; WX 677 ; N w ; B 1 -16 677 400 ;
C 120 ; WX 479 ; N x ; B 8 0 471 400 ;
C 121 ; WX 458 ; N y ; B -2 -246 461 400 ;
C 122 ; WX 427 ; N z ; B 33 0 393 400 ;
C 123 ; WX 480 ; N braceleft ; B 105 -249 378 669 ;
....
C 248 ; WX 240 ; N lslash ; B 1 0 238 672 ;
C 249 ; WX 500 ; N oslash ; B 37 -29 464 429 ;
C 250 ; WX 771 ; N oe ; B 37 -16 744 416 ;
C 251 ; WX 521 ; N germandbls ; B 15 -5 486 672 ;
C -1 ; WX 438 ; N ecircumflex ; B 37 -16 399 616 ;
C -1 ; WX 438 ; N edieresis ; B 37 -16 399 551 ;
C -1 ; WX 469 ; N aacute ; B 39 -10 459 626 ;

```

```

C -1 ; WX 760 ; N registered ; B 34 -18 726 674 ;
C -1 ; WX 250 ; N icircumflex ; B -15 0 267 616 ;
EndCharMetrics

```

Note, we stress again, that the -1 assigned to ecircumflex means that this character is by default unavailable, i.e., that the font must be re-encoded to allow access to it.

The third section consists of kerning information, the amount of space (perhaps negative) to insert between pairs of characters; the first entry below means that when the letter A is followed by C, they should be moved a little closer together.

```

StartKernData
StartKernPairs 496
KPX A C -28
KPX A G -37
KPX A O -37
KPX A Q -37
KPX A T -74
KPX A U -46
KPX A V -120
KPX A W -66
KPX A Y -111
KPX A p 18
....
KPX quoteright v -37
KPX quoteright quoteright 0
KPX quotesinglbase quotesinglbase -28
EndKernPairs
EndKernData

```

Last is a section of information about composite characters that shows how certain characters are constructed; for example, Aacute is not a character in its own right, but is formed by joining an A and an acute accent.

```

StartComposites 56
CC Aacute 2 ; PCC A 0 0 ; PCC acute 172 248 ;
CC Acircumflex 2 ; PCC A 0 0 ; PCC circumflex 172 248 ;
CC Adieresis 2 ; PCC A 0 0 ; PCC dieresis 172 248 ;
CC Agrave 2 ; PCC A 0 0 ; PCC grave 172 248 ;
CC Aring 2 ; PCC A 0 0 ; PCC ring 172 245 ;
....
EndComposites
EndFontMetrics

```

The third and fourth sections are optional; a math symbol font would simply have the general information and the individual character metrics.

As you can see, unlike the `tfm` format, the `afm` format records only kerning information but not ligatures, both of which are needed for high quality typesetting with $\TeX$. This means that programs converting `afm` to `tfm` need to be given extra information that is not shipped with the PostScript font. How this can be accomplished is shown on p. 392.

10.3.3 Encoding of PostScript fonts

PostScript fonts are arranged as a set of procedures that use a lookup table of *names* for the characters; fonts include a mapping between the names and numbers (the encoding vector), but this is easily changed. Thus, while the character exclamationdown (!) is *often* mapped to decimal 161, it can always be called by name or mapped to another number.

An encoding vector is a normal PostScript array of 256 elements:

```
/AnEncoding [           % now 256 chars follow
/grave /acute /circumflex /tilde /dieresis /hungarumlaut
/ring /caron /breve /macron /dotaccent /cedilla
/ogonek /quotesinglbase /guilsinglleft /guilsinglright
. . . . ]
```

When the font is requested, a vector can be supplied to override the default as specified in the font. It is an unfortunate fact of life that there is no generally accepted standard for encoding layouts of fonts; everyone agrees on a certain number of characters and where they are to be found (the ASCII character set), but this is not adequate for serious typesetting. Tables A.3 and A.4 in Appendix A.6 show the normal Roman text characters and their positions in various widely used encodings:

EC the $\LaTeX$ T1 encoding, also known as “Cork” (after the $\TeX$ Users Group meeting in Cork (Ireland) in 1990 where it was agreed upon);

TeXBase1 an arbitrary re-encoding to access all the standard characters in PostScript fonts, used by the PSNFSS virtual fonts;

ISO Latin 1 the extended ASCII set designed to cover western European languages;

Standard the default encoding for PostScript fonts, created by Adobe; this does *not* encode all the characters in a font;

Windows ANSI a non-standard extended ASCII similar to ISO Latin 1, widely used due to the Microsoft products that support it;

Mac Roman the Macintosh equivalent of the above, used by all Mac applications;

PDF another Adobe encoding, the internal one used as standard by the Portable Document Format.

These encodings, and the PostScript fonts themselves, are within the so-called “eight bit” computer world, which means that each “character” code has only 256 different possibilities. Computer operating systems, languages, and fonts are now moving to 16 (or even 32)

bits, which allow over 65000 (or more) different characters for each encoding. This means that we can uniquely identify all the characters from most of the world’s alphabets within a single encoding. The Unicode standard (Unicode Consortium, 1996) describes a 16-bit encoding that is being widely adopted; a version of $\TeX$ called Omega (Ω) is being developed to work with this encoding.

With the *current* $\TeX$, PostScript, and fonts, however, it is very important to remember two facts:

1. the PostScript font characters are arranged *by name*—there is neither any “right” encoding, nor any overhead in setting up a new encoding;
2. there may very well be characters in the font which are *unencoded* by default with Adobe Standard encoding; indeed, there may well be more than 256 characters, so that no one encoding can access all the characters (luckily, a font may be loaded more than once, with different encodings).

10.3.4 Rendering PostScript fonts

Turning outlines into pixels

In printing the character outline is transferred to a sheet of paper through a device called a raster image processor (RIP) which builds the image of the character out of lots of little picture elements (pixels). PostScript code is normally rasterized in a printer, which has a read-only memory (ROM) containing the PostScript interpreter as well as the most commonly used fonts. The PostScript code can equally well be rasterized on a computer and used for screen display; on PCs and Macintoshes, Adobe Type Manager (see below) renders fonts *within* other applications such as word processors, while the widely used free program Ghostscript (discussed in Chapter 11 and available for many platforms) is an example of a complete PostScript interpreter. The NeXT operating system does a great deal of its work using Display PostScript, also seen in Sun’s OpenWindows.

In considering what it takes to turn an outline picture of a character into a very large number of tiny pixels, it is clear that processors, in a printer or for a computer screen, require considerable power. The original Apple LaserWriter probably had a more powerful chip and more memory than the Macintoshes driving it. On a 300 dpi black and white printer, you need about 1.1 Mbytes of memory to rasterize a page of fonts; at 600 dpi, you need about four times more. As you must also leave room for the workings of the rasterizing process, for the PostScript dictionaries, for expanding bitmaps that might be present in the input stream, and, of course, for the PostScript code corresponding to one output page, you need at least 4 Mbytes of RAM on a 300 dpi printer, while 8 Mbytes is a minimum for a 600 dpi printer. Obviously high-resolution phototypesetters and color laser printers require even more memory. An A0 device capable of printing in four colors needs about 70 Mbytes of memory just to hold each rasterized page ready for printing.

To show the difference between font images at various resolutions and using different methods, Figs. 10.7 and 10.8 show the word “PostScript” set in Computer Modern rasterized at 85 and 300 dpi. (In each case the output is shown at a large size to exaggerate the

Figure 10.7: A font generated for 85 dpi (e.g., computer screen).

Figure 10.8: A font generated for 300 dpi (e.g., LaserWriter).

Figure 10.9: An outline font.

differences.) Fig. 10.9 shows the same text set using a PostScript outline version of the same font rasterized at 2400 dpi (the resolution used to print this book).

The Adobe Type Manager

Adobe Type Manager (ATM) is a font rasterizer for display on “normal” computers (i.e., not printers) that intercepts an application’s standard text calls to the operating system, so that ATM can image the text instead. Applications can also call ATM directly through its application program interface (API).

When ATM intercepts a text call referencing a Type 1 font, it either renders the bitmaps directly to the screen or hands them to the device driver. If the font is not Type 1, control is returned to the system, which processes the request in the normal manner as though ATM were not there.

ATM allows the same outline font to be used for both screen display and printing, making possible a more precise correspondence between the screen and printer. It creates screen

Table 10.3: The standard PostScript fonts in the *Fontname* scheme.

pagk	AvantGarde-Book	phvrrn	Helvetica-Narrow
pagko	AvantGarde-BookOblique	phvrron	Helvetica-NarrowOblique
pagd	AvantGarde-Demi	pncb	NewCenturySchlbk-Bold
pagdo	AvantGarde-DemiOblique	pncbi	NewCenturySchlbk-BoldItalic
pbkd	Bookman-Demi	pncr	NewCenturySchlbk-Italic
pbkdi	Bookman-DemiItalic	pncr	NewCenturySchlbk-Roman
pbkl	Bookman-Light	pplb	Palatino-Bold
pbkli	Bookman-LightItalic	pplbi	Palatino-BoldItalic
pcrb	Courier-Bold	pplri	Palatino-Italic
pcrbo	Courier-BoldOblique	pplr	Palatino-Roman
pcrr	Courier	psyr	Symbol
pcrro	Courier-Oblique	ptmb	Times-Bold
phvb	Helvetica-Bold	ptmbi	Times-BoldItalic
phvbo	Helvetica-BoldOblique	ptmri	Times-Italic
phvbrn	Helvetica-NarrowBold	ptmr	Times-Roman
phvbon	Helvetica-NarrowBoldOblique	pzcmi	ZapfChancery-MediumItalic
phvr	Helvetica	pzdr	ZapfDingbats
phvro	Helvetica-Oblique		

bitmaps directly from the outline font, thus avoiding the jagged characters associated with the traditional method of bitmap scaling. ATM also allows the printing of Type 1 fonts to a wide variety of non-PostScript printers, rasterizing the outline fonts and automatically downloading them to the printer.

Both the Type 1 and Type 3 formats are scalable, and both can be run on any PostScript interpreter. However, because a Type 3 font program needs a full PostScript interpreter to be sure of rendering all features, Type 3 font programs cannot be used by the Adobe Type Manager.

10.4 Classifying PostScript fonts

This section describes Version 2.0 of *Fontname*, a naming scheme for TeX font filenames that is the basis of the L^AT_EX P_SNFSS support for PostScript fonts. It is maintained by Karl Berry, and the full scheme is available as an electronic document in the CTAN archives, `info/fontname`.

The aim of *Fontname* is short, portable filenames. To ensure portability across the maximum number of platforms, we must limit filenames to eight characters, for compatibility with DOS filesystems and the ISO 9660 standard used for CD-ROM distribution. Moreover, these filenames may consist only of letters (monospace), numerals, and underscore. A typical result is shown in Table 10.3, which lists short names for the standard 35 PostScript fonts.

The basic scheme for the filenames is to use eight letters in seven groups

S TT W [V...] [N] [E] [DD]

where

S is the *supplier* of the font,

TT is the *typeface name*,

W represents the *weight*,

V... represents the *variant(s)*, and is omitted if both it and the width are normal. Many fonts have more than one variant;

N is the *encoding*, and is omitted if the encoding is nonstandard. Encodings are described in the section on variants;

E represents the width (*expansion*), and is omitted if it is normal;

DD is the design size (in decimal), and is omitted if the font is linearly scalable.

The weight, variants, and width are probably all best taken from the original name of the font, rather than trying to relate them to some external standard. Some font families (notably Univers) are constructed with a large number of variants according to rational schemes, but these are the exception rather than the rule.

The “supplier” is the source of a font, typically a (digital) type foundry. You should use the supplier letter for the supplier from which you obtained the font, not the original source, since font resellers typically make modifications to the original design. For example, Avant Garde was designed by Herb Lubalin for ITC, but Adobe also sells it; the name of the font that you get from Adobe should start with p.

We may note the codes for anomalous “suppliers”:

- f* Fonts that are distributed by individuals or with no specific attribution to their creator.
- r* Obsolete (specifies raw fonts in an old distribution of dvips). New fonts should never use *r*. (The right thing to do is specify the correct encoding, variant, or whatever the font’s characteristics actually are.)
- z* For fonts that just don’t fit well into the naming scheme. The *z* should be followed by the real supplier letter.

A full list of known suppliers is given in Table 10.4; the second column shows a suggested abbreviation that can be used to name directories.

In the *Fontname* scheme a “typeface” is a collection of related fonts. Sometimes the same typeface abbreviation is used for fonts with different supplier names if one knows (usually by inspection) that the fonts truly are the same. Appendix A.5 lists the typefaces named at the time of writing; variant names from Monotype (m) and Bitstream (b) are noted.

Table 10.4: Font suppliers defined in the *Fontname* scheme.

9	unknown	
a	autologi	Autologic
b	bitstrea	Bitstream
c	cg	Compugraphic
d	dtc	Digital Typeface Corporation
e	apple	Apple
f	public	freely distributable (i.e., public domain)
g	gnu	Free Software Foundation
h	bh	Bigelow & Holmes
i	itc	International Typeface Corporation
k	softkey	SoftKey
l	linotype	Linotype
m	monotype	Monotype
n	ibm	IBM
o	corel	Corel
p	adobe	Adobe (p for PostScript)
r	—	raw (for [obsolete] use with virtual fonts)
s	sun	Sun
u	urw	URW
z	—	bizarre (fontname is nonstandard)

Table 10.5: *Fontname* weight codes.

a	Thin Hairline	l	Light
b	Bold	m	Medium
c	Black	p	Poster
d	Demi	r	Regular Roman
h	Heavy Heavyface	s	Semibold
j	ExtraLight	u	Ultra UltraBlack
k	Book	x	ExtraBold ExtraBlack

The “weight” of a font specifies its boldness. Arranged from lightest to heaviest (more or less), these are *hairline*, *extra light*, *light*, *book*, *regular*, *medium*, *demibold*, *semibold*, *bold*, *extra bold*, *heavy*, *black*, *ultra*, and *poster*. An alphabetically ordered list of abbreviations for weights is given in Table 10.5. The names are those typically used in real font names.

“Variants” include typeface variations (e.g., italic, sans-serif), and font encodings (e.g., Adobe standard, TeX text). A fontname may require multiple variants. To resolve the worst ambiguities, any encoding variant (7, 8, or 9, see below) should come last and any other numeral variant come first (to avoid confusion with a design size). It is recommended (though not required) that all other variants be specified in alphabetical order.

The letterspacing possibilities introduced by fontinst and virtual fonts have not yet become sufficiently widespread to make standardization beneficial and likewise for the

Table 10.6: *Fontname* variant codes.

Codes assigned in earlier versions of the scheme are shown in the second column.			
4	fax; now typeface 1x, Lucida Fax	c	small caps, small text
5	escape for phonetic encodings	d	display, titling, headline
5a	phonetic alternate	e	copperplate, elite
5i	phonetic IPA	f	gothic, old English
5s	SIL-IPA	i	italic, cursive
7	escape for (mostly) 7-bit encodings	l	outline
7a	alternate characters only	m	TeX math italic
7c	2 Dfr (changed characters only)	n	informal
7d	9 old-style digits; Adobe's OsF	o	oblique or slanted
7f	3 fraction	p	ornament
7m	TeXMathItalicEncoding	r	roman or sans; often omitted
7t	TeX text encoding	s	sans-serif
7v	TeXMathExtensionEncoding	t	typewriter, monowidth
7y	TeXMathSymbolEncoding	u	unslanted italic
8	escape for (mostly) 8-bit encodings	v	TeX math extension
8a	0 StandardEncoding	w	script, handwritten, calligraphic
8m	Macintosh standard encoding	x	built with Adobe expert encoding
8r	TeXBase1Encoding	y	TeX math symbol
8t	q CorkEncoding	1	semi-sans (obsolete)
8u	XT2Encoding (cmtt + Latin 2)	6	semi-serif (obsolete)
8w	Windows 3.1 ANSI encoding	b	bright; now typeface lh
8x	Expert encoding	g	grooved (obsolete)
8z	XL2Encoding (cmr + Latin 2)	h	shadow (obsolete)
9	escape for expert encodings	j	invisible (obsolete)
9e	expert + Cork	k	Greek (obsolete)
9t	expert + TeX text	z	Cyrillic (obsolete)
a	alternative arrows		

many possible ways to generate pseudo-small-cap fonts. The available variants are listed in Table 10.6, with the following caveats:

1, 2, 4, 6, b These variants (“semisans”, “bright”, “semiserif”, and “fax”) were used only for single typefaces; to conserve variant abbreviations, they now have typeface abbreviations instead.

g, h, j Since these variants were not actually used in any font names, they are now obsolete and may be reassigned in the future.

5, 7, 8, 9 These characters begin a two-character “encoding variant”. Generally, 7 is for 7-bit encodings, 8 is for 8-bit encodings, and 9 is for “expertised” encodings (see the x item below), but this is not an absolute rule. For a font to be named with a certain encoding variant, all the characters need not appear in precisely the same positions as in the encoding definition: it is enough that the usual TeX macros work. In practice, this means that it is alright to label a font 7t if its only difference from Computer Modern

Table 10.7: *Fontname* width codes.

c	Condensed, Cond	q	ExtraCompressed, ExtraCondensed
e	Expanded	r	Regular, Normal, Medium
n	Narrow, Thin	u	UltraCompressed
o	UltraCondensed	w	Wide
p	Compressed, Compact	x	Extended, Elongated

is that the ligatures and the lslash and Lslash characters are different, since it is impossible to access or change the ligature table from TeX. If a name does not contain a specific encoding variant, its encoding is unspecified.

9 This is needed only because x followed by a two-character regular variant makes some names too long. That is, 9t is equivalent in meaning to x7t.

k, z These variants (indicating Greek and Cyrillic fonts, respectively) are obsolete; future fonts in different scripts should be assigned an appropriate encoding abbreviation (these are not currently standardized.)

m, v, y These math-related variants are retained for the sake of typeface-specific math encodings, e.g., Lucida. Fonts that specifically use the Computer Modern math encodings should use 7m, 7v, and 7y.

r Included only if no other variants, including encodings, apply and if either the width is not r or a design size is present; i.e., r is used only as a placeholder. When the normal version of the typeface is sans-serif (e.g., Helvetica), r should be used when necessary, not s. Use s only when the typeface family has both serif and sans-serif variants.

x, 8x 8x indicates a font in the Expert encoding itself. x indicates an “expertized” font, i.e., a composite (i.e., virtual) font that includes characters from an 8x font.

The “width” of a font specifies the compression or expansion of the font. Arranged from narrowest to widest (more or less), widths are *ultra compressed*, *extra condensed*, *compressed*, *condensed*, *narrow*, *regular*, *extended*, *expanded*, and *wide*.

Expansion or compression of fonts is sometimes done automatically (as by the PostScript scale operator), and sometimes by human designers. In the latter case, the designer also presumably chooses a font name including “Extended” or “Expanded” or “Narrow” or whatever, according to preference, and the abbreviation can follow along. When creating a new synthetically expanded or compressed font for use with TeX, e.g., with afm2t1fm or fontinst, use n and e. Table 10.7 shows the possible widths.

Most implementations of TeX look up a metric t1fm file by searching for a file with the user-supplied name (possibly in any of a series of directories). But one could also first look up t1fm names in another file (or set of files) that specifies the actual filename. In this case, the fontname used in the TeX source file could be almost anything, of any length. Karl Berry has implemented such a mapping file in his Unix Web2c TeX implementation, and other schemes are proposed, including variants of the X Windows font database and PostScript font resource databases. None of these schemes is as yet widespread.

10.5 Setting up new PostScript fonts

10.5.1 TeX and encoding

TeX itself works not with character names but with absolute numeric codes, and thus translates a request for typesetting a character into a request to typeset the character at a specific position in a certain font. But when we request a character using a macro like `\' {c}`, we want the effect to be `é`; thus TeX has to know where to find this in a font, a nontrivial task if the position of the characters can change from one font to another. We are likely to meet at least three major TeX font layouts, or five if we use PostScript fonts:

1. the layout of Knuth's original Computer Modern fonts, which in fact consist of several slightly different layouts, sometimes differing in only one character;
2. the "Cork" extended TeX layout;
3. one of the extended ASCII layouts, as used in Windows or Macintosh applications;
4. the Adobe standard layout (the default encoding of PostScript fonts);
5. a re-encoded PostScript layout such as "TeXBase1" described below.

This means that the definition of macros like `\'` must vary according to the font we are using or, alternatively, all fonts must use exactly the same font encoding to avoid absolute chaos.

L^ATeX calls such commands "encoding-specific" and provides an interface for specifying different definitions that is activated when the font encoding changes. This even allows us to define a command like `\'` in one encoding to produce a character composition out of an accent and a base character, and in another encoding to select a single character (if `é` is present in the font.) It is the first and so far only major macro format that supports this concept; all other macro packages define these commands statically and thus run into severe problems when new fonts are used.

10.5.2 The TeXBase1 encoding

The font names in the PSNFSS packages we looked at in Section 10.1.1, created according to the *Fontname* scheme, reflect the encoding used. Thus the OT1-encoded Times Roman font is called `ptmr7t` and the T1-encoded font `ptmr8t`. But the map files supplied with PSNFSS (for dvips) do *not* use these names but refer to an intermediate "raw" form, which for Times Roman is `ptmr8r`. What is happening is that each font is re-encoded to a new encoding standard, TeXBase1, which is used to build TeX virtual fonts for various different user encodings; both `ptmr7t` and `ptmr8t` are virtual fonts referring to the same underlying raw font. Since not all the characters in the Cork encoding are present in a typical PostScript font, it is not possible to simply re-encode the font directly to the final form.

The raw font encoding TeXBase1 was created by an ad hoc group of interested TeX font experts, primarily Karl Berry, Berthold Horn, Alan Jeffrey, Pierre MacKay, and Sebastian

Rahtz. The aim was to have available for typesetting all the characters normally included in Type 1 fonts. This is effectively the characters in Adobe Standard Encoding, plus ISO Latin 1, plus extra characters available in Lucida Bright. The encoding was generated with the following precepts:

1. The Windows ANSI characters are almost all in their Windows positions, since some Windows users cannot easily re-encode the fonts and it makes no difference on other systems. The only Windows ANSI characters not available are those that make no sense for typesetting—`rubout` (127), `nobreakspace` (160), and `softhyphen` (173). "quotesingle" and "grave" are moved simply because it is so irritating not to have them in TeX positions.
2. The remaining characters are arbitrarily assigned to the lower part of the range, avoiding 0, 10, and 13 to allow for software that assumes that these are used for special operations.
3. Lucida Bright includes some extra text characters; in the hope that other PostScript fonts, perhaps created for public consumption, will implement them, they are included starting at 18.
4. The remaining positions are left undefined for use in (we hope) upward-compatible revisions, if some day more characters are generally available.
5. "hyphen" appears twice for compatibility with both ASCII and Windows.

Fig. 10.10 shows the TeXBase1 layout in a conventional TeX font chart. Tables A.3 and A.4 in Appendix A.6 contrast this encoding with other commonly used ones.

10.5.3 Making TeX metrics for PostScript fonts

Although L^ATeX can deal with different encodings, we still have to decide which one to use in a particular font and create a TeX `tfm` metric. Every PostScript font comes with an `afm` file (see Section 10.3.2 above), and this contains all the information that we need. Various programs are available to convert it to TeX font metric form, as we see next.

10.5.4 AFM to TFM, and VF, conversion tools

There are five commonly used programs that convert `afm` format to `tfm` and possibly create auxiliary virtual fonts at the same time:

1. the Font Manipulation Tools utilities by Y&Y;
2. `afm2tfm`, distributed with dvips;
3. the `fontinst` package by Alan Jeffrey;

	'0	'1	'2	'3	'4	'5	'6	'7	
'0x		.	fi	fl	/	-	L	l	"0x
'01x	.	.		.	-	-	Ž	ž	
'02x	.	i	j	ff	ff	ff	.	.	"1x
'03x									
'04x		!	"	#	\$	%	&	.	"2x
'05x	(	)	*	+	.	.	.	/	
'06x	0	1	2	3	4	5	6	7	"3x
'07x	8	9	:	:	<	=	>	?	
'10x	@	A	B	C	D	E	F	G	"4x
'11x	H	I	J	K	L	M	N	O	
'12x	P	Q	R	S	T	U	V	W	"5x
'13x	X	Y	Z	[	\	]	^	-	
'14x	.	a	b	c	d	e	f	g	"6x
'15x	h	i	j	k	l	m	n	o	
'16x	p	q	r	s	t	u	v	w	"7x
'17x	x	y	z	{		}	~		
'20x		.	f	"	...	†	‡		"8x
'21x	.	%	Š	€	€				
'22x				"	"	.	-	-	"9x
'23x	.	™	š	›	œ		Ÿ		
'24x		i	€	£	¤	¥	¦	§	"Ax
'25x	.	©	ª	«	¬	.	š	.	
'26x	.	±	²	³	.	µ	¶	.	"Bx
'27x	.	ı	°	»	¼	½	¾	¿	
'30x	À	Á	Â	Ã	Ä	Å	Æ	Ç	"Cx
'31x	È	É	Ê	Ë	Ì	Í	Î	Ï	
'32x	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	"Dx
'33x	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß	
'34x	à	á	â	ã	ä	å	æ	ç	"Ex
'35x	è	é	ê	ë	ì	í	î	ï	
'36x	ð	ñ	ò	ó	ô	õ	ö	÷	"Fx
'37x	ø	ù	ú	û	ü	ý	þ	ÿ	
	"8	"9	"A	"B	"C	"D	"E	"F	

Figure 10.10: The TeXBase1 font layout (Lucida Bright).

4. Edmetrics, distributed with the Macintosh Textures implementation of TeX;
5. Y&Y's dviwindo TeX previewer, which has a sophisticated interface to Adobe Type Manager and supports direct generation of TeX font metrics from font instances.

In the next two sections we look briefly at the first two of these and spend some time explaining the third; the details of Macintosh fonts are beyond the scope of this book, and users of Textures should refer to the documentation supplied with their software.

10.5.5 The Y&Y font manipulation tools

The Font Manipulation Tools (available only for DOS) provide a set of very useful tools that go far beyond simple afm to tfm conversion. Some of the utilities are:

- afmtotfm** afm to Windows pfm metric format;
- afmtotfm** afm to TeX font metric;
- composit** add new composite characters to a Type 1 font
- mactopfa** convert Macintosh format Type 1 font to generic pfa format;
- mergepfa** merge Type 1 fonts;
- pfatoafm** create afm metrics from pfa Type 1 font;
- pfatopfb** convert pfa format to binary Type 1 pfb format;
- pfbtomac** convert generic pfb Type 1 font format to Macintosh format;
- pfbtopfa** convert binary pfb format to Type 1 pfa format;
- pfmtoafm** convert Windows pfm font metric to afm format;
- reencode** change the default encoding of a Type 1 font;
- sidebear** change side bearings in a Type 1 font;
- subfont** extract a subset of a Type 1 font;
- tfmtoafm** convert TeX font metric to afm.

Some of these programs are needed because different platforms need fonts or metrics in special forms; most of the transformations are "safe" (i.e., no information is lost), but we should note that none of the transformations to the afm format provide as much information as a real afm file. The most important utilities are those that manipulate the Type 1 font itself; these can be used by TeX users to perform nearly all the functions described earlier in the context of virtual fonts. Instead of maintaining a raw font and performing all the changes via virtual fonts, the Type 1 font itself is transformed and used directly with TeX. This allows us to:

1. re-encode the font;
2. add new composite accented characters;
3. merge fonts to add small caps or characters from expert fonts.

The resulting fonts can be used directly with systems like Adobe Type Manager. This avoids the overhead of virtual fonts, but also means that the user must have access to the Type 1 fonts themselves (many people have them built into printers) and must use non standard

copies. The virtual font system, while imposing a performance overhead, is generally more flexible and portable.

10.5.6 The afm2tfm program

Tom Rokicki's afm2tfm can be used to transform a single afm file to a TeX tfm file. By default it leaves the encoding as it is in the afm file, but it can also write a virtual font that performs some kinds of re-encoding, and can deal with transformations like stretching or slanting characters. It is a Unix or DOS command-line program with a series of switches and is used as follows (with options in square brackets):

```
afm2tfm inputfile [-O] [-u] [-v | -V vplfile] [-e expansion]
[-s slant] [-c capheight] [-p | -t | -T encodingfile] outputfile
```

The program writes (on standard output) the appropriate commands for the dvips map file. The normal result is a tfm file, but some options also generate a vpl file (which you must run through vptovf to generate another tfm file and a vf file.)

The first two options allow the font to be stretched or slanted; they require adding some extra code to the line in the map file, but afm2tfm writes this out for you.

- e *ratio*
Stretch characters horizontally by *ratio*; if the ratio is less than 1.0, the font is condensed.
- s *slant*
Slant characters to the right by *slant*; a negative value makes characters slant to the left. This can be used to make a slanted font or (a curious effect) make an italic font upright.

The next four options relate to encoding of the font, which may or may not generate a virtual font:

- p *encodingfile*
Specify that the PostScript font must be encoded using the vector in *encodingfile* (ligature and kern information is ignored). afm2tfm proceeds on the assumption that this encoding is in place. The encoding must be specified as a header resource file in the map file for the font.
- t *encodingfile*
Use *encodingfile* for the TeX encoding of the font; the ligature and kern information in the file is used.
- T *encodingfile*
Use *encodingfile* for both the PostScript and target TeX encoding.

- v *file*
Generate a virtual property list vpl file as well as a tfm file, making use of any encodings specified.

If you want pseudo-small caps, afm2tfm has to use the virtual font machinery:

- V *file*
Same as -v, but the virtual font generated is a small caps font obtained by scaling uppercase letters (by default, to 0.8 of their natural size) and placing them in the lower-case positions. The process deals with accented letters and kerning properly.
- c *fraction*
When creating a small caps font (with -V), the scaling for the small caps letters is changed from the default 0.8 to *fraction*.

Finally, there are two general options without arguments:

- O Force the character code in the vpl file be given in octal; this is used when processing symbol fonts, to make sure that the symbols at positions normally associated with the letters A-Z and a-z are not labeled as those letters since this confuses the human reader.
- u Use only the characters present in the output encoding. The program normally includes the characters in the output encoding and also any additional characters that may exist in the font.

What is the real meaning of these -p, -t, and -T options? afm2tfm can change the encoding in the output file in two different ways. First, you can ask PostScript to adjust the encoding to suit TeX's layout and simply use standard TeX encoding. For this you use the -p option. This does *not* need a virtual font, but does require you to download the necessary encoding vector with the font. Secondly, you can decide to leave the PostScript font alone and make a virtual font; this allows TeX to stick with its normal encoding but have the characters rearranged by the dvi driver to suit the PostScript font. For this you use the -t option; in this encoding file you can specify ligature pairs and kerning information to be used in addition to the information in the afm file.

To combine the effects of -p and -t, you can use -T; this means that you specify the desired output encoding and afm2tfm creates a virtual font that matches it, and you also say that the PostScript font is encoded in the same way (you still have to download the encoding with the font). The difference between this and -p is that the ligature and kern information is added into the virtual font. This option thus gives maximum flexibility and is the one normally used for text fonts.

afm2tfm's encoding files have precisely the same format as an encoding vector in any PostScript font. Here is a skeleton encoding:

```
/MyEncoding [ % exactly 256 entries, each starting with /
/Alpha /Beta /Gamma /Delta ...
```

```
... /A /B ... /Z
/aacute /logicalnot /eth ] def
```

Comments, which start with a percent sign and continue to the end of the line, are ignored unless they start with LIGKERN. The first “word” of the file, which must start with a forward slash (a PostScript literal name), defines the name of the encoding. The next word must be an left bracket [. Following that must be precisely 256 character names (use /.notdef for any that you do not want defined). Finally, there must be a matching right bracket]. A final def token is not checked for, but it is normally required by PostScript interpreters. All names are case-sensitive.

Any ligature or kern information is given in the comments. If the first word after the percent sign is LIGKERN, then the entire rest of the line is parsed for ligature and kern information. This ligature and kern information is given in groups of words, each terminated by a semicolon (with a space before and after it, unless it occurs at the end of a line.)

Three types of information may be specified in these LIGKERN statements: ligature pairs, kerns to remove or ignore, and the character value of this font’s TeX boundary character. Which of the types the particular set of words corresponds to is automatically determined by the allowable syntax. The boundary character is specified as || and can be set with a command like || = 39 ;.

To indicate removal of a kern, give the names of the two characters (without the leading slash) separated by {}, as in one {} one ;. This is similar to the use of {} in a TeX file to turn off ligatures or kerns at a particular location. Either or both of the character names can be given as *, which is a wild card matching any character; thus, all kerns can be removed with * {} * ;.

To specify a ligature, give the names of the two characters, followed by the ligature “operation” (as in METAFONT), followed by the replacing character name. Either (but not both) of the first two characters can be || to indicate a word boundary. Normally the “operation” is =:, meaning that both characters are removed and replaced by the third character, but by adding | characters on either side of the =:, you can specify that one of the two leading characters should be retained. In addition, by suffixing the ligature operation with one or two > signs, you can indicate that the ligature scanning operation should skip that many characters before proceeding. This works just as in METAFONT. A typical ligature might be specified with ff i =: ffi ;. A more convoluted ligature is one one |=:|>> exclam ;, which indicates that every pair of adjacent 1s should be separated by an exclamation point, and then two of the resulting characters should be skipped over before continuing searching for ligatures and kerns. No more >’s than |’s are allowed in a ligature operation, so there are a total of eight possible ligature operations:

```
=: |=: |=:> =:| |=:> |=:| |=:> |=:|> |=:|>>
```

which correspond to the eight LIG property variants in pl files described on p. 366.

The default set of ligatures and kerns built into afm2tfm are designed to provide the same effects as Knuth created for Computer Modern, to maximize the portability of TeX

documents; they could be specified with the following lines in an encoding file:

```
% LIGKERN question quoteleft =: questiondown ;
% LIGKERN exclam quoteleft =: exclamdown ;
% LIGKERN hyphen hyphen =: endash ; endash hyphen =: emdash ;
% LIGKERN quoteleft quoteleft =: quotedblleft ;
% LIGKERN quoteright quoteright =: quotedblright ;
% LIGKERN space {} * ; * {} space ; 0 {} * ; * {} 0 ;
% LIGKERN 1 {} * ; * {} 1 ; 2 {} * ; * {} 2 ; 3 {} * ; * {} 3 ;
% LIGKERN 4 {} * ; * {} 4 ; 5 {} * ; * {} 5 ; 6 {} * ; * {} 6 ;
% LIGKERN 7 {} * ; * {} 7 ; 8 {} * ; * {} 8 ; 9 {} * ; * {} 9 ;
```

An example of the use of afm2tfm with the cheq font is given in Section 8.1.5.

10.5.7 The fontinst package

The most flexible way to create virtual fonts for TeX is Alan Jeffrey’s fontinst package. This implements in TeX itself a language that allows the user to describe the input font metrics, procedures to be followed, and desired result; TeX is run on the specification you write and it reads, e.g., afm files (it can also read TeX property lists files) and writes a virtual font in the ASCII vpl form. These are turned into production vf files with the standard vptovf program. In this section we describe only the high-level fontinst commands and give examples of the other commands; the details of the lower-level parts of fontinst are given in Appendix A.7.

Three types of files are used by the fontinst package:

- *fontinst files* contain the TeX commands you write to install fonts;
- *encoding files*, such as t1.etx, contain information about an encoding, including the code table, ligatures, and font dimensions;
- *metric files*, or character description files, contain information about sets of glyphs (fonts), including glyph dimensions, composite characters, and kerning.

For example, latin.mtx is a *metric file* that describes the *set* of Latin characters we may find in a font, while t1.etx gives the *order* in which they should appear and ptmr8r.mtx gives the actual set of available characters.

The fontinst macros are loaded with

```
\input fontinst.sty
```

and the file is terminated with \bye. A fontinst control file normally consists of either a high-level form for the straightforward case of whole Latin font families:

```
\latinfamily{family name}{extra .fd code}
```

or one or more structures of the form:

```
\installfonts
install commands
\endinstallfonts
```

The `\latinfamily` interface

Many users do not want to do anything more complicated with fontinst than create PSNFSS-like metrics for a new font family. The `\latinfamily` command performs this function. It is designed to work with sets of afm files named according to the *Fontname* scheme, and is supplied with the foundry and font family name only. With this information it attempts to execute a `\installfont` command for every combination of weight and series for which it can find an appropriate afm file; it performs a re-encode to TeXBase1 (8r) and generates output metrics for both OT1 and T1 encodings. Expert sets, if available, can also be used where possible (for small caps, extra ligatures, etc.) by appending an “x” to the family name; if a “g” is appended to the name, then a set of fonts using old-style digits is built.

A simple fontinst command file need, therefore, be no more complicated than this example, which does all the necessary work for Times (ptm), Helvetica (phv) and Courier (pcr):

```
\input fontinst.sty
\latinfamily{ptm}{}
\latinfamily{phv}{}
\latinfamily{pcr}{\hyphenchar\font=-1 }
\bye
```

The `\installfont` interface

Now we turn to the *install commands* that describe the fonts, glyphs and encodings used to build fonts.

```
\installfamily{encoding}{family}{fd commands}
```

This sets up a L^AT_EX family with the given encoding and family; for example, to initiate the Cork-encoded Optima family, you say:

```
\installfamily{T1}{pop}{}

```

The *fd commands* are L^AT_EX commands that are written into the font description file generated for the family; and are executed every time a font in that family is loaded. Thus to stop the Courier family from being hyphenated, you might say:

```
\installfamily{T1}{pcr}{\hyphenchar\font=-1}
```

```
\substitutesilent{old series or shape}{new series or shape}
\substitutenoisy{old series or shape}{new series or shape}
```

The above commands set up L^AT_EX font description commands to perform shape or series substitution.

```
\transformfont{target font}{generation command}
```

Creates an internal mtx *target font* by running the command *generation command*. This is one of the following:

```
\scalefont{integer expression}{font}
```

Scales the font in both the *x* and *y* dimensions by *integer expression*.

```
\xscalefont{integer expression}{font}
```

Scales the font in the *x* dimension by *integer expression*.

```
\yscalefont{integer expression}{font}
```

Scales the font in the *y* dimension by *integer expression*.

```
\slantfont{integer expression}{font}
```

Slant the font by *integer expression*.

```
\reencodefont{etx file}{font}
```

Re-Encode the *font* to the encoding in the *etx file*.

The *font* in each of the above can be itself a call to one of the other transformations, but is most commonly

```
\fromafm{afm}
```

which reads an afm file and returns a form that can be manipulated by fontinst.

An example of a font transformation is:

```
\transformfont{pplr8r}{\reencodefont{8r}{\fromafm{pplr8a}}}
```

This reads the original Adobe afm file for Palatino (pplr8a.afm) and creates an mtx font metric description pplr8r.mtx by rearranging the characters to suit the 8r encoding. pplr8r can now be used in `\installfont` commands. It is important to remember, however, that this re-encoding (or other transformations) is applied to the *metrics* only; the font

itself must also be transformed in the same way when it is loaded (Section 11.2.4 considers how this is done when using dvips.)

```
\installfont{font}{files}{etx file}{encoding}{family}{series}{shape}{size}
```

This produces a TeX font called *font* from the comma separated list of *files*, which should be *mtx*, *afm*, or *pl* files. The TeX font is encoded using *etx file*.*etx*, and can be accessed in L^ATeX with the given *encoding*, *family*, *series*, *shape*, and *size*. For example, to install the Cork-encoded Times Roman font, you could say:

```
\installfont{ptmr8t}{ptmr8r,latin}{T1}{T1}{ptm}{m}{n}{}
```

This takes the list of *available* metrics from the *ptmr8r.mtx* file, looks at the *latin.mtx* description of how to build characters, and creates entries in the output for all characters listed in the *t1.e* *etx* encoding file (the encoding name is given in upper-case, but a filename in lower-case is searched for). The same font is used for all sizes.

Metric files

The metric file *latin.mtx* is worth considering in some detail. It defines all the characteristics of commonly used fonts and serves as an example of many fontinst techniques. Metric files can contain anything (documentation, etc.) up to the first `\metrics` command (which means we are ready to start):

```
1 \metrics
2 \needsfontinstversion{1.334}
```

One of the important aspects of fontinst is that it is written using familiar TeX macro programming; here is a macro specifying how to make a small caps character, by putting a little space on either side of a scaled-down glyph:

```
3 \setcommand\cscglyph#1{
4   \mover{\int{smallcapsextraspace}}
5   \glyph{#1}{\int{smallcapsscale}}
6   \mover{\int{smallcapsextraspace}}
7 }
```

All the TeX font dimensions and other parameters can be set to absolute or calculated values; these can, of course, be overridden in later control files. All dimensions are in relation to a grid of 1000 × 1000.

```
8 \setint{italicslant}{0}
9 \setint{xheight}{\height{x}}
10 \setint{capheight}{\height{A}}
11 \setint{ascender}{\height{d}}
12 \setint{descender}{\depth{g}}
```

```
13 \setint{underlinethickness}{40}
14 \setint{visiblespacedepth}{200}
15 \setint{visiblespacewidth}{400}
16 \setint{visiblespacesurround}{50}
17 \setint{smallcapsscale}{800}
18 \setint{smallcapskerning}{900}
```

The complexity of the kerning is up to the author of the control file; here are some of fontinst's defaults in *latin.mtx*:

```
19 \setleftrighthkerning{Aacute}{A}{1000}
20 \setleftrighthkerning{Abreve}{A}{1000}
21 ...
22 \setleftkerning{ffi}{f}{1000}
23 \setleftkerning{ffl}{f}{1000}
24 ...
25 \setleftrighthkerning{Aacutesmall}{Aacute}{\int{smallcapskerning}}
26 \setleftrighthkerning{Abrevesmall}{Abreve}{\int{smallcapskerning}}
27 \setleftrighthkerning{Acircumflexsmall}{Acircumflex}{\int{smallcapskerning}}
```

Next comes the list of characters needed in Latin alphabets; the first set is defined with a macro `\unfakable`, which means that they generate a warning if they are not in a font; the order of these is unimportant.

```
28 \unfakable{Gamma}
29 \unfakable{Delta}
30 \unfakable{Theta}
31 ...
32 \unfakable{guillemotleft}
33 \unfakable{guillemotright}
34 \unfakable{endash}
35 \unfakable{emdash}
36 ...
37 \unfakable{germandbls}
38 \unfakable{lslashslash}
```

The remaining characters are given a definition that is used if the font does not provide a real glyph:

```
39 \setglyph{fi}
40   \glyph{f}{1000}
41   \mover{\kerning{f}{i}}
42   \glyph{i}{1000}
43 \endsetglyph
44 \setglyph{ff}
45   \glyph{f}{1000}
46   \mover{\kerning{f}{f}}
```



```

47 \glyph{f}{1000}
48 \endsetglyph
49 ...
50 \setglyph{Abreve}
51 \topaccent{A}{breve}{500}
52 \endsetglyph
53 \setglyph{Aogonek}
54 \botaccent{A}{ogonek}{900}
55 \endsetglyph
56 ...
57 \setglyph{Qsmall}
58 \cscglyph{Q}
59 \endsetglyph

```

Finally, the floating accents are revisited to give them the zero depth TeX needs; the following commands also show how existing definitions can be overridden:

```

60 \setcommand\zerodepth#1{
61 \resetglyph{#1}
62 \glyph{#1}{1000}
63 \resetdepth{0}
64 \endresetglyph
65 }
66 \zerodepth{acute}
67 \zerodepth{grave}
68 \zerodepth{caron}
69 \zerodepth{dieresis}
70 ...
71 \endmetrics

```

Everything after `\endmetrics` is ignored by fontinst, and thus textual documentation can follow this command.

Encoding files

An *encoding file* consists mainly of a set of `\setslot` commands saying what is to appear; these commonly specify the upper- and lower-case letters for the slot and create ligatures. Some examples are given below. Note that the order *is* important, as the position of, e.g., the letter A is determined solely by its position in the encoding file (slots can be skipped and the next number set explicitly with `\nextslot`, if necessary).

```

\setslot{\uc{A}{a}}
\comment{The letter '{A}' .}
\endsetslot
\setslot{hyphen}
\ligature{LIG}{hyphen}{rangedash}
\ligature{LIG}{hyphenchar}{hyphenchar}

```

```

\comment{The hyphen '-'.}
\endsetslot
\setslot{\lc{fl}{fl}}
\comment{The 'fl' ligature. It should be two characters wide in a
monowidth font.}
\endsetslot
\setslot{dollar}
\comment{The dollar sign '\$'.}
\endsetslot
\setslot{\lc{Breve}{breve}}
\comment{The breve accent '\u a'.}
\endsetslot

```

While most fontinst users have no need to understand all the `latin.mtx` and encoding files, the extracts above give some flavor of the system.

Examples of `\installfont` command files

Let us now examine some examples of real fontinst control files. Our first example is designed to process part of the Adobe Minion family. First, we load the macros, set up a `\installfont` group, and initialize the encoding and family:

```

1 \input fontinst.sty
2 \needsfontinstversion{1.317}
3 \installfont
4 \installfamily{T1}{pmn}{ }

```

Next we put in some common substitutions to say that we do not have “bold extended”, so “bold” should be used, and that we do not have “upright italic” so “italic” should be used.

```

5 \substitutesilent{bx}{b}
6 \substitutenoisy{ui}{it}

```

Now there are three sets of fonts to create: the normal font, a display size, and an extra bold. In each group we first re-encode a base font and then use it in `\installfont`.

```

7 \transformfont{pmnr8r}{\reencodefont{8r}{\fromafm{pmnr8a}}}
8 \transformfont{pmnr8r}{\reencodefont{8r}{\fromafm{pmnr8a}}}
9 \installfont{pmnr8t}{pmnr8a,pmnr8x,latin}{T19}{T1}{pmn}{m}{n}{<-14>}
10 \installfont{pmnr8t}{pmnr8a,pmnr8x,latin}{T19}{T1}{pmn}{m}{it}{<-14>}
11 \installfont{pmnr8t}{pmnr8a,pmnr8x,latin}{T1c}{T1}{pmn}{m}{sc}{<-14>}

```

You can see that each font is derived from three metric files: the base font, an expert set companion font (which contains extra ligatures, old-style digits, etc.), and the Latin character set. The order in which these appear is crucial; the commands in `latin.mtx` depend on the previous definition of a set of available characters, so that it can examine values and

override them if need be. The target encoding is suffixed with a 9, meaning that the old-style digits should be used by default.

The first set of fonts specified were for use only up to 14 point; a special Display set is used at larger sizes, and the resulting font description file reflects this.

```

12 \transformfont{pmnrd8r}{\reencodefont{8r}{\fromafm{pmnrd8a}}}
13 \transformfont{pmnrdi8r}{\reencodefont{8r}{\fromafm{pmnrdi8a}}}
14 \installfont{pmnrd8t}{pmnrd8a,pmnrd8x,latin}{T19}{T1}{pmn}{m}{n}{<14->}
15 \installfont{pmnrdi8t}{pmnrdi8a,pmnrdi8x,latin}{T19}{T1}{pmn}{m}{it}{<14->}
16 \installfont{pmnrdc8t}{pmnrd8a,pmnrd8x,latin}{T1c}{T1}{pmn}{m}{sc}{<14->}

```

Finally, we provide support for an extra bold font

```

17 \transformfont{pmnc8r}{\reencodefont{8r}{\fromafm{pmnc8a}}}
18 \installfont{pmnc8t}{pmnc8a,pmnc8x,latin}{T19}{T1}{pmn}{eb}{n}{-}
19 \endinstallfonts
20 \bye

```

The result of running the above script through TeX would be a set of p1 property list files, vp1 virtual font property lists, and an fd file for L^AT_EX. The vp1 files should be processed using vptovf to create tfm and vf files, and the p1 files should be processed using tftopl.

Our second example builds some OT2-encoded Cyrillic fonts using Monotype Times; as before, we start by loading fontinst and defining the version we need:

```

1 \input fontinst.sty
2 \needsfontinstversion{1.317}

```

Now we take advantage of using TeX by encapsulating in a macro what we want done for each member of the font family. For a given shape and series combination, we want to load first the Cyrillic font (suffix z) and then the corresponding Latin one (for numerals, punctuation, etc.); any characters needed for the encoding that were not in the Cyrillic font can be picked up in the Latin font. The latter is first re-encoded to the 8r standard.

```

3 \def\TimesCyrillicFont#1#2#3#4{% Fontname name, series, shape,
4 % alternate name
5 \transformfont{#48r}{\reencodefont{8r}{\fromafm{#48a}}}
6 \installfont{#17k}
7 {#1z,#48r,monocyr}
8 {monoot2}{OT2}{mnt}{-}{#3}{-}
9 }

```

The metric file monoot2.mtx defines the order in which the characters should appear, e.g.:

```

\nextslot{060}\setslot{guillemotleft}\endsetslot
\nextslot{061}\setslot{dotlessi}\endsetslot
\nextslot{062}\setslot{guillemotright}\endsetslot
\nextslot{063}\setslot{question}\endsetslot

```

```

\nextslot{064}\setslot{brave}\endsetslot
\nextslot{065}\setslot{As}\endsetslot
\nextslot{066}\setslot{Buki}\endsetslot
\nextslot{067}\setslot{Zui}\endsetslot

```

while monocyr.mtx defines the available character set, where the names are as used in Monotype's afm files, e.g.:

```

\unfakable{guillemotright}
\unfakable{question}
\unfakable{cyrbreve}
\unfakable{As}
\unfakable{Buki}
\unfakable{Zui}
\unfakable{Dobro}

```

Having set up the command, we can now run it for normal, italic, bold, and bold italic and end the job:

```

10 \installfonts
11 \installfamily{OT2}{mnt}{-}
12 \TimesCyrillicFont{mnt}{m}{n}{mnt}
13 \TimesCyrillicFont{mnt}{m}{it}{mnti}
14 \TimesCyrillicFont{mnt}{b}{n}{mntb}
15 \TimesCyrillicFont{mnt}{b}{it}{mntbi}
16 \endinstallfonts
17 \bye

```

Our third example is a command file to build math fonts using a mixture of Minion Multiple Master and Lucida New Math (originally designed for this book but not ultimately used). Section 10.1.1 on p. 354 describes the mathptm package. It consists of a package file redefining small parts of math and setting up the new font families, but the bulk of the work is done in virtual fonts. These were created by a fontinst script that makes sure that the characters have all their right dimensions and characteristics. The user of mathptm need not be concerned about this, but it is worth noting that it is fairly straightforward to use the same technique to create other mixtures, like Computer Modern math with Garamond, or our present example, derived from Alan Jeffrey's mathptm script.

As before, after loading fontinst, we define a macro to do most of the work:

```

1 \input fontinst.sty
2 \needsfontinstversion{1.5}
3 \def\MathSize#1#2#3#4{% start of size range,
4 % end of size range,
5 % font size, scaling

```

What is the body of this macro trying to achieve? We want to create the math symbol font (OML encoding) and math alphanumeric font (OT1 encoding) for a new family called `zmnLuc`. For each of these, we take a scaled down Lucida New Math font and a Minion Multiple Master instance and use characters from both. Since we have Minion instances at six different sizes, we want to use the best one at each point, while Lucida can simply be scaled to fit any size.

As usual, we start by building 8r-encoded versions of the text fonts:

```
5 \transformfont{zmn8r#3}{\reencodefont{8r}{\fromafm{zmn8a#3}}}
6 \transformfont{zmmi8r#3}{\reencodefont{8r}{\fromafm{zmmi8a#3}}}
7 \transformfont{zmbi8r#3}{\reencodefont{8r}{\fromafm{zmbi8a#3}}}
8 \transformfont{zmb8r#3}{\reencodefont{8r}{\fromafm{zmb8a#3}}}
```

and call `\installfont`

```
9 \installfont{zmnLucrm#3}
10 {kernoff,hLCrma scaled #3,kernon,unsetluc,zmmi8r#3
11 scaled 1075,lmathit,lhax}
12 {LOML}{OML}{zmnLuc}{m}{it}{<#1-#2>}
```

The important feature here is the list of metric files called on to make an OML-encoded font; in order, these are:

1. `kernoff.mtx`; this merely contains commands to disable kerning, which we do not want in the characters drawn from a math font;
2. `hLCrma.mtx`, the character set of `LucidaNewMath-AltItalic`;
3. `kernon.mtx`, which restores kerning;
4. `unsetluc.mtx`; this *deletes* characters picked up from the Lucida font that we want to use from Minion. It consists of a set of lines like:

```
\unsetint{xheight}
\unsetint{ascender}
\unsetint{descender}
\unsetint{capheight}
\unsetint{italicslant}
\unsetglyph{A}
\unsetglyph{B}
...
```

5. The Minion font, whose characters fill the slots we just undefined in the previous stage;
6. `lmathit.mtx`, which defines the target character set;
7. `lhax.mtx`, which does some extra manipulation of the characters, changing kerns, moving characters around, etc.; this attempts to make characters from an ordinary text

font have the characteristics of a math font, and is entirely the work of Alan Jeffrey.

Those files which are not part of the standard fontinst distribution are available in the CTAN archives, `info/lgc`.

The job is repeated for a bold version:

```
14 \installfont{zmnLucbf#3}
15 {kernoff,hLCdima scaled #3,kernon,unsetluc,zmbi8r#3
16 scaled 1075,lmathit,lhax}
17 {LOML}{OML}{zmnLuc}{b}{it}{<#1-#2>}
```

Then a similar job for the OT1 encoding, in italic, upright, and bold. This is simpler since the target is the Latin character set, but we can pull some useful characters (such as Greek) from Lucida:

```
18 \installfont{zmnLucmi#3}
19 {hlcrima scaled #3,unsetluc,zmmi8r#3 scaled 1075,latin}
20 {OT1I}{OT1}{zmnLuc}{m}{it}{<#1-#2>}
21 \installfont{zmnLucmn#3}
22 {hlcrm scaled #3,unsetluc,zmn8r#3 scaled 1075,latin}
23 {OT1}{OT1}{zmnLuc}{m}{n}{<#1-#2>}
24 \installfont{zmnLucbn#3}
25 {hlcdm scaled #3,unsetluc,zmb8r#3 scaled 1075,latin}
26 {OT1}{OT1}{zmnLuc}{b}{n}{<#1-#2>}
27 }
```

Now we are ready to build fonts; we define two target sets, in the OT1 and OML encodings, and call the main building macro for each of the desired size ranges:

```
28 \installfonts
29 \installfamily{OT1}{zmnLuc}{
30 \installfamily{OML}{zmnLuc}{\skewchar\font=127}
31 \MathSize{6.5}{6}{990}
32 \MathSize{6.5}{7.5}{970}
33 \MathSize{7.5}{8.5}{970}
34 \MathSize{8.5}{10.5}{940}
35 \MathSize{10.5}{12.5}{930}
36 \MathSize{12.5}{14.5}{930}
37 \MathSize{14.5}{17}{920}
38 \endinstallfonts
39 \bye
```

One more task remains: to tell $\TeX$ about the new math fonts. This is a bit more complicated than simply redefining `\rmdefault`. The minimum necessary is to define the “letters”, “operators”, “symbols”, and “largesymbols” math groups, and set up bold and normal

versions of the “operators” and “letters”; this assumes that a normal text family named `zmn` has already been set up:

```
\DeclareSymbolFont{letters}{OML}{zmnluc}{m}{it}
\DeclareSymbolFont{operators}{OT1}{zmn}{m}{n}
\DeclareSymbolFont{symbols}{OMS}{hlcy}{m}{n}
\DeclareSymbolFont{largesymbols}{OMX}{hlcv}{m}{n}
\SetSymbolFont{letters}{normal}{OML}{zmnluc}{m}{it}
\SetSymbolFont{letters}{bold}{OML}{zmnluc}{b}{it}
\SetSymbolFont{operators}{bold}{OT1}{zmn}{b}{n}
\SetSymbolFont{operators}{normal}{OT1}{zmn}{m}{n}
```

In practice, we can use more of the Lucida math fonts by making some additional fonts available, namely upright math, and an extra symbol font (full of arrows, hence the name used here):

```
\DeclareSymbolFont{mathupright}{OML}{hlcm}{m}{n}
\DeclareFontEncoding{LMR}{}{}
\DeclareFontSubstitution{LMR}{hlcm}{m}{n}
\DeclareSymbolFont{arrows}{LMR}{hlcm}{m}{n}
\DeclareSymbolFont{boldarrows}{LMR}{hlcm}{b}{n}
```

Since the Lucida arrows font contains a complete “blackboard bold” font, we can map it to the `\mathbb` command for use as a text-like font:

```
\DeclareSymbolFontAlphabet{\mathbb}{arrows}
```

We can also redefine some common symbol names to fetch them from the “arrows” font:

```
\DeclareMathSymbol{\rightleftharpoons}{3}{arrows}{"7A}
\DeclareMathSymbol{\leftrightharpoons}{3}{arrows}{"79}
```

and define some new ones:

```
\DeclareMathSymbol{\dashuparrow}{0}{arrows}{"39}
\DeclareMathSymbol{\dashdownarrow}{0}{arrows}{"3B}
```

The math upright font in Lucida contains some useful upright Greek to which we can get access:

```
\DeclareMathSymbol{\upalpha}{\mathalpha}{mathupright}{11}
\DeclareMathSymbol{\upbeta}{\mathalpha}{mathupright}{12}
```

Fig. 10.11 shows the resulting font mix.

The default math mode font is *math italic*. This should not be confused with ordinary *text italic*. `\mathbf` produces **bold-face roman** letters. If you wish to have **bold-face math italic** letters and bold-face Greek letters and mathematical symbols, use the `\boldmath` command *before* going into math mode. This changes the default math fonts to bold. Greek is available in upper and lower case: $\alpha, \beta, \Gamma, \Delta \dots \omega, \Omega$

normal: $x = 2\pi \Rightarrow x \simeq 6.28$

mathbf: $x = 2\pi \Rightarrow x \simeq 6.28$

boldmath: $x = 2\pi \Rightarrow x \simeq 6.28$

There is also a calligraphic font for upper-case letters; these are produced by the `\mathcal` command: $\mathcal{A}\mathcal{B}\mathcal{C}\mathcal{D}\mathcal{E}$

$$\phi(t) = \frac{1}{\sqrt{2\pi}} \int_0^t e^{-x^2/2} dx \quad (1)$$

$$\prod_{j \geq 0} \left(\sum_{k \geq 0} a_{jk} z^k \right) = \sum_{k \geq 0} z^n \left(\sum_{\substack{i_0, k_1, \dots, \geq 0 \\ i_0 + k_1 + \dots = n}} a_0 k_0 a_1 k_1 \dots \right) \quad (2)$$

$$\pi(n) = \sum_{m=2}^n \left[\left(\sum_{k=1}^{m-1} \lfloor (m/k) / \lceil m/k \rceil \rfloor \right)^{-1} \right] \quad (3)$$

$$\overbrace{\{a, \dots, a, b, \dots, b\}}^{k \text{ } a\text{'s} \quad l \text{ } b\text{'s}} \quad (4)$$

$k+1$ elements

$$\begin{array}{l} \nearrow \mu^+ + \nu_\mu \\ W^+ \rightarrow \pi^+ + \pi^0 \\ \rightarrow \kappa^+ + \pi^0 \\ \searrow e^+ + \nu_e \end{array}$$

$$\pm \frac{\begin{vmatrix} x_1 - x_2 & y_1 - y_2 & z_1 - z_2 \\ l_1 & m_1 & n_1 \\ l_2 & m_2 & n_2 \end{vmatrix}}{\sqrt{\begin{vmatrix} l_1 & m_1 \\ l_2 & m_2 \end{vmatrix}^2 + \begin{vmatrix} m_1 & n_1 \\ n_1 & l_1 \end{vmatrix}^2 + \begin{vmatrix} m_2 & n_2 \\ n_2 & l_2 \end{vmatrix}^2}}$$

Figure 10.11: Math test with mixture of Minion Multiple Master and Lucida New Math.

10.6 Multiple Master fonts — a case study

Multiple Master fonts are usually manipulated in one of three ways:

1. entirely behind the scenes by software like Adobe Acrobat, which generates font instances to match some criteria;
2. with the advanced version of Adobe Type Manager, which allows the user to experiment interactively with settings, and then create an instance;
3. by direct manipulation of metric files and writing appropriate PostScript code.

For L^AT_EX users, the second method is the simplest, since the end result is an *afm* file and the right fragment of PostScript code (a “stub”) to download with the font itself; the *afm* file can be converted as usual to get a T_EX metric. If L^AT_EX is being run in an environment with a previewer and/or printer driver making use of ATM, this system is very convenient. The commercial Windows T_EX system from Y&Y fully supports and documents the procedure, so the only issue for users of that setup is the choice of values for the font instance when using ATM.

Multiple Master users on a computer platform without ATM (such as Unix or DOS systems) can also generate font instances by using the font metric files directly and the free Ghostscript program. The remainder of this section explores the details, which are of interest to those who simply want to understand what is happening, or want to calculate font instances automatically in an algorithmic rather than visual way.

Multiple Master fonts come with a set of master *afm* files called AMFM (Adobe Master Font Metrics). These files contain information about the number of master designs, the number of axes, the `BlendDesignPositions` and `BlendDesignMap` arrays, as well as the names and `weightvector` for the master designs, from which all font instances are derived.

To get the actual metric information for the characters in a font instance, we must combine the metric information of the master designs (eight, in the case of Minion). To do this we need to calculate the `weightvector` for the given instance. Starting from design-coordinate space, we can use the `NormalizeDesignVector` operator to transform to the normalized coordinate space; from there with the `ConvertDesignVector` operator we obtain the weight vector. These two operators are peculiar to a font (since they depend on the master designs) and are present in the Multiple Master font dictionary. We can decode the PostScript code for calculating the weight vector and translate it into another computer language, and then use the procedure to combine the values in the *afm* files for the master designs to calculate the values needed for the font instance. For instance, in the case of the MinionMM font, the PostScript code defines the eight components of the weight vector as follows:

$$\begin{aligned} w_1 &= xyz & w_2 &= (1-x)yz \\ w_3 &= x(1-y)z & w_4 &= (1-x)(1-y)z \\ w_5 &= xy(1-z) & w_6 &= (1-x)y(1-z) \\ w_7 &= x(1-y)(1-z) & w_8 &= 1 - \sum_{n=1}^7 w_n \end{aligned}$$



Figure 10.12: Minion instances from opposite ends of the optical size axis set at the same size (exaggerated).

where x is the normalized weight, y the normalized width, and z the normalized optical size. These eight numbers w_i allow the calculation of all parameters needed in an *afm* file for a font instance. We read each parameter value in turn in the eight master design *afm* files and apply the relevant weight; the weighted sum thus obtained is the desired interpolated value of the given parameter for the font instance.

Myriad is a sans-serif companion font to Minion that has two design axes and four master designs. The weights for deriving font instance parameters in normalized coordinate space are given by:

$$\begin{aligned} w_1 &= 1-x & w_2 &= (1-x)y \\ w_3 &= x(1-y) & w_4 &= xy \end{aligned}$$

where x is the normalized weight and y the normalized width. The corresponding mapping parameters between design space and normalized coordinates are:

```
BlendDesignPositions [ [0 0] [1 0] [1 1] [1 1] ]
BlendDesignMap [[ [215 0] [830 1] ] [ [300 0] [700 1] ] ]
BlendAxisTypes [ /Weight /Width ]
```

Now we can extract any of the bounding box and kern entries for a given font instance by getting the element in question from the eighth (or four, in the case of Myriad) master files and calculating the interpolated value. To make matters simpler, an explicit example is given for the Myriad font, since it involves only four numbers in each case.

When the instance *afm* files have been created, a suitable metric for L^AT_EX can be built in the normal way.

10.6.1 Generating instance files using Ghostscript

The ideas outlined above are instantiated as a Unix shell script (makeinstance) and an adaptation of an afm-parsing program distributed by Adobe (mmafm); these are available in the CTAN archives, fonts/utilities/mm. The main script takes the following actions:

1. creates a small PostScript file to invoke Multiple Master operators with values passed to the script;
2. runs Ghostscript on this file to derive normalized weights and writes them to a temporary file; note that this must be version 3.33 or later of Aladdin Ghostscript—earlier versions of the program did not have the code to deal with Multiple Master fonts;
3. runs our mmafm program to read master afm files and writes a new instance afm file.

The result is an afm file and a pro header file. We can then use the afm file to create TeX metrics for 8r base-encoded metrics and EC-encoded virtual fonts for actual use; a dvips map entry is needed, with the name of the header file and the real (Multiple Master) font name.

Since the script is quite short, we reproduce it here:

```
#!/bin/sh
#
#Usage: makeinstance fontname instancename design-space-variables 1 to 4
X=$2.control
cat > $$instance <<CLOSE
/mmfile (mmfile) (w) file def
/str 20 string def
/MASTERFONT findfont dup begin [
V1 V2 V3 V4 NormalizeDesignVector ConvertDesignVector
counttomark 1 sub -1 0 {
index str cvs mmfile exch writestring
mmfile (\n) writestring} for
mmfile closefile
] end
CLOSE
#-----
MASTERFONT="$1"
V1="$3"
V2="$4"
V3="$5"
V4="$6"
sed -e "s/MASTERFONT/$MASTERFONT/" \
    -e "s/V1/$V1/" \
    -e "s/V2/$V2/" \
    -e "s/V3/$V3/" \
    -e "s/V4/$V4/" \
```

```
< $$instance > $$ps
rm $$instance
gs -dNODISPLAY 'kpsewhich dvips_header $1.PFA' $$ps quit.ps
rm $$ps
#-----
# build control file for mmafm
#-----
# number of weights
wc -l mmfile | awk '{print $1}' > $N
cat $1.afmnames >> $N
echo $2 >> $X
cat mmfile >> $X
#-----
rm mmfile
mmafm $X | \
sed -e "/~FontName/s/~FontName .*/FontName $1-$3-$4-$5-$6/" \
    -e "/~FontName/s/~$/"/ > $2.afm
#-----
echo "/$2 /$1 findfont dup begin [" > $2.pro
echo "    $V1 $V2 $V3 $V4 NormalizeDesignVector ConvertDesignVector" >> $2.pro
echo "    ] end makeblendedfont definefont pop " >> $2.pro
#-----
exit
```

A call to this script might have the following parameters:

```
MinionMM zmn18ac6 360 460 6
```

This creates a metric file called zmn18ac6, using the *Fontname* scheme to name “Minion, light weight, 8a-encoded, condensed, at 6pt design size”. The entry in the dvips map file for it should read:

```
zmn18rc6 zmn18ac6 "TeXBase1Encoding ReEncodeFont
<8r.enc <MinionMM.pfb <zmn18ac6.pro
```

where the prologue file zmn18ac6.pro contains instructions to the PostScript interpreter on how the given font instance should be generated from the Multiple Master font code in MinionMM.pfb. The file zmn18ac6.pro contains the code:

```
/zmn18ac6 /MinionMM findfont
dup begin [
    360 460 6 NormalizeDesignVector
    ConvertDesignVector
] end makeblendedfont definefont pop
```

Note the presence of the *NormalizeDesignVector* and *ConvertDesignVector* PostScript operators described earlier.

In addition, we need font description files to tell L^AT_EX how to match up the various weight and width instances created to its notions of series and shape. The only complication here is that the Minion font has an optical size axis, and we need L^AT_EX to use different instances at different user sizes:

```
\DeclareFontShape{T1}{zmn}{lc}{n}{
  <-7>zmn18tc6
  <7-10>zmn18tc8
  <10-15>zmn18tc11
  <15->zmn18tc18}{}
```

The effect of the optical sizes is demonstrated by Fig. 10.12 on p. 407, which shows the 6pt and 18pt instances scaled to the same size. The differences in design are as apparent as the corresponding examples from Computer Modern would be.

The keen T_EX expert using Unix Web2c systems can create a MakeTeXTFM script⁵ to apply the programs on the fly from within T_EX; this would allow T_EX macros to control the parameters of the instance creation. The main problem with this, if we use the fontinst package described in this chapter, is that it is far too slow (several minutes to create for each font) but this could be overcome by using other tools.

10.6.2 The Multiple Master setup of this book

In typesetting this book, we tried to match up the L^AT_EX font selection scheme's standard list of shapes and series (Goossens et al., 1994, p. 190) to corresponding Minion and Myriad font instances and name them according to the *Fontname* scheme.

Minion has three axes: the weight ranges from 345 to 620, the width ranges from 450 to 600, and the size varies from 6 to 72. In Myriad there are only two axes: the weight ranges from 215 to 830 and the width ranges from 300 to 750. If we take the *Fontname* ranges of weight as our starting point and simply allocate values at equal intervals, the result is unsatisfactorily bold, as the “medium” point which we preferred was not in the middle of the font range; similarly, the simple allocation of widths yields an over-squashed default. Clearly the values must be chosen as part of a careful design; however, Table 10.8 offers a suggested set of values for Minion and Myriad that was used for this book. The effect is shown in Table 10.9.

It is important to realize that T_EX has a fixed memory allocation for fonts, and that every size at which a single font is loaded takes up another slot. Constructing a Multiple Master instance for each commonly used L^AT_EX size (point sizes 5, 6, 7, 8, 9, 10, 11, 12, 14, 17, 20, and 25) takes no more space than scaling all sizes from 10 point. More importantly, however, for each text font size L^AT_EX might load corresponding math fonts⁶; depending on how math is set up (for instance, if we declare a “math typewriter” alphabet or routinely load another

⁵When T_EX fails to load a font, Web2c can be configured to call an external script to create the font, load it, and let T_EX continue.
⁶L^AT_EX does this whenever it encounters a math formula; unfortunately, many macros and environments (such as `tabular`) use math mode internally and thus trigger the font loading.

Table 10.8: Multiple Master instance values matched to *Fontname* and L^AT_EX NFSS codes.

<i>Fontname</i> code	L ^A T _E X weight	Minion value	Myriad value	<i>Fontname</i> name
h	ul	345	215	Hairline
j	el	360	280	Extralight
l	l	380	360	Light
k	sl	400	400	Book
m	m	415	443	Medium
s	sb	500	540	Semibold
b	b	580	640	Bold
x	eb	600	740	ExtraBold
u	ub	620	830	UltraBlack

<i>Fontname</i> code	L ^A T _E X width	Minion value	Myriad value	<i>Fontname</i> name
o	uc	450	300	UltraCondensed
q	ec	468	350	ExtraCondensed
c	c	490	400	Condensed
n	sc	510	450	Narrow
r	m	535	500	Regular
x	sx	550	550	Extended
e	x	565	600	Expanded
v	ex	584	650	Extra Expanded
w	ux	600	700	Wide

	UltraCondensed	ExtraCondensed	Condensed	Narrow	Regular	Extended	Expanded	Wide
Extra Light	Majn	Majn	Majn	Majn	Majn	Majn	Majn	Majn
Light	Majn	Majn	Majn	Majn	Majn	Majn	Majn	Majn
Book	Majn	Majn	Majn	Majn	Majn	Majn	Majn	Majn
Medium	Majn	Majn	Majn	Majn	Majn	Majn	Majn	Majn
Semibold	Majn	Majn	Majn	Majn	Majn	Majn	Majn	Majn
Bold	Majn	Majn	Majn	Majn	Majn	Majn	Majn	Majn
ExtraBold	Majn	Majn	Majn	Majn	Majn	Majn	Majn	Majn
UltraBlack	Majn	Majn	Majn	Majn	Majn	Majn	Majn	Majn

Table 10.9: Minion Multiple Master instances for *Fontname* combinations.

family for AMS symbols), this can mean that switching to a new text size might force the loading of up to 37 new fonts. In addition, fonts do not occupy a fixed amount of memory; the number of ligatures and kerns can vastly increase the space needed. Since we built our fonts with the T1 256 character encoding using fontinst, we had very large ligature tables. TeX's limits can easily be exceeded in this way, and we had to take extreme care in the book (which obviously loads many unusual fonts) to be able to process it all in a single TeX run. Eventually we had to build a variant copy of TeX itself with more memory allocated to fonts than normal (TeX's memory pool is fixed in size when it is built in most implementations, and can be increased only to a certain limit).

CHAPTER 11

PostScript drivers and tools

Much of this book deals with using PostScript in some way, whether in setting fonts, drawing pictures, or creating color. In this chapter we look at the higher-level relationship and are concerned with how to generate PostScript from dvi files, how to manipulate the resulting PostScript file, and how to view or transform it. Among the enormous number of packages for generating, manipulating, and viewing PostScript, we concentrate primarily on public-domain tools that are likely to be available to most L^AT_EX users; specifically, we give detailed descriptions of the dvips driver, the psutils manipulation tools, the Ghostscript viewer, and tools that make TeX pk font bitmaps from PostScript fonts.

To show how some of the tools described can be used together, we end the chapter with two case studies that provide solutions to common problems: converting fragments of L^AT_EX output to bitmap form for use on the World Wide Web, and annotating existing PostScript pictures using L^AT_EX.

11.1 Introduction to dvi drivers

We have already stressed that you should normally, if possible, use the standard graphics and color packages, and to some extent it is thus irrelevant which driver you use. Table 2.1 on p. 29, which lists the facilities supported by different drivers, is a good starting point for readers considering which system or driver to use.

Almost all implementations of TeX include a dvi-to-PostScript driver; some of the notable commercial and public-domain programs are reviewed below.

- The driver in Textures for the Macintosh supports virtual fonts, direct PostScript inclusion and all the functionality needed by the L^AT_EX graphics package. Textures takes

special care to provide facilities for high-end prepress work, with careful attention to the details of the Adobe *Document Structuring Conventions* (see Appendix G in Adobe Systems (1990)) and the needs of color work.

- The drivers in Y&Y T_EX (a matched pair of dvipsone and the Windows previewer dviwindo previewer) are unique in that they rely totally on Adobe Type Manager (ATM) for display, and support *only* PostScript fonts (i.e., not the pk files generated by METAFONT—although a utility is provided to convert these to PostScript Type 3 fonts). Like the Textures driver, these drivers are carefully maintained for maximum compatibility with all PostScript devices and integration with other commercial tools, including ATM for managing, for example, Multiple Master font instances. The drivers are strong on font re-encoding facilities but do *not* support virtual fonts. dvipsone is unique in the commercial world in providing partial downloading of PostScript Type 1 fonts, making a font subset containing just the characters used by a job; this capability is now present in the public-domain dvips, but dvipsone remains its best implementation.
- The drivers in Michael Vulis' T_EX variant, VTeX (Vulis, 1996), are interesting primarily because of their support for VTeX's features of font manipulation inside T_EX.
- Andrew Trevorrow's shareware Macintosh T_EX package, OzTeX, includes a powerful PostScript driver that sorts the text on a page by font; this lets it load a font once, set all the text that uses it, and then start the next font, without continuously swapping between fonts. It has the disadvantage that commands to (for instance) turn color on and off by inserting raw PostScript code into the output stream are almost impossible to make work. The OzTeX distribution also includes an implementation of dvips.
- Unix dvi-to-PostScript drivers go back to the very beginning of PostScript, and the original dvi2ps (by many hands) is still sometimes seen, as is James Clark's dvi2ops, which was revolutionary in the late 1980s for its portability, re-encoding features, and handling of file inclusion. Unfortunately, it was never upgraded to include, e.g., virtual font support.
- The most widely used dvi driver package at this time is undoubtedly Tom Rokicki's dvips. It is available for almost all computer platforms (including DOS, Macintosh, Unix, VMS, and VM/CMS), has been the first-choice test vehicle for many projects (Donald Knuth himself added the first virtual font support), and is regarded as the standard by which other drivers are measured.

It is impossible to do justice here to all the available drivers, and we decided to concentrate on just one (dvips) and hope that the discussion of its features would enable readers to look at their driver's facilities in a useful way. There are two vital PostScript-related considerations for a driver:

- The *portability* of the output: does it conform to standards for Encapsulated PostScript? If so, you can take advantage of a huge quantity of software for post-processing and including the generated files in other applications.

- The *inclusion of raw PostScript* in the output: this “back door” is the key to many useful facilities, such as the whole of the PSTricks package we have described in Chapter 4 and the insertion of pdfmark commands which can be interpreted by Acrobat Distiller (see Haralambous and Rahtz, 1995).

11.2 The dvips PostScript driver

The dvips program has been developed over many years and is hence a mature product, with many important and useful features. It generates standard PostScript that conforms to the Adobe Document Structuring Conventions in nearly all respects.¹ It takes special care that the PostScript generated uses as little printer memory as possible, so that complex documents with many fonts can be printed on devices even as old as the original Apple LaserWriter. (Such complex documents are automatically split into pieces, and printer memory reclaimed between each piece.)

The \special support in dvips is extensive; it is the only driver to support all the conventions of METAPost, emT_EX, and tpic, and sets the current standard for color support (although there is still room for improvement here—see Rokicki (1994)).

Two particular areas of general interest in drivers were pioneered by dvips:

- Automatic generation of missing fonts: if all else fails, dvips temporarily halts and calls another program to build what is needed. This facility is configurable and is not limited simply to running METAFONT.
- Named configuration files: almost all settings can be changed in configuration files that can be specified on the command line. This makes it easy to control several devices with different characteristics.

The automatic generation of fonts depends quite heavily on the operating environment. If used in conjunction with the emT_EX system under DOS, for instance, the emT_EX program dvidrv can be used; on Unix systems a script called MakeTeXPK is normally called, which may differ quite considerably in different installations.

11.2.1 Command line and configuration file options

The output from dvips can be controlled in two ways: by command line switches for a particular job and by commands in one or more configuration files. Using configuration files, you can set parameters globally for the whole system, on a per-printer basis, and on a per-user basis.

When dvips starts up, a global config.ps file is searched for (this must exist; the search path for configuration files depends on how the program was compiled). After this master file is loaded, Unix versions of dvips try to load a configuration file called .dvipsrc

¹At this time, dvips does not correctly identify the font resources it needs or includes, nor does it yet include as many useful structured comments as Textures.

from the user's home directory, and other systems look for a file called `dvips.ini`. Next the command line is read. If the `-P` option is used, the corresponding configuration file is read at this point. Each configuration file (there can be multiple `-P` options) can override anything in the global or user configuration file, and it can also override anything seen in the command line up to the point that the current `-P` option is read.

After the command line has been completely scanned, if no `-P` option was selected, and the `-o` and `-f` options were not set, a `PRINTER` environment variable is searched for. If this variable exists and a configuration file for the corresponding printer exists, this configuration file is loaded last of all. Since the configuration path usually includes the current directory and can be set to include any of the user's directories, users can provide local versions of printer-specific configuration files to override any global system copies.

The number of configuration options and command-line switches is rather large, and nearly all functions can be accessed in either way. In the following catalogue we classify the functionality available and show the command line switch and/or the configuration file command letter. Most options with no parameter can be turned off by immediately suffixing the option with a zero; for instance, to turn off page reversal if it is on by default, use `-r0` on the command line or `r0` in a configuration file. The options that can be turned off in this way are `-a`, `-f`, `-j`, `-k`, `-i`, `-m`, `-q`, `-r`, `-s`, `-E`, `-F`, `-K`, `-M`, `-N`, `-U`, and `-Z`.

Within a configuration file, any empty line or line starting with a space, asterisk, equal sign, or a pound sign is ignored. All other lines must take the form of a command letter, followed by a space, and then some parameter.

Each entry below shows the command-line option, the configuration file option, and the type of argument (if any) that is expected. Note that this set of options is that available for Versions 5.60 and later of `dvips`; you should make sure you have the latest version.

Page selection

`-p` *number* Start printing at the page numbered *number*. The default is the first page in the document. If *number* is prefixed by `=`, then it is treated as the absolute sequence number rather than the `LaTeX` page number (technically, the value of `\count0`). Thus, using `-p =9` starts with the ninth page of the document, regardless of how the pages are actually numbered on the printout.

`-l` *number* Stop printing after the page numbered *number*. The default is the last page in the document. A prefix of `=` means that the number is relative to the number of pages in the file rather than the `LaTeX` page numbers.

`-pp` *numbers* Print the range of pages specified by a comma-separated list of *numbers*. This list can consist of individual page numbers or page ranges of two numbers separated by a colon. Thus you can print pages 3 to 10, 21, and 73 to 92 by specifying the option `-pp 3:10,21,73:92`; note that you cannot use this to reverse the sequence of pages—if you say `-pp 3,2,1`, pages 3, 2, and 1 are printed in their natural order.

`-n` *number* Print *number* pages; the default is 100000. This is sometimes more convenient than using `-l`; e.g., to get 20 pages starting from page 53, say `-p 53 -n 20`.

`-A` Print only the odd-numbered pages.

`-B` Print only the even-numbered pages.

`-r` *r* Produce pages in reverse order; by default, page 1 is printed first. This option is useful on printers like HP Deskjets that stack paper face up, so that the page printed first is at the bottom of the stack.

Output

`-o` *name* Send the PostScript output to a file called *name*; if no file name is given, then for a `dvi` file called *file.dvi*, the output file is called *file.ps*. If the first character of *name* is an exclamation mark, then the remainder is used as a command to pipe the output; e.g., if you specify `-o !lpr`, the output is fed to the standard Unix print command. (This may not be enabled under some operating systems, such as DOS or Macintosh.) The option also disables the use of the `PRINTER` environment variable and turns off the automatic sending of control-D if it was turned on with the `-F` option or in the configuration file; use `-F` after this option if you want both.

`-f` *f* Write the PostScript to standard output. Note, incidentally, that `dvips` requires the `dvi` file to be "seekable", so the input cannot come from a pipe.

`-m` Specify manual feed for the printer.

`-t` *papertype* Set the paper type to *papertype*. The value of *papertype* should be defined in one of the configuration files, along with the appropriate code to select it. You can also specify `-t landscape`, which rotates a document by 90 degrees. To rotate a document whose size is not letter, you can use the `-t` option twice, once for the page size and once for landscape. The upper left corner of each page in the `dvi` file is placed one inch from the left and one inch from the top. Use of this option is highly dependent on the configuration file, since this is where paper types are defined.

@ Define a paper size in a configuration; described in detail in Section 11.2.2.

`-O` *offset* Move the origin of the page. The *offset* is a comma-separated pair of *x*, *y* values such as `.1in, -.3cm`; the *x* value moves the page to the right and the *y* value moves it downwards. The default origin is one inch down and one inch to the right from the upper left corner of the paper.

`-T` *offset* Set the paper size to a pair of dimensions. This option takes its arguments in the same style as `-O`. It overrides any paper size `\special` in the `dvi` file.

`-x` *number* Set the magnification ratio to *number*/1000 and override the magnification specified in the `dvi` file. The *number* must be between 10 and 100000. It is recommended that you use standard magstep values (1095, 1200, 1440, 1728, 2074, 2488, 2986, and so on) to limit the total number of `pk` files generated.

`-D` *D* *number* Set the horizontal and vertical resolution in dpi (dots per inch) to *number*. This affects the choice of bitmap fonts that are loaded and also the positioning of letters in resident PostScript fonts. The *number* must be between 10 and 10000.

`-X` *X* *number* Set the horizontal resolution only in dots per inch to *number*.

`-Y` *Y* *number* Set the vertical resolution only in dots per inch to *number*.

Output configuration

-E ☐ Attempt to generate Encapsulated PostScript with the correct bounding box.

This works only on one-page files (or one-page selections from longer files), and it considers only the positions of characters and rules, ignoring any included graphics (see p. 457 for discussion of how to deal with this). In addition, it gets the glyph metrics from the `tfm` file, so characters lying outside their enclosing `tfm` box may confuse it; the bounding box may also be a bit too loose if the character glyph has significant left or right side bearings. Note that `dvips` output may be resolution-dependent if `pk` fonts are used and this does not make very good EPS files, especially if the images are to be scaled; use these EPS files with great care.

-e ☐ *e* *number* Place each character so that it is no more than *number* pixels from its “true” resolution-independent position on the page. The default value of this parameter is resolution-dependent (it is the number of entries in the list 100, 200, 300, 400, 500, 600, 800, 1000, 1200, 1600, 2000, 2400, 2800, 3200, ... that are less than or equal to the resolution in dots per inch). Allowing individual characters to “drift” from their correctly rounded positions by a few pixels and then regain the true position at the beginning of each new word improves the spacing of letters within words. However, this facility can be dangerous in some tabular work (e.g., with the `hhline` package) with complicated rules that may not be aligned properly. If you encounter problems, set the value to 0.

-h ☐ *h* *name* Insert file *name* as an additional PostScript header. If the name is simply “-”, this option suppresses the addition of all header files to the output. The contents of the header file is inserted into the PostScript `userdict` dictionary.

-k ☐ Print crop marks. This option increases the paper size (which should be specified, either with a paper size `\special` or with the `-T` option) by half an inch in each dimension. Each page is then moved by a quarter of an inch and crop marks are drawn in each corner.

Font handling

-mode ☐ *M* *mode* Specify the METAFONT *mode* for any `pk` fonts that may have to be created. If this contradicts a specified resolution, it is ignored.

-M ☐ Turn off automatic font generation. If any fonts are missing, commands to generate the fonts are appended to the file `missfont.log` in the current directory.

-V ☐ Download nonresident PostScript fonts as bitmaps. This requires use of programs like `ps2pk` or `gsftopk` to generate the required bitmap fonts (see Section 11.5).

-Z ☐ *Z* Compress bitmapped fonts before they are downloaded, thereby reducing the size of the PostScript file. This is very useful at high resolutions or when very large fonts are used, but it slows down printing, especially on older printers.

-j ☐ *j* Partially download PostScript Type 1 fonts. Font subsets are created containing only those characters used in the document. (Note that this feature appeared first in Version 5.60 of `dvips`.)

R *numbers* Set up a list of default resolutions (a list of *numbers* separated by spaces) to search for `pk` fonts, if the size requested is not available. The output then uses PostScript scaling to scale the font to the requested size. The resultant output is ugly, and a warning is issued. To turn this feature off, use a line containing `R` and no numbers.

The configuration file and the printer

-P ☐ *configname* Set up the output for a particular *configuration* or printer. This is implemented by reading in `config.configname` (`configname.cfg` in DOS systems), which can set most of the options. It is recommended that all standard defaults go in one master `config.ps` file and that only things that vary from printer to printer go in the configuration files, as `config.ps` is read before `config.configname`. If no `-P` command is given, the environment variable `PRINTER` is checked. If that variable exists and a corresponding configuration file is found, that latter file is read. Alternatively (under Unix only), specifying a `-P` option with no corresponding configuration file is interpreted as a request to send the output to a system printer called *configname*. Thus `-Plp1` would direct output to the system `lp1` printer queue if `config.lp1` does not exist.

m *number* Set *number* (default 180000) as the size of *printer memory* that can be used for fonts. This is intended for memory conservation and automatic splitting into sections. You can determine the available memory by sending a file containing the following to your printer:

```
%!
/Times-Roman findfont 30 scalefont setfont 144 432 moveto
vmstatus exch sub 40 string cvs show pop showpage
```

It is usually a good idea to tell `dvips` that the printer has rather *less* memory than the number returned by this procedure, as other things may fill up memory. For typical `TeX` jobs, you should worry if the reported memory size is less than about 300000. If you have “unlimited” virtual memory, just use a very high number like one million.

I ☐ Ignore the `PRINTER` environment variable.

p *name* Set the name of the file (default `psfonts.map`) which lists available PostScript fonts. This option allows us to distinguish between different sets of resident fonts in printers. If the name starts with a “+” character, then the rest of the name is used as an *additional* map file, rather than one overriding the current set.

Memory considerations

a *a* ☐ Conserve memory by making three passes over the `dvi` file instead of two in order to load only those characters actually used.

S *i* ☐ *number* Set the *maximum number of pages* in each “section”. This option is most commonly used with `-i`; when used in a configuration file, the `i` option is equivalent to `-i -S`.

-i ☐ ☐ Output each section into a separate file. Under certain circumstances, dvips splits the document up into "sections" to be processed independently. With this option, dvips places each section into a separate file; the new file names are constructed by replacing the suffix of the supplied output file name by a three-digit sequence number. This option is most often used in conjunction with the **-S** option, which sets the maximum section length in pages. If, for instance, your device cannot handle more than 10 pages at a time in one job, you can use **-i -S 10** to split it automatically into ten-page sections that are written into separate files.

Miscellaneous

-b ☐ ☐ *number* Generate *number* copies of each page by duplicating the page body. This can be used, for example, in conjunction with a header file setting **bop-hook** to do color separations (see Section 9.5 on p. 346).

-c ☐ ☐ *number* Generate *number* copies of every page by using PostScript's **#copies** feature. Default is 1. (For collated copies, see the **-C** option below.)

-C ☐ ☐ *number* Create *number* of *collated* copies (by replicating the data in the PostScript file). This takes longer to print than with the **-c** option, but it is much more convenient and is easier than submitting the same PostScript file over and over again.

-d ☐ ☐ *number* Set *debugging* options. This is intended only for emergencies or detailed examination by experts; it works only if dvips has been compiled with the **DEBUG** option. The useful values of *number* are discussed below in Section 11.2.6; use a value of -1 for maximum output.

-q ☐ ☐ Run in *quiet mode*. Normally, dvips shows page numbers and names of files it includes; this option tells it to print nothing but error messages to the normal error output.

-R ☐ ☐ Run *securely*. This disables command execution in **\special** (via **'**) and configuration files (via the **"E"** option), pipes as output files, and opening of any absolute filenames.

☐ ☐ **E** *command* Execute the system *command* immediately, while the configuration file is being read. In many installations this is disabled so as to simply produce an error message, as it is clearly quite insecure.

☐ ☐ **W** *text* If *text* is given, it is written to the standard (error) output. Just the option by itself cancels any previous message. This is useful in the default configuration file if you want to require the user to specify a printer, for instance, or if you want to notify the user that the resultant output has special characteristics.

Paths

☐ ☐ **H** *path* Search *path* for PostScript header files (including PostScript Type 1 fonts).

☐ ☐ **P** *path* Search *path* for bitmap **pk** font files. If **%** characters are found in *path*, sub-

stitutions as described below are made, depending on the following letter, and a search is then undertaken for the resulting filename.

%f	font name
%b	output device horizontal resolution (dots per inch)
%d	font size (dots per inch)
%p	font type (this is always pk)
%m	font mode (given by the M option)

If a path *does* contain a **%** character, the full filename must be given, including the path, rather than just the directory name; a path element such as **/fonts/%b** tries to open **/fonts/300** when looking for **cmr10.329pk**, for instance, and this may not be what is intended; **/fonts/%b/%f.%dpk** is needed.

☐ ☐ **S** *path* Search *path* for illustrations (such as Encapsulated PostScript files).

☐ ☐ **T** *path* Search *path* for **tfm** files.

☐ ☐ **V** *path* Search *path* for virtual font **vf** files.

Compatibility issues

-N ☐ ☐ **N** Turn off *structured comments*; this might be necessary on some systems that try to interpret PostScript comments in the wrong way, or on some old PostScript printers or software.

-F ☐ ☐ Add Control-D (ASCII code 4) as the very last character of the PostScript file. You may need this when dvips is being used in an operating system like DOS and is talking to the printer directly instead of via spooling software. The Control-D indicates to some PostScript interpreters that a job is finished.

-s ☐ ☐ **s** Enclose the entire output in a PostScript *save* and *restore* pair. Try to avoid using this, since the result is not proper EPS, but it may be necessary if you are driving the printer directly and do not care about the portability of the output.

-K ☐ ☐ **K** Remove comments in included PostScript graphics, font files, and headers. This is sometimes necessary to get around bugs in spoolers or PostScript post-processing programs. Specifically, **%%Page** comments inside EPS files often cause difficulties. However, use of this option can cause some included graphics to fail, since the PostScript header macros from some software packages read portions of the input stream line by line, searching for a particular comment. The option is turned *on* by default because some PostScript previewers and spoolers have problems with the structuring conventions.

-U ☐ ☐ **U** Disable a PostScript virtual memory saving optimization that stores the character metric information in the same string used to store the bitmap information. This is only necessary when using a Xerox 4045 PostScript interpreter, which has an odd bug.

All of the path-searching configuration can be overridden at run-time by setting environment variables which we do not list here.

11.2.2 Paper sizes

Documents are normally designed for a particular paper size, whether in portrait or landscape orientation. Information on page size should ideally be placed in the dvi file, not chosen at print time, and dvips therefore supports a `papersize \special` command, although it can also be specified in configuration files and on the command line. The format of the `\special` command (which must occur somewhere on the first page of the document²) is

```
\special{papersize=dimension, dimension}
```

The first dimension is the horizontal size of the paper and the second is the vertical size; they can be expressed using any of the normal TeX dimensions of *in* (inches), *cm* (centimeters), *mm* (millimeters), *pt* (points), *sp* (scaled points), *bp* (big points, the same as the default PostScript unit), *pc* (picas), *dd* (didot points), and *cc* (ciceros). Thus American letter-size paper would be chosen by putting:

```
\special{papersize=8.5in,11in}
```

in the L^AT_EX document (or, better, in the document class), and A4 paper to be used in landscape would be chosen with:

```
\special{papersize=297mm,210mm}
```

Of course, using such a command merely informs dvips of the desired paper size; you must set the `\textwidth` and `\textheight` dimensions (and the appropriate margins) in your L^AT_EX document class to actually make use of the page size.

When dvips processes a file, it matches up the requested paper size with a setup defined in the configuration file, from which it works out what to put in the PostScript output. Paper sizes are defined in configuration files by lines starting with `@`.

There are three formats for `@` lines. First, an option specified on a line by itself, with no parameters, instructs dvips to discard all other paper size information (possibly from another configuration file) and start from scratch. Second, if the option contains three parameters, a keyword followed by two dimensions, it is interpreted as starting a new paper size description, where the keyword is the name and the dimensions are the horizontal and vertical size of the sheet of paper. Third, if both dimensions are zero (note that you must still supply units, like `0mm`), then any page size in the dvi file is a match. If the `@` character is immediately followed by a `+` character, then the remainder of the line (after skipping any leading blanks) is treated as PostScript code to send to the printer to select the paper size being defined. Within those `+` lines, if the first character is an exclamation mark, then the line is put in the initial *comments* section of the final output file; otherwise, it is put in the *setup*

²L^AT_EX has a convenient way of doing this with the `\AtBeginDvi` macro.

section of the output file—thus the definition of A4 paper in the standard configuration file looks like this:

```
@ A4 210mm 297mm
@+ ! %%DocumentPaperSizes: A4
@+ %%BeginPaperSize: A4
@+ a4
@+ %%EndPaperSize
```

This indicates that the structured comment `%%DocumentPaperSizes: A4` is to be placed in the initial document description area, and the `PaperSize` lines go in the setup section. In this example, the single PostScript command `a4` (defined in the dvips header files) is what actually does the work of instructing the PostScript interpreter, but the code may be arbitrarily complicated. If you know that all your printers understand Level 2 PostScript, you can use a command like:

```
@+ << /PageSize [ 595 842 ] >> setpagedevice
```

When dvips sees a paper format on the command line, it looks for a match by *name*; when it sees a `papersize \special` command, it looks for a match by dimensions (they must match within a quarter of an inch). The first match found (in the order in which the paper size information is found in the configuration file) is used. If nothing matches, a warning is printed and the first paper size given is used, so the first paper size should always be the default.

Landscape mode for all paper sizes is supported automatically; after trying to match a pair of dimensions, dvips then tries to match them in reverse. If that succeeds, it inserts PostScript code in the output to rotate the pages. Note that this is not the same as including instructions asking the printer to switch to a different paper tray.

If our L^AT_EX file contains

```
\special{papersize=297mm,210mm}
```

dvips finds that a paper size of 210mm × 297mm in the configuration file makes a match, but adds landscape mode. By default it behaves as if the paper were rotated ninety degrees counterclockwise; if your landscape orientation should be rotated clockwise, the document class or document should include the following code:

```
\special{! /landplus90 true store}
```

(`\special` commands like this are discussed in Section 11.2.3 on the following page). However, if the printer directly *supports* landscape A4 paper, then the configuration file should have a special entry for 297mm × 210mm and supply the correct PostScript code to select the device.

If your printer has a command to set special arbitrary paper sizes, then specify dimensions of size zero (like 0in 0in); the PostScript code that sets the paper size can refer to the user-requested dimensions as `hsize` and `vsize` which are macros defined in PostScript that return the requested size in default PostScript units. For example, in sending output to a Linotronic typesetter, which has a continuous roll of paper, you can define a paper size name to make the text run across the width of the film as follows:

```
@ film1 0in 0in
@+ statusdict /setpageparams known
@+ { hsize vsize 0 1 statusdict begin
@+ { setpageparams } stopped end }
@+ { true }
@+ ifelse
@+ { statusdict /setpage known
@+ { hsize vsize 1 statusdict begin
@+ { setpage } stopped pop end } if
@+ }
@+if
```

This could then be used with a command line option of `-t film1 -T 297mm,420mm` to set A3 pages with the long side along the edge of the film. `dvips` is distributed with a configuration file `config.lino` that defines these setups.

Note that the PostScript commands used to define paper sizes may well be device dependent and thus make the output less portable.

11.2.3 \special support

The wide range of `\special` commands supported by `dvips` fall into four groups:

1. specifying resources, general configuration and including PostScript files;
2. inserting literal PostScript code;
3. defining and using color;
4. supporting the conventions of `emTeX`, `tpic`, and `METAPOST`.

We recommend that you do *not* use these `\special` commands directly in `LaTeX` documents, but instead use higher-level macro packages that can work with more than one driver. Specifically, we strongly recommend that all graphics inclusion, rotation, scaling, and color typesetting be done using the standard `LaTeX` graphics and color packages. Our description here is intended for writers of new macro packages and those simply curious about how the packages work under the hood.

Including PostScript files and literal PostScript code

The `\specials` for resource and graphics inclusions are as follows:

Name	Example	Comments
<code>papersize</code>	<code>\special{papersize=5in,8in}</code>	set paper size
<code>landscape</code>	<code>\special{landscape}</code>	special form of <code>papersize</code> to select landscape orientation of default paper size
<code>header</code>	<code>\special{header=duplex.pro}</code>	insert PostScript header file into output
<code>PSfile</code>	<code>\special{PSfile="pot.ps"}</code>	insert graphics file
<code>psfile</code>	<code>\special{psfile=pot.ps}</code>	insert graphics file (alternative syntax)

To include some graphics files in `dvips` output or to support sophisticated packages using literal PostScript in `LaTeX` (such as `PSTricks`), it is sometimes necessary to put a PostScript header file in the output; `dvips` provides support for this with the `header \special` command. The dictionary stack is at the `userdict` level when these header files are included. It is important to consider how much printer memory they might consume; header files may contain, in the first 1024 bytes, a line of the form:

`%VMusage: minimum maximum`

The maximum value is used to calculate how much memory remains for `dvips`' own purposes. If there is no `%VMusage` line, then the total size in bytes of the header file is used to approximate the memory requirements.

The `\special{PSfile=...}` command is followed by a set of *keyword=value* pairs (each pair separated by spaces) to specify the picture size and other parameters. The possible keys are as follows (the dimension parameters are all expressed in PostScript points which are equivalent to `TeX`'s bp unit):

Key	Default	Meaning
<code>hoffset</code>	0	horizontal offset
<code>voffset</code>	0	vertical offset
<code>hsize</code>	612	horizontal clipping size
<code>vsize</code>	792	vertical clipping size
<code>hscale</code>	100	horizontal scaling factor
<code>vscale</code>	100	vertical scaling factor
<code>angle</code>	0	rotation
<code>clip</code>		force clipping to the bounding box
<code>llx</code>		<i>x</i> coordinate of lower left corner
<code>lly</code>		<i>y</i> coordinate of lower left corner
<code>urx</code>		<i>x</i> coordinate of upper right corner
<code>ury</code>		<i>y</i> coordinate of upper right corner
<code>rwi</code>		final width of picture
<code>rhi</code>		final height of picture

It is important to understand that dvips does *not* read the graphics file to be included to find the bounding box; this is done by the $\LaTeX$ file and the appropriate values passed through by the keys in the `\special` command. The dvips driver for the standard $\LaTeX$ graphics package uses the `llx`, `lly`, `urx`, and `ury` keys to set the picture bounding box, the `clip` key if needed, and the `rwi` and `rhi` keys if the picture is to be scaled.

There are two ways to insert PostScript code directly into the output; in the first, “safe” method, the code is surrounded in the output by PostScript commands to limit its effect; the second method is unprotected and can affect the current drawing state. The latter is needed for operations like text rotation, when we really *do* want to change the current state, while the former might be used to draw a page border independently of the main text.

The “safe” `\special` commands are:

Name	Example	Comments
"	<code>\special{"newpath 0 0 moveto} 100 100 lineto stroke</code>	insert literal PostScript
!	<code>\special{! /magscale false def}</code>	insert literal PostScript into user dictionary

You can define your own macros for use in literal graphics code; they are defined just like literal graphics, except that the `\special` command begins with an exclamation mark instead of a double quote. These literal macros are included as part of the header material in a special dictionary, `SDict`, which is the first one on the PostScript dictionary stack when any literal PostScript is used.

The unprotected literal PostScript insertion `\specials` are:

Name	Example	Comments
ps:	<code>\special{ps: .5 setgray}</code>	insert literal PostScript, with no protection
ps::	<code>\special{ps:: 0.5 setgray}</code>	insert literal PostScript
ps::[begin]	<code>\special{ps::[begin]gsave .5 setgray}</code>	start inserting literal PostScript
ps::[end]	<code>\special{ps::[end]grestore}</code>	end inserting literal PostScript
ps: plotfile	<code>\special{ps: plotfile foo.ps}</code>	insert file into output with no protection

The `::` commands can be used to construct an unbroken set of PostScript commands in the output with a series of `ps::` commands, starting with `ps::[begin]` and ending with `ps::[end]`. This may be useful when you have a lot of PostScript to write. In general, however, use the simple `ps:` convention wherever possible.

You should realize when writing unprotected PostScript that dvips works with its own coordinate system, which is dependent on the resolution. This is stored in the PostScript macro called `Resolution`.

Color

dvips recognizes two `\special` commands for specifying color. The first `\special` sets the color of the page:

```
\special{background colorname}
```

If several background colors are set on a page, the last one is that which is used, and it affects the whole page; you cannot set a background for part of a page (use `\colorbox` instead).

To set the current color for text, there are three `\special` commands, which are handled correctly even when the pages are repeated or reversed.

```
\special{color colorname}
```

This sets the current color to *colorname*.

```
\special{color push colorname}
```

This saves whatever is the current color on a special stack and sets the new color to *colorname*.

```
\special{color pop}
```

This takes the top color off the stack and sets the current color to that value.

Color names in these `\special` commands may be one of three things:

1. The name of a PostScript procedure defined in a header file; the `color.pro` file supplied with dvips defines 64 color names (shown in Color Plate XIII) as procedures.
2. A color model name (*rgb*, *hsb*, *cmyk*, or *gray*) followed by an appropriate number of parameters; e.g., `color cmyk 0.1 0.2 0.4 0.9`.
3. A double quote followed by any arbitrary PostScript code; this could be used, for instance, to access PostScript patterns (an addition in PostScript Level 2), or to set color in color models other than those supported directly by dvips.

Although we can use this mechanism to access PostScript patterns or screens, only one color can be active at a time; this means that we cannot switch colors and patterns independently.

Other \special support

Support for $\epsilon\TeX$ and `tpic` is discussed in Appendix A.1 on p. 463 ($\epsilon\TeX$) and A.2 on p. 464 (`tpic`), and for `METAPOST` in Chapter 3.

11.2.4 Font support

TeX traditionally supplies bitmap (pk) fonts to printers; when the device is a PostScript one, however, we also want access to built-in PostScript fonts as well as those we can download. In dvips, the built-in font list is a file called (by default) `psfonts.map`, which consists of lines that map a TeX font name onto a full PostScript font name. Thus the simplest case would be:

```
boldtimes Times-Bold
```

which tells dvips to map the TeX name `boldtimes` onto the full name `Times-Bold`. When dvips finds a font name in the dvi file it is checked against this list. In practice, because we are following the naming scheme described in Section 10.4 on p. 381, our line would probably look like:

```
ptmb8r Times-Bold "TeXBase1Encoding ReEncodeFont" <8r.enc
```

Here we are using two extra features of the mapping file. First, any PostScript code in double quotes following the PostScript name is applied to the font when it is loaded. The string `TeXBase1Encoding ReEncodeFont` re-encodes the font to make all the characters available (`ReEncodeFont` is defined in the header files that dvips downloads). Second, anything following a `<` is interpreted as a file name whose contents are to be placed in the output; in this case `8r.enc` actually defines the `TeXBase1Encoding` just mentioned. This re-encoding step is explained in Section 10.2.3 on p. 367.

dvips supports two other PostScript commands that can be given in the mapping file in the same way as `ReEncodeFont`, namely `ExtendFont` and `SlantFont`. Their use is shown in the following extract from a map file:

```
purro8x CentaurMT-Expert ".167 SlantFont "
hlcrt8rn LucidaTypewriter ".850 ExtendFont TeXBase1Encoding ReEncodeFont " <8r.enc
```

The first line says that the TeX font `purro8x` is obtained by taking the PostScript font `CentaurMT-Expert` and slanting it to the right by `.167`. The second line states that `hlcrt8rn` is the PostScript font `LucidaTypewriter`, squashed horizontally to 85% of its normal width (a value of 1.5 would make it half as wide again as its normal width) and re-encoded.

Finally, the dvips map file lets you request that a PostScript font be downloaded with the current job to the printer and thenceforth treated as if it were built-in. This uses the same `<` syntax as for encoding files:

```
mzdr MZapfDingbatsPS <mzdr.pfb
```

This indicates that the font `mzdr.pfb` should be downloaded to satisfy requests for `MZapfDingbatsPS`.

All the available features are exercised by this line:

```
mbbb8r Bembo-Bold "TeXBase1Encoding ReEncodeFont" <8r.enc <mbbb8a.pfb
```

which maps a slanted and re-encoded version of `Bembo-Bold` (downloaded with the job) to the TeX name `mbbb8r`. Note that font files (and encoding files) are downloaded only once, however often map files refer to them.

Although many users simply place all the mapping lines in the default file called `psfonts.map`, dvips allows for granularity by the use of multiple configuration files and map files. Extra map files can be loaded with the `p` command in a configuration file; thus

```
p new.map
```

establishes `new.map` as the ~~on~~set of mappings, whereas

```
p +new.map
```

appends the values in `new.map` to the current list.

11.2.5 Special hooks

dvips provides a series of "hooks" that let the user plug in code to be executed at several useful places. It does this by defining four PostScript procedures (`start-hook`, `end-hook`, `bop-hook`, and `eop-hook`) in the PostScript userdict; these procedures are executed at the start and the end of the document and at the beginning and the end of every page, respectively. The procedures can be redefined in headers added by the `-h` option or the `\special{header=...}` command. This code is executed outside the save/restore context of the page, so it can accumulate information about the whole document (unless it is broken into sections because of memory constraints.) The default PostScript coordinate system and origin are in effect when the code in these procedures is run.

When `bop-hook` is executed, two values are placed on the PostScript stack, the TeX page number and the sequence number of the page in the file. For `start-hook`, the horizontal size, the vertical size, the magnification, the horizontal and vertical resolutions (in dpi), and the name of the device output file are put on the stack. The procedures must leave these parameters on the stack. The other hooks have no values on the stack.

If you do use `bop-hook` or `eop-hook` to keep information across pages, then the document no longer complies with the Adobe document structuring conventions, so you should use the `-N` option to turn off structured comments.

Typical uses of the `bop-hook` procedure are writing very large words across each page (overlying the text) or printing cropmark identifiers in the corner of each page. For example, if you put the following code into a file called `secret.pro`:

```
userdict begin
/bop-hook{gsave
200 150 translate
65 rotate
```

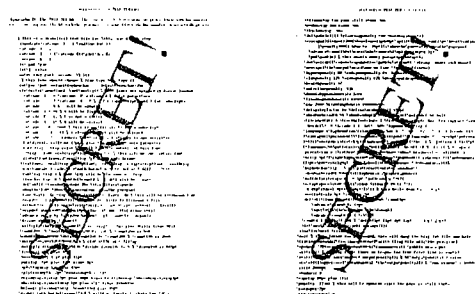



Figure 11.1: Stamping pages using dvips

```

/Times-Roman findfont 144 scalefont setfont
0 0 moveto
0.7 setgray
(SECRET!) show
grestore} def end

```

and run a file called `test.dvi` through `dvips` with the command line

```
dvips -h secret.pro test.dvi -o test.ps
```

each page looks like those in Fig. 11.1.

Jürgen Vollmer's `draftcopy` package uses this technique to offer a more sophisticated $\text{\LaTeX}$ interface to add overlay information on pages; the default text can be in different languages ("Draft" in English, "Entwurf" in German, etc.), or the user can specify some other word. It also supports several dvi drivers other than `dvips`.

An elegant trick suggested by the author of `dvips` is to write a header file dynamically from a configuration file that can give you, for instance, access to the current date. You can put the following code into a file called `config.dte`:

```

E echo /bop-hook userdict begin \{ gsave > date.txt
E echo /Times-Roman findfont 10 scalefont setfont >> date.txt
E echo 10 10 moveto \{printed on 'date'\} show grestore >> date.txt
E echo \} def end >> date.txt
h date.txt

```

and then say

```
dvips -Pdte test.dvi -o test.ps
```

to write, at the bottom of each page, the date on which the document went through `dvips`. This works only on Unix systems (the program must be able to execute commands like `date` from within itself), and does not work if the `E` option has been disabled in the compilation of `dvips`; it does present a security loophole.

As an alternative, you can write material to the PostScript header area using the `\special{!...}` technique; you can therefore recode the last example as:

```

\special{!
userdict begin
/bop-hook {gsave
/Times-Roman findfont 10 scalefont setfont
10 10 moveto (typeset on \today) show grestore
} def end
}

```

This of course writes the date on which the document was typeset, not put through `dvips`.

Another use for this kind of hook is in prepress applications. In using $\text{\LaTeX}$ and `dvips` to prepare film directly for a printer, you might need to print it "face down", as a mirror image. The following code in a header file (personal communication from Henning Heinze) achieves this simply with PostScript `translate` and `scale` operators (note that the `translate` in the example is expressed in terms of the page width; `dvips` makes available a PostScript macro called `hsize` for this purpose):

```

userdict begin /mirror {
hsize 25.4 div 72 mul 0 translate
-1 1 scale } def
/bop-hook{ mirror} def
end

```

The effect is shown in Fig. 11.2(a). A simple change gives a mirror on the horizontal axis (Fig. 11.2(b)). In Level 2 PostScript this mirror effect, and the white-on-black effect also needed in printing, are directly supported with the `setpagedevice` command (see Adobe Systems, 1990, p. 237). This means you need merely insert the following code in the PostScript preamble:

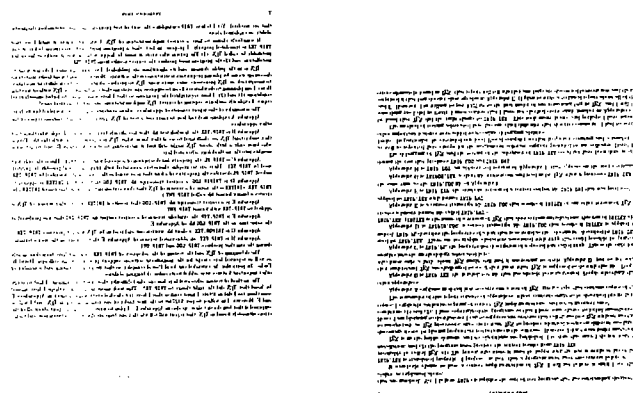
```
<< /MirrorPrint true >> setpagedevice
```

to get mirroring, and

```
<< /NegativePrint true >> setpagedevice
```

to get black on white. Unfortunately, not all PostScript implementations support this (Ghostscript does not, for instance). High-end phototypesetters, for which these functions are important when printing to film, most often do support them.

Section 9.5.2 on p. 348 describes a system for making color separations that also uses the `bop-hook` procedure. Users of the hooks are warned that `dvips` makes no provision for checking the content or accumulating commands. If you want to write `SECRET!` on each



(a) (b)

Figure 11.2: Mirror-image printing with dvips.

page and print crop marks, you cannot just load two header files in succession, but you must instead write more sophisticated PostScript. Bogusław Jackowski's `bop-hax.tex` in the collection of useful tricks called `TeX-PS` in the CTAN archives, `macros/generic` shows how this can be done.

11.2.6 Debugging

The `-d` option for `dvips` is for tracking down errors and understanding what is going on. You must supply an integer specifying the class of information to be displayed. To get several types of information, simply add the numbers together for the types you are interested in. Choose from:

1	specials	4	fonts	16	headers	64	files
2	paths	8	pages	32	font compression	128	memory

For example calling `dvips` with an option of `-d 4` yields something like this:

```
This is dvipsk 5.58f Copyright 1986, 1994 Radical Eye Software
' TeX output 1995.03.25:2323' -> mirror2.ps
Defining font () cmr7 at 17.4pt
Defining font () cmr10 at 10.0pt
Defining font () cmsl10 at 10.0pt
```

```
Defining font () cmr8 at 8.0pt
Defining font () cmbx10 at 10.0pt
Defining font () cmtt10 at 10.0pt
Font cmr10 <cmr10> is resident.
Font cmr8 <cmr8> is resident.
Font cmtt10 <cmtt10> is resident.
<texc.pro><texps.pro>. <cmtt10.pfb><cmr8.pfb><cmr10.pfb>[2]
```

This shows which fonts are called for and how they are located (here PostScript versions of the CM fonts are included in the output file and downloaded with the job).

11.3 PostScript page-manipulation tools

It is very often useful to be able to manipulate whole pages of PostScript as they come out of programs like `dvips`; one of the reasons for the Adobe File Structuring Conventions is to provide enough information to make this possible. You may wish to reorder them, scale them, merge files together, and so on. In the next section, we look at a useful suite of programs that perform most of the tasks you need. Before that, however, let us briefly consider some of the other ways of making several logical pages print on one sheet of paper.

Many people want no more than to print '2 up', i.e., two normal pages shrunk to fit on a single sheet of paper (in order to save paper, or to make a booklet.) In the $\text{\TeX}$ world, this can be done entirely in $\text{\TeX}$ itself by clever redefinition of the `\shipout` primitive, and magnification; Timothy van Zandt's `2up` package implements this strategy, and is used in the seminar package discussed in Section 9.4 on p. 338 to print two slides on a single page. Its big advantage is that it requires no special `dvi` driver and scales the fonts correctly (it requires the user to generate a new set of bitmap fonts for the shrunken pages or to use scaleable fonts). You can specify the width and height of the source and target pages, and change other options; the only driver requirement in normal use is that landscape printing can be specified for the normal case of two pages shrunk and placed side by side.

Another simple way to print multiple pages on one sheet of paper, if you use a PostScript `dvi` driver, is to include a PostScript header file redefining the PostScript low-level `showpage` operator. (It is of course vital that you use scaleable PostScript fonts and not bitmap `pk` fonts, or PostScript's scaling will break up the characters badly.) One such header file is Ross Cartlidge's `multi.pro`; if you load this as a header file you need to put some starting PostScript code in your preamble and some closing code at the end of the pages. The `dvips` `start-hook` and `end-hook` procedures described in the previous section can be used to do this.

The closing PostScript code is always `endmulti`; the starting code is the command `multi` preceded by anywhere from 3 to 7 parameters. The compulsory parameters are:

`landscape` divide the page using landscape or portrait orientation; possible values are `true` (landscape) and `false` (portrait);

`rows` the number of rows of pages;

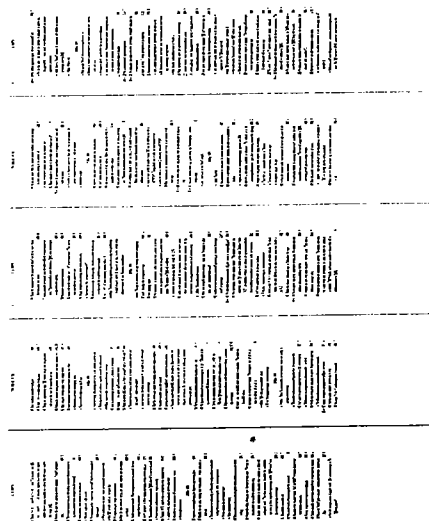


Figure 11.3: Multiple logical pages on one physical sheet, using `multi.pro`.

`columns` the number of columns of pages.

The other parameters are:

`left-to-right` whether the physical page fills with logical pages left to right (`true`) or right to left (`false`); the default is `true`;

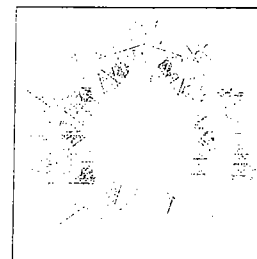
`top-to-bottom` whether the physical page fills with logical pages top to bottom (`true`) or bottom to top (`false`); the default is `true`;

`row-first` whether to fill rows first (`true`) or columns first (`false`);

`dividers` whether to print rules between logical pages; the default is `true`.

Thus a starting code of `true 1 5 multi` places logical pages on the physical page in five columns and one row, in landscape orientation; pages are forced to fit the requested layout, so in this case they are stretched sideways to fit. The result is shown in Fig. 11.3. A more useful layout would be specified by `false 4 4 multi`, four rows and four columns on a portrait page; printing 16 pages on one physical sheet page is surprisingly useful if you are just doing something like checking layout of a long book.

For really arbitrary arrangements of a set of logical pages on a physical page, you can resort to a technique we used in this book. To show an effect, we used `dvips` to create an EPS file of the result as a single-page $\text{\LaTeX}$ job and included it back in our text as a graphic. This allows ridiculous effects like printing tiny versions of the first nine pages of the $\text{\LaTeX}$ “Frequently Asked Questions” around a circle like this (using `PSTricks`):



11-3-1

```
\usepackage {pstricks,graphicx}
\newcommand{\Page}[2]{% angle, filename
  \put[#1]{#1}(0,0){%
    \includegraphics[width=.5in]{#2}}}%
\fbbox{%
  \begin{pspicture}(-2,-2)(2,2)
    \psset{labelsep=.8cm}
    \Page{360}{p1} \Page{36}{p2}
    \Page{72}{p3} \Page{108}{p4}
    \Page{144}{p5} \Page{180}{p6}
    \Page{216}{p7} \Page{252}{p8}
    \Page{324}{p9}
  \end{pspicture}%
}
```

11.3.1 The psutils suite

A general set of tools to manipulate PostScript files is Angus Duggan's `psutils`, freely available in the CTAN archives, `support/psutils`. It is a set of Unix shell or Perl scripts and programs written in C that can be compiled easily on Unix and other systems. These programs are not interactive or graphical, but are controlled by command-line switches. Since these can be quite complex, with lots of options, we indicate the *optional* parameters with gray shading: `width`. Dimensions, when required, can be specified in inches (3in) or centimeters (10cm). Many of the programs have an option to run quietly (`-q`) instead of printing page numbers as they process them, and most either read from standard input and write to standard output or have two parameters of an input file and an output file.

The programs operate on a page by page basis. Almost always, the original PostScript file must contain enough of the Adobe Document Structuring Conventions to allow the program to identify pages. Most `dvi` drivers produce PostScript files that meet these criteria, but you cannot assume that any PostScript you find is suitable. A set of filters is provided to “correct” files from various applications that generate non-standard output.

As an example of using `psutils`, let us look again at methods for putting multiple logical pages on one output page; the `psnup` program is a fast and efficient way of doing this. Figure 11.4 on the next page shows the result with a command line starting with:

```
psnup -9 -d
```

which asks for nine logical pages on one physical page, with borders around each one.

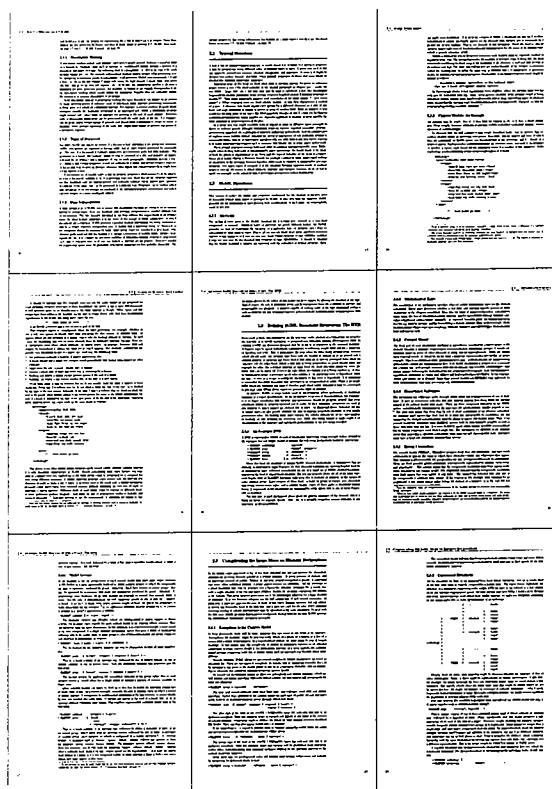


Figure 11.4: Nine logical pages on one output page.

pstops does the same job but gives you much more detailed control on placing and scaling pages. For instance, to get four logical A4 pages on one physical A4 sheet of paper, you would use the command:

```
pstops '4:00.5(0,14.5cm)+10.5(10.5cm,14.5cm)' \
'+20.5(0,0.5)+30.5(10.5cm,0.5)'
```

Table 11.1 on the facing page gives an overview of the programs and scripts available in psutils. The sections that follow describe some of the most important ones.

Table 11.1: Overview of the tools in the psutils set.

<i>Programs, Perl[†] and shell[†] scripts</i>	
psbook	rearrange pages into signatures
psselect	select pages and page ranges
pstops	general utility for rearranging and selecting pages
psnup	combine multiple pages on a single physical sheet
psresize	alter document paper sizes
epsffit	fit an encapsulated PostScript file inside a given bounding box
getafm [‡]	output PostScript to retrieve the AFM file of a font from a PostScript printer
showchar [‡]	output PostScript to draw a character with metric info
fixdlsrps [†]	fix dvilaser/PS output so that the psutils programs work properly
fixfm [†]	fix FrameMaker documents
fixmacps [†]	fix (some) Macintosh output
fixpsditps [†]	fix Transcript psdit output
fixpspps [†]	fix psprint PostScript output
fixscribeps [†]	fix Scribe PostScript output
fixtpps [†]	fix troff Tpscript output
fixwfwps [†]	fix Microsoft Word for Windows output
fixwpps [†]	fix WordPerfect output
fixwwps [†]	fix Windows Write output
extractres [†]	extract resources from PostScript files
includeres [†]	include resources into PostScript files
psmerge [†]	script to merge multiple PostScript files

pstops: rearranging pages in a PostScript file

```
pstops -q -b -wwidth -hheight -ppaper -dlinewidth
pagespecs infile outfile
```

pstops rearranges pages in a very general way; this allows printing *n*-up, making booklets, reversing, scaling, etc. It is the most general tool in the set, and most users find it easier to use the higher-level psnup or psbook. The options are:

- b Prevent bind operators in the PostScript prolog from binding. This may be needed when complex multi-page rearrangements are being made.
- dlinewidth Draw line of width *linewidth* around each page; if just -d is given, the width defaults to one point. It is important to realize that the width is relative to the size of the *original* page and is scaled. If you put four logical pages on one sheet of paper and specify -d 2pt, the line actually drawn is 1pt.
- hheight The height used by the h dimension specifier.

-ppaper Set a named paper size; possible choices are a3, a4, a5, b5, letter, legal, tabloid, statement, executive, folio, quarto or 10x14 (the default paper size is a4).

-q Run quietly.

-wwidth The width used by the w dimension specifier.

The parameter `pagespecs` allow you to specify how pages are combined and processed, as follows:

```
pagespecs = [modulo:]specs
specs     = spec[+specs] [, specs]
spec      = [-]pageno [L] [R] [U] [%scale] [(xoff,yoff)]
```

modulo The number of pages in each block; must be >0 (default is 1).

specs The page specifications for the pages in each block. The value of `pageno` in each `spec` should be $0 \geq$ (first page in block) and $\leq \text{modulo}-1$ (last page in block). The optional dimensions `xoff` and `yoff` shift the page by the specified amount. These dimensions are given by default in PostScript points, but may also be specified in centimeters or inches by following them with the string `cm`, or `in`. Alternatively, one can specify the flag `w` or `h`, meaning that one is using multiples of the width or height. The optional parameters `L`, `R`, and `U` rotate the page left, right, or upside-down. The optional parameter `scale` scales the page by the specified fraction. If the optional minus sign is specified, the page number is relative to the end of the document, instead of the start.

Page `specs` separated by `+` affect pages that are to be merged into one page; page `specs` separated by a space to be on separate pages. If there is only one page specification, and `pageno=0` then `pageno` may be omitted.

The shift, rotation, and scaling are performed in sequence, regardless of the order in which they appear on the command line.

psnup: put multiple pages on a single sheet

```
psnup -wwidth hheight ppaper -wwidth hheight ppaper
      -l r e c amargin bborder alinewidth sscale
      -nup c0 infile outfile
```

psnup takes a PostScript file and puts several logical pages on each physical sheet of output paper. The options are:

-bborder Leave a margin of width `border` around each (logical) page.

-c Arrange the pages down columns on the page, rather than in rows.

-dlinewidth Draw a line of width `linewidth` around each page; if just `-d` is specified, the width defaults to 1 point. It is important to realize that the width is relative to the size of the *original* page and is scaled. If you put four logical pages on one sheet of paper, and specify `-d 1pt` the line actually drawn is 1pt.

-f Swap the page's height and width.

-hheight Set the height of the output pages.

-Hheight Set the height of the input pages.

-l Print the output in landscape mode (rotated 90° anti-clockwise).

-mmargin Leave a margin of width `margin` around the whole output page.

-nup Set the number of logical pages to put on each output page. This can be any arbitrary number, and `psnup` does its best to find a suitable arrangement, but may give an error if it cannot find anything sensible.

-ppaper Instead of explicitly setting the size of the output pages, this option gives a named page size; the possibilities are a3, a4, a5, b5, letter, legal, tabloid, statement, executive, folio, quarto or 10x14 (the default paper size is a4 for both input and output).

-Ppaper Same as `-p`, for input pages.

-r Make the output print in "seascape" mode (rotated 90° clockwise).

-sscale Set `scale` explicitly for page reduction. This is necessary if you have pages that are already the right size, and just want them arranged *n*-up on the output pages.

-wwidth Set the width of the output pages.

-Wwidth Set the width of the input pages.

psbook: rearrange pages in a PostScript file into signatures

```
psbook -q -signature infile outfile
```

psbook takes the pages in a PostScript document and creates a new file in which the pages are rearranged as "signatures", the sections in a printed book. In printing a book, a number of pages are printed on a large sheet of paper that is then folded and bound to make the final volume. In making a booklet by hand, it is helpful to arrange the pages so that when folded they come out in the right order. The options to the program are:

-q Run quietly, without showing the numbers of the pages being processed.

-signature Set signature size (default 1), i.e., the number of sides (a multiple of four), to be folded and bound together in the proposed book. Extra blank pages are added to make up the right number if needed.

psselect: selecting pages from a PostScript file

```
psselect [-e] [-o] [-ppages] [-r] [-q] [-r] infile outfile
```

psselect takes a subset of pages from the input file and writes a new file. The options are:

- e Select even-numbered pages only; this can be used in conjunction with the other options.
- o Selects odd-numbered pages only.
- ppages Specify the pages to be selected, with a comma-separated list of page ranges; these can be either a single page number or a range of two numbers separated by a colon (:). The first or last number in a range can be omitted and defaults to the first and last pages in the file. Page numbers preceded by an underscore (_) are relative to the end of the document, counting backwards. It is important to realize that page numbers are absolute, i.e., start at one at the start of the document; the *printed* page number may be something else entirely.
- q Run quietly.
- r Output the selected pages in reverse order.

psmerge: merging PostScript files

```
psmerge outfile.ps file1.ps file2.ps
```

psmerge merges a set of PostScript files into one output file. This is quite a difficult thing to do, and can really be successful only if all the files come from the same application, since the specification of fonts and other resources tends to vary quite widely. The options are:

- outfile.ps Specify the name of the output file.

psresize: scale and resize PostScript

```
psresize [-w] [-h] [-p] [-w] [-h] [-p] [-q] infile outfile
```

psresize takes an input PostScript document, rescales and centers it to fit on different-sized paper. The options are:

- height Set height of the output page.
- Hheight Set height of the input page.
- ppaper Set a named paper size for output; the possibilities are a3, a4, a5, b5, letter, legal, tabloid, statement, executive, folio, quarto or 10x14 (the default paper size is a4).

-Ppaper Same as -p but for the input page.

-q Run quietly.

-wwidth Set width of the output page.

-Wwidth Set width of the input page.

One often needs to print, on a printer with letter-size paper, a document that was set up for A4 paper. To transform the file, you could use a command like:

```
psresize -pletter -Pa4 a4file.ps letter.ps
```

Handling resource information in a PostScript file

extractres looks for "resources" in a PostScript document and writes them to files with the same name as the resource (with an appropriate file extension). Conversely, includeres physically inserts requested resources into a PostScript document. Resources are searched for in the current directory and the system default directory under the resource name.

Resources are elements of a PostScript file that are common to many different documents; the obvious examples are fonts and sets of PostScript procedures (procsets), but Level 2 PostScript also supports resources like patterns and forms. They are requested in the prologue of a PostScript document as comments of the form:

```
%%IncludeResource
```

The resources actually present in a PostScript file should be noted in a comment of the form

```
%%DocumentSuppliedResources
```

At present, the scripts described here do not change such comments, so the result is not quite perfect.

A possible application for these utilities is to rewrite a document containing several copies of resources so that it has just one copy of each in a clean way. Using Unix pipes, you can perform this job with a command like:

```
extractres file1.ps | includeres > file2.ps
```

You could also use extractres to downsize files for sharing with colleagues by email, if you are sure they have the same fonts and procsets to re-include when they receive your file.

The programs are:

```
extractres [-m] < infile.ps > outfile.ps
includeres < infile.ps > outfile.ps
```

- m Combine resources with the same name into a single file.

epsffit: fitting EPS files into a constrained size

```
epsffit -c -r aa em ee llx lly urx ury infile outfile
```

epsffit puts suitable PostScript code in an EPS file to make it fit into a new bounding box. The bounding box is specified in the form *llx lly urx ury*, where *llx lly* are the *x* and *y* coordinates (in PostScript points) of the lower left corner and *urx ury* of the upper right corner. The options are:

- a Adjust the aspect ratio to fit the bounding box (by default the aspect ratio is preserved).
- c Center image in the bounding box.
- m Rotate image to maximize its size if that would fit the bounding box better.
- r Rotate image by 90 degrees counterclockwise.
- s Add a showpage command at the end of the output file to force the image to print.

11.4 Ghostscript, a PostScript interpreter

Aladdin Ghostscript is a freely available PostScript interpreter written by L. Peter Deutsch. Strictly speaking, the program interprets Ghostscript, a programming language “similar to” Adobe Systems’ PostScript language, but only a very few PostScript Level 2 features are not completely implemented. In addition, it interprets the Portable Document Format language. As the source code of Ghostscript is written in C and is freely available on Internet archives, the program has been compiled on almost all possible computer platforms. Ghostscript can be used for various purposes:

- *Previewing PostScript code.* Ghostscript displays your PostScript on screen (supporting a wide variety of devices), making possible fast checking of your drawings before sending them to a printer; in this case you may prefer to use more sophisticated interfaces available for Unix X Windows System and Microsoft Windows.
- *Preparing output for various printing devices.* The Ghostscript program comes with a large set of drivers, so PostScript code can be transformed into a format that is understood by most printers.
- *Converting into raster formats.* Ghostscript can convert PostScript into many different raster formats, such as TIFF, PCX, and PBM.
- *Manipulating PostScript.* Ghostscript performs various helper tasks, such as extracting text from PostScript files, calculating the bounding box of an EPS file, and converting PostScript into Adobe Illustrator format.
- *Multi-platform PDF reader.* Recent versions of Ghostscript also read PDF (Portable Document Format) files, so that you can view Acrobat files on those platforms on which Adobe’s free Acrobat Reader is not yet available. The most recent versions of Ghostscript can even “distill” PostScript code into PDF.

How to start and use Ghostscript depends on the operating system. It is beyond the scope of this book to cover how Ghostscript is obtained, installed, possibly compiled, and set up; Thomas Merz’ excellent book on PostScript and PDF (Merz, 1996) has a detailed appendix that should answer most questions; this is also freely available on the World Wide Web at <http://www.muc.de/~tm>. In this section, we assume a Unix, DOS or VMS command-line interface; Windows and Macintosh users work in a slightly different way.

The command

```
gs options files ...
```

reads each of the files and executes the PostScript commands contained in them. Depending on the options, the result may be displayed on screen, passed to some other process, or written to a file. When all files have been read, the program normally prompts for further keyboard input. To exit the interpreter, enter the command quit.

11.4.1 Ghostscript options and initialization

The Ghostscript command-line options perform three functions: they let you supply files to run, change options, and give values to various PostScript objects that can be accessed by the program or your files.

The options are as follows:

@filename Read file *filename* and treat its contents as if they were typed on the command line; this is for use on systems like DOS that forbid long command lines. The options or file names in the file can be on as many lines as convenient.

--filename ...

-+ filename ... Interpret *filename* as normal but take all remaining arguments and write them in the PostScript userdict as an array of strings called ARGUMENTS.

-c tokens ... Interpret the arguments, up to the next argument that begins with - followed by a non-digit or with @, as PostScript code. You can use this, for instance, to force execution of a PostScript showpage at the end of a file *picture.ps* by the command:

```
gs picture.ps -c showpage
```

More commonly, command lines are ended with **-c quit** to make Ghostscript quit after the files are finished.

-Dname=value

-dname=value Define a name in systemdict with the given value, which must be a valid PostScript token. If no value is supplied, it is set to true.

-f filename Run the file, even if its name begins with a - or an @. **-f** by itself does nothing, but it can be used to end a set of -c options.

- gn1xn2* Equivalent to the two options `-dDEVICEWIDTH=n1` and `-dDEVICEHEIGHT=n2`. This lets you set the width and height of the output device.
- Idirectories* Add the list of directories to the front of search path for system files; directories are separated by : under Unix and by ; under DOS or VMS.
- P* Look first in current directory for system files (this is the default).
- P-* Do *not* look first in current directory for system files.
- q* Run quietly, with no startup messages.
- rnumber*
- rn1xn2* This sets the resolution of the output device, such as a printer with varying resolution, or a bitmap file. It has the same effect as using `-dDEVICEXRESOLUTION=n1` and `-dDEVICEYRESOLUTION=n2`. Since some printing devices have different *x* and *y* resolutions, the second form of the option allows you to set this.
- Sname=string*
- sname=string* Define a name in `systemdict` with a given string as value. The difference between this and `-d` is that the value is a PostScript *string*, not a token.
- uname* Undefine a name, cancelling `-d` or `-s`.
- Tell Ghostscript that the standard input is coming from a file or a pipe. Ghostscript reads from `stdin` until reaching an end-of-file, executing it like any other file, and then continues processing the command line. At the end of the command line, Ghostscript exits, rather than going back to an interactive prompt. Due to the way PDF files are structured, you can only use this switch with PostScript input files.

In practice, the `-d` and `-s` options are normally used to set some of the special names that Ghostscript defines; the following is a list of commonly used properties you may need to set, but it is not exhaustive—many device drivers introduce additional features.

- dCOLORSCREEN*
- dCOLORSCREEN=0*
- dCOLORSCREEN=false* Force the use of separate halftone screens on devices with a resolution over 150 dpi; `-dCOLORSCREEN=0` uses separate screens with the same frequency and angle; `-dCOLORSCREEN=false` forces the use of a single binary screen. The default behavior is to use separate screens with different angles if the device has fewer than 5 bits per color.
- sDEVICE=device* Set the output device.
- dDISKFONTs* When fonts are requested, they are loaded character by character from the disk files. Normally, Ghostscript reads the whole font into memory, but this option may be needed if memory is short (although it slows the program down).

- dFirstPage=page* Start interpreting a PDF document at page *page*.
- dFIXEDMEDIA* Fix the media size after initialization, so that pages of other sizes or orientations that are requested in the input files are rotated or scaled automatically.
- dFIXEDRESOLUTION* Fix the media resolution after initialization, regardless of what the input file says; the `-r` option also sets this.
- sFONTMAP=filename* Set the name of a Fontmap file to load.
- sFONTPATH=directories* Set a list of directories that is scanned automatically for fonts.
- dLastPage=page* Stop interpreting a PDF document after page *page*.
- dNOBIND* Disable the PostScript bind operator (useful for debugging).
- dNOCACHE* Disable character caching (useful for debugging).
- dNOCIE* Substitute DeviceGray and DeviceRGB for CIEBasedA and CIEBasedABC color spaces respectively. This speeds up printing if great color accuracy is not needed.
- dNODISPLAY* Do not start a screen display; this is used when you just want to see what the program says about your input.
- dNOFONTMAP* Do not load a Fontmap file.
- dNOGC* Disable the garbage collector in Level 2 PostScript.
- dNOPAUSE* Disable the prompt and pause at the end of each page. This is often specified when driving a printer.
- dNOPLATFONTS* Disable the use of fonts supplied by the underlying platform (e.g., X Windows System or MS Windows).
- dNOPROMPT* Do not display a prompt at the end of each page; this is different from `NOPAUSE`, as the program still halts, but the prompt does overwrite the picture on some displays.
- sOutputFile=filename* Set a file name for output.
- sPAPERSIZE=papersize* Select one of the paper sizes known to Ghostscript, e.g., `-sPAPERSIZE=a4` or `-sPAPERSIZE=legal`. A default size can be specified in the initialization file `gs_init.ps`, and the valid names are in the file `gs_statd.ps`.
- dQUIET* Run quietly with no messages.
- dSAFER* Stop the PostScript `deletefile` and `renamefile` operators from working, and do not allow files to be opened for writing. It is sensible to set this if you use Ghostscript to display other people's documents, as the PostScript language has full power to delete all your files.
- sSUBSTFONT=fontname* Substitute *fontname* for all unknown fonts; Ghostscript normally tries to find a suitable font, and uses Courier as a last resort.

When Ghostscript starts, it needs to find a set of initialization files (much of the interpreter's functions are written in PostScript itself and loaded when the program starts). Normally, the program is set up to look in the current directory for your files and in a system directory for the program files, but you may need to override this. You can ask for extra directories to be searched by using the `-I` option on the command line; or you can set an environment variable `GS_LIB` to point to directories. Directories can be specified as a list, separated by a `:` under Unix, or by `;` under DOS or VMS.

11.4.2 Ghostscript and fonts

In addition to its system files, Ghostscript needs to find fonts. Unlike most PostScript printers, it normally has no built-in fonts at all, but loads everything as PostScript Type1 fonts or TrueType fonts from your disk. Ghostscript comes with a basic set of 35 fonts to match those found in most PostScript printers. These are good-quality Type 1 PostScript fonts made freely available by the German font company URW. It starts by looking for a file catalog called `Fontmap` (you can change the name with `-sFONTMAP`) among the system files. This file specifies how to relate PostScript font names to actual files on the system. It then searches any of the directories specified with the `GS_FONTPATH` environment variable or `-sFONTPATH` option for files that appear to contain PostScript fonts. If it decides they are valid fonts, Ghostscript adds those files and fonts to its internal copy of the `Fontmap` catalog.

When your files are read, font requests are satisfied using the `Fontmap` catalog; if no match is found, the default font is used. Some Unix systems (such as Sun's OpenWindows) have a Display PostScript setup with a set of PostScript Type1 fonts. It makes sense to set `GS_FONTPATH` to point to these, so that all fonts are automatically available.

The `Fontmap` catalog file consists of a set of lines describing a font; each line has the name of a PostScript font (prefixed with `/`), followed by some white space, and then the name of a file on disk, in parentheses. Alternatively, the second part of the line can be the name of an already defined font, to act as an alias. The line must be terminated by a semi-colon. Comments are added by starting a line with a `%` character.

As an example, the familiar Palatino family is provided by URW's Palladio fonts; this extract from `Fontmap` shows how the names are mapped to actual files on disk:

```
/URWPalladioL-Roma      (p0520031.pfb) ;
/URWPalladioL-Ital      (p0520231.pfb) ;
/URWPalladioL-Bold      (p0520041.pfb) ;
/URWPalladioL-BoldItal  (p0520241.pfb) ;
```

The full path name of the file is not normally given, since the fonts are expected to be found in the font directory path. In order to allow files to call up the font by the real Palatino names, a set of aliases is provided:

```
/Palatino-Roman      /URWPalladioL-Roma ;
/Palatino-Italic     /URWPalladioL-Ital ;
```

```
/Palatino-Bold      /URWPalladioL-Bold ;
/Palatino-BoldItalic /URWPalladioL-BoldItal ;
```

Since Ghostscript normally searches the directories for suitable font files anyway, the first example above is actually redundant, but the alias entries in `Fontmap` are vital, unless you have a copy of real Palatino on your disk. Users of Adobe Type Manager *do* have real Adobe fonts that Ghostscript can use, and special versions of the `Fontmap` file are provided for this and other setups.

11.4.3 Using different devices with Ghostscript

Ghostscript supports a great many output devices; which ones are available depends on how your copy of Ghostscript was compiled, but all versions support screen preview, some printers, and some bitmap file formats. You can see which devices are available by typing `gs -h` at the command line, or by starting the program and then typing:

```
devicenames ==
```

The result for a typical Unix installation looks like this (note the option `-sNODISPLAY` to prevent Ghostscript from starting a screen window, since we simply want to query the interpreter):

```
> gs -sNODISPLAY
Aladdin Ghostscript 4.03 (1996-9-23)
Copyright (C) 1996 Aladdin Enterprises, Menlo Park, CA.
All rights reserved. This software comes with NO WARRANTY:
see the file PUBLIC for details.
GS>devicenames ==
[/ljet3 /pjxl300 /ljet2p /nullpage /tiffpack /pgm /ljetplus
/tiff1zw /pbmraw /pjxl /laserjet /tiffg4 /pbm /pj /png256
/tiffg32d /pcx24b /dje-500 /cdj550 /png16 /tiffg3 /pcx256
/deskjet /pnggray /tiffcrle /pdfwrite /pcx16 /x11mono
/pngmono /ppmraw /pcxgray /x11cmk /bitcmk /ppm /pcxmono
/x11alpha /cdjmono /bitrgb /pnmraw /faxg4 /bjc600 /cdjcolor
/bit /pnm /faxg32d /x11 /cdeskjet /psmono /pgmraw /faxg3
/bj200 /png16m /tiff24nc /pgnm /bjc800 /bj10e /ljet4
/tiff12nc /pgmraw]
GS>quit
>
```

Some of the devices available in Version 4.03 of Ghostscript (the latest at the time of writing) are listed in Appendix A.8 on p. 494.

You can choose the device with the command line option `-sDEVICE=device`. For example, to generate output for a color Deskjet 550C printer (if the program is compiled with

that driver included, as is the case above), you could issue the command:

```
gs -sDEVICE=cdj550 file.ps
```

Note that you still have to take care to direct the output to the actual printer; we discuss this presently.

Alternatively, from inside the program, you can type:

```
(cdj550) selectdevice
(file.ps) run
```

In this case it is easy to switch devices by issuing another `selectdevice` command at any time, e.g.,

```
(cdj500) selectdevice
(cat.eps) run
(tiffg4) selectdevice
(cat.eps) run
```

This generates a printable picture from `cat.eps` for your color Deskjet and then a monochrome compressed TIFF picture.

If you always want to use the same device, you can set the `GS_DEVICE` environment variable to it. This can be overridden on the command line, of course.

When a printer or a graphics bitmap format is chosen as the device, output is directed into a file (by default, Ghostscript sends it to a scratch file in the system temporary directory specified by the `TEMP` environment variable). Alternatively, you can say where you want the output to go with a command-line option:

```
-sOutputFile=outfile
```

For a file with more than one page, you can make a *series* of files by using the syntax:

```
-sOutputFile=name%d.ext
```

The `%d` is a C or Unix `printf` format specification that is replaced with the number of each page (another possibility would be `%02d`, where in this case two decimal digits are used). Each file receives one page of output; the file extension (*ext*) is up to you. For example, multipage PostScript document called `doc.ps`, you could make a series of TIFF files called `doc01.tif`, `doc02.tif` etc. with the command:

```
gs -sOutputDevice=tiffg4 -r600 -dNOPAUSE -sOutputFile=doc%02d.tif doc.ps
```

We use the `-dNOPAUSE` option to stop the program from pausing after each page, and we set the resolution at 600 dpi (see below). A similar procedure can be used to send generated fax pages from your computer. When generating bitmaps, you can ask Ghostscript to

perform anti-aliasing by setting `TextAlphaBits` and/or `GraphicsAlphaBits`; by default these are set to 1 (no anti-aliasing), but values of 2 or 4 turn on the feature (e.g., `-dTextAlphaBits=4`). Anti-aliasing is a technique commonly used to make on-screen text more readable that involves setting some of the pixels in letters or lines to a shade of gray, to make the transition from black to white less abrupt (see, e.g., Foley et al., 1990, pp. 132ff for a detailed explanation). We demonstrate this, and other details of generating bitmap files using Ghostscript, in Section 11.6 on p. 455.

If your system supports “pipes” (like Unix), you can direct the output of Ghostscript directly to the printer with a command option like:

```
-sOutputFile=\|lpr
```

This directs the output of Ghostscript to the *standard input* stream of the `lpr` program (used on Unix to spool output to a printer). Under Windows 95 or NT, you can specify a UNC path name for your printer, such as:

```
-sOutputFile="\\spool\printername"
```

More generally, the output can be sent to the standard output (`stdout`) for piping purposes with the option

```
-sOutputFile=-
```

If you do this, you should use the `-q` (quiet) switch to ensure that Ghostscript does not interleave your output with its messages.

For some printers and most bitmap file formats, you need to specify the resolution of the output; this can be set with the `-r` option:

```
gs -sDEVICE=device -rresolution
```

Some devices have different *x* and *y* resolutions, and you can set these separately if needed. For example, for a 9-pin Epson-compatible printer, you can specify the lowest-density (fastest) mode with:

```
gs -sDEVICE=epson -r60x72
```

and the highest-density output mode is specified with:

```
gs -sDEVICE=epson -r240x72
```

In practice, most users who want to control their printers with Ghostscript either use the `GSview` or `ghostview` programs, which wrap up the details for them in a convenient interface, or integrate Ghostscript into Unix print-spooling systems. The Ghostscript documentation file `unix-lpr.txt` provides hints for how to do this, but experience in Unix system programming is required.

wrapped up in a script called `ps2ascii`, used as follows:

```
ps2ascii myfile.ps myfile.txt
```

Another utility, `pstotext`, developed by Andrew Birrell and Paul McJones, is a little more robust. Like `ps2ascii.ps`, this reads one or more PostScript files and writes out a representation of the plain text that would be displayed if the PostScript file were printed. Internally `pstotext` uses Ghostscript and loads a PostScript library that locates each string in the file, and enough additional information to approximate the string's bounding rectangle. `pstotext` then post-processes this information and outputs a sequence of words delimited by space, newline, and formfeed.

`pstotext` translates all eight-bit input codes into an ISO 8859-1 character code. When a character is not present in ISO 8859-1, it produces a sequence of characters, e.g., "---" for em dash.

Creating EPS Interchange files

The utility script `ps2epsi` takes a PostScript file as input and generates an output file that conforms to Adobe's Encapsulated PostScript Interchange (EPSI) format. This special form of Encapsulated PostScript adds a bitmap version of the final displayed page (in the form of PostScript comments) to the beginning of the file. This is useful for text processors or desktop publishing tools like FrameMaker which use it to display an approximate representation of the picture in the regular display. The script is used as follows:

```
ps2epsi myfile outfile
```

If the output file is not specified then the same name as for the input file name is used, but with the extension `epsi`.

Preparing Adobe Illustrator files

The Adobe Illustrator program is popular for preparing and editing sophisticated illustrations. It uses highly structured and simplified PostScript as its native storage form, and Ghostscript comes with a PostScript program, `ps2ai.ps` that tries to convert normal PostScript into a form Adobe Illustrator can digest. There are some restrictions in the PostScript code that it can understand, but in general the code generated can be used by other programs that can input Adobe Illustrator-compatible PostScript code. The file can be used as follows:

```
gs -q -dNODISPLAY ps2ai.ps infile.ps > outfile.ai
```

A few variables are available for controlling the process (`jout` for writing to a file, `jtxt3` for outputting in Adobe Illustrator version 3 format, `joutln` for replacing fonts by their outlines, etc.); the file `ps2ai.ps` gives more details.

Working with PDF (Portable Document Format)

ghostscript comes with two scripts, `ps2pdf` and `pdf2ps`, that convert, respectively, ordinary PostScript to Portable Document Format and PDF to PostScript. PDF, the format used by Adobe Acrobat products, is effectively a less complex and structured version of PostScript; the programming language features have been removed, facilities for hypertext and intelligent font substitution have been added, and the structure of PDF files is optimized for fast screen display. Both programs are very simple scripts that, as usual, call Ghostscript with the right parameters and PostScript program files. They are used with command lines like this:

```
ps2pdf myfile.ps outfile
pdf2ps myfile.pdf outfile
```

11.5 PostScript font to PK font format conversion

The majority of TeX previewers cannot display fonts unless they are available in the `pk` format (or the older, obsolete, `pxl` format), like that generated from METAFONT. The exceptions are those previewers that are tightly integrated with a sophisticated font display system like Adobe Type Manager under Windows or the Macintosh operating system. Thus the NeXTStep TeX viewer, Blue Sky's Textures for Macintosh, Y&Y's dviwindo, and Kinch's TrueTeX previewer for Windows all use the *system* software to render fonts, using PostScript Type 1 or True Type sources. This is undoubtedly the best way to work, since the previewer can concentrate on the `dvi` file and the placement of words, leaving all the precise character-glyph rendering to specialist machinery. Alternatively, the `dvi` file can be converted to PostScript and displayed using a PostScript viewer like Ghostscript, but many users find this two-stage process too slow.

There are, however, many TeX users who do not work in Windows or Macintosh environments and do not want to invest in commercial software but would like to take advantage of the excellent TeX previewers already available for their systems. Such users can create `pk` format font files from PostScript Type 1 fonts by using one of two freely available programs, `ps2pk` and `gsftopk`. The quality obtained depends on the rendering engines used, and you should look very carefully at the output if you plan to do serious typesetting with these bitmap fonts; but for previewing they are undoubtedly excellent.

11.5.1 The `ps2pk` program

`ps2pk` was written by Piet Tutelaers using code donated by IBM to the X Consortium for rendering PostScript Type 1 fonts. It is a free-standing program that reads a Type 1 font and its Adobe Font Metric (`.afm`) file, and writes a `pk` file at the requested resolution.

```
ps2pk -v -Xxresolution -Yyresolution -Ppointsize
      -Slant -eencoding -eextension -aAFMfile
      -mmodename -raspectratio -Rbaseresolution
      type1fontname pkfontname
```

The meaning of the different options is:

- aAFMfile Set the name of the afm file; normally this is taken to be the same as that of the Type 1 font.
- eenc Set an encoding file (the default is that specified in the afm file); the format is the same as that used by dvips.
- Eextension Stretch or shrink the font (default is 1.0, i.e., no change).
- Ppointsize Set the point size of the result (default is 10pt).
- Slant Slant the font (default is no slant).
- Xxres Set the resolution in the *x* direction in dots per inch (default is 300).
- Yyres Set the resolution in the *y* direction in dots per inch (default is 300); normally this is the same as the *x* direction and is not specified.
- v Be verbose, i.e., provide information about what is happening as the program runs.

Some options are primarily for compatibility with other programs that generate pk files (like METAFONT), to make sure enough information gets into the output file:

- mmodename Set a mode name as used by METAFONT; this is used to write identification information into the output.
- raspectratio Set the aspect ratio (by default 300/300). Again, this is used for identification information written to the output.
- Rbaseres Set the base resolution you are working at (default is 300dpi); the difference between this and the *x* resolution (if any) is written to the output as a TeX magnification.

The use of the -X and -P options may cause some confusion. It is important to appreciate the theoretical difference between a font rendered at twice normal size for a 300-dpi device and one rendered at normal size for a 600-dpi device: they both have the same resolution (600 dots to the inch) for different purposes. The results are named differently, and the internal identifiers make the distinction clear. Thus the two commands:

```
ps2pk -P20 Times-Roman.pfb
ps2pk -X600 Times-Roman.pfb
```

yield Times-Roman20.300pk and Times-Roman10.600pk respectively as their output

files. In practice, using the font naming scheme and the 8r encoding advocated in Chapter 10, our command lines are more likely to be of the form:

```
ps2pk -e8r.enc -mcx -P20 ptmr8a.pfb cx/dpi600/ptmr8r.pk
ps2pk -e8r.enc -mljfour -X600 ptmr8a.pfb ljfour/dpi600/ptmr8r.pk
```

where the output pk files are explicitly named, complete with directories, to make it entirely clear which file is intended for which device.

The specification of encoding, slant, and stretch or shrink is directly comparable to that in dvips' font map files (see Section 11.2.4 on p. 428).

ps2pk finds Type 1 fonts and their afm files using the environment variable T1INPUTS. ps2pk is supplied fully installed in the TeTeX Unix TeX package, but is also available compiled in DOS and OS/2 versions on the CTAN archives.

11.5.2 The gsftopk program

Paul Vojta's gsftopk performs the same task as ps2pk but uses Ghostscript to do the rendering of characters from PostScript Type 1 fonts into TeX's pk form. It is used primarily on Unix systems. The program is called as follows:

```
gsftopk -q fontname dpi
```

where the -q option tells the program to run quietly. The required parameters are the font name (such as ptmr8r) and the resolution (such as 600). The result is a pk file called ptmr8r.600pk.

gsftopk needs to find TeX font metric (tfm) files for fonts and uses the same map file of PostScript font names and re-encoding information as dvips (see Section 11.2.4 on p. 428); it is normally used inside a MakeTeXPK script where the right paths, etc., are correctly set, and not directly by the user. Like ps2pk, it is supplied fully installed in the TeTeX Unix TeX package.

11.6 Generating images for Web pages using dvips and Ghostscript

Many people these days want to publish their L^AT_EX documents on the World Wide Web. However, until there is full support for mathematics in HTML, or until some of the helper equation viewers for Web browsers become more widely available, you cannot perform a full conversion. Math formulae and L^AT_EX pictures are, therefore, commonly converted to GIF images. This section explores two aspects to this: creating reusable PostScript pictures of portions of L^AT_EX output and converting that PostScript sensibly to GIF format.

To produce reusable PostScript output from dvips (and this includes output destined for Acrobat Distiller), the absolute priority is to use outline fonts. If you do not use outline

$$-\Phi_0 \frac{\partial}{\partial \varphi} (\Phi_1 a \sin \varphi) - \Phi_1 \frac{\partial}{\partial \varphi} (\Phi_0 a \sin \varphi) - A_1 \left[\Phi_0 + \frac{\partial}{\partial a} (a \Phi_0) \right] \sin \varphi = -a \Phi_0 f \sin \varphi \quad (1)$$

Figure 11.6: Bitmap EPS file, enlarged and distorted.

$$-\Phi_0 \frac{\partial}{\partial \varphi} (\Phi_1 a \sin \varphi) - \Phi_1 \frac{\partial}{\partial \varphi} (\Phi_0 a \sin \varphi) - A_1 \left[\Phi_0 + \frac{\partial}{\partial a} (a \Phi_0) \right] \sin \varphi = -a \Phi_0 f \sin \varphi \quad (1)$$

Figure 11.7: Outline font EPS file, enlarged and distorted.

fonts and reuse your output scaled up, the effect is horrible; compare Fig. 11.6 with Fig. 11.7. If you want to turn your documents into PDF, Distiller produces very poor results from PK fonts.

The second priority is to get the right bounding box. Surprisingly many applications cheat by simply making it the page size, regardless of whether the whole area is used. dvips does this by default too, but has the command-line option `-E` that makes dvips calculate the actual extent used. Note that EPS files are, by definition, only one page, so you also must use dvips options to select just one page. There are two caveats when preparing the input. First, make sure you do not include a page number (use `\pagestyle{empty}` in L^AT_EX) since otherwise the bounding box covers that too. Second, dvips does not always work out the text extent correctly. For instance, if you wrote:

```
Hello\raisebox{10pt}[0pt][0pt]{Up there}!
```

you would be asking L^AT_EX to raise *Up there* off the baseline but to pretend that it has no effect on the height calculation. dvips believes this and calculates a bounding box based on the *claimed* height. If you use complicated packages like PSTricks, which add in arbitrary PostScript code, you can also end up in real trouble. In these cases you can either adjust the `%BoundingBox` line in the PostScript file by hand or place invisible marks in L^AT_EX to make sure that dvips recognizes the full extent.

If you think that L^AT_EX knows what you want but dvips does not, a useful trick to remember is to make judicious use of color. Suppose you want to use PSTricks to encircle a

mathematical symbol. You might write:

11-6-1

absurd 

```
\usepackage {pstricks,graphicx}
absurd \pscirclebox{\surd$}
```

L^AT_EX leaves the right space, since the PSTricks macros understand what is going on, but dvips is told to draw the circle in raw PostScript, and the bounding-box calculation ignores that. The result is that the limits are set just around the size of the letters. If you write:

```
\framebox{absurd \pscirclebox{\surd$}}
```

it works correctly, because dvips looks at the enclosing frame, not just the words. But you end up with an unwanted box; you should make the box (in effect) invisible by writing:

```
{\color{white}\fboxsep{0pt}%
\framebox{%
{\color{black}absurd \pscirclebox{\surd$}}%
}%
}
```

This creates a white frame around black text; L^AT_EX proceeds happily and so does dvips, calculating the right extents, but nothing shows on paper. Obviously, this works only in a monochrome environment.

Another method for getting an accurate bounding-box in your dvips output is Russell Lang's program `epstool`, distributed with G^Sview for Windows; we used it extensively in preparing examples for this book. The program adds a preview header to a PostScript file, but as a side effect it can calculate an accurate bounding box, as it uses Ghostscript to create a bitmap whose size it can then examine. You can then use the same tool to remove the preview header if you like. This tool, with other techniques, is discussed in (Merz, 1996, pp. 97–99).

Why would you do all this in practice? Often, as we have said, because you want your L^AT_EX mathematical output on the World Wide Web, and one way to do this is to embed GIF images in your HTML. The sophisticated `latex2html` program does all this for you. The technique of this program is worth understanding, as it is a general utility; the sequence of events in the converter (a large Perl script) is as follows:

1. place bits of L^AT_EX in a special file, one fragment per page, with no page number;
2. run L^AT_EX to generate a multipage dvi file;
3. use dvips `-i` and `-S` options to generate one self-contained output file per page;
4. ask Ghostscript to render each page in PBM (Portable Bitmap) form;
5. use the PBMplus/Netpbm utility `pnmcrop` to trim away white space;
6. use the `ppmtogif` utility to convert the result to a GIF image.

Note that this does *not* use the `-E` option for `dvips`, but relies on simply removing all white pixels until just text is left. This has the advantage of avoiding the problem of wrong bounding boxes but it has three disadvantages:

1. the PBM utilities are primarily Unix tools, and many people do not have access to them;
2. the cropping process is memory-intensive, slow, and uses a great deal of temporary disk space;
3. the cropping effectively forces everything to the baseline, so that character like em-dash (—) that sits above the baseline is cropped above and below, and the generated GIF image is wrongly placed.

The root of the problem is the use of Ghostscript, which always creates a page-sized bitmap, even if there is only one word on the page. What you want Ghostscript to do is render just the portion of the image inside the bounding box (assuming you use the `-E` flag for `dvips`). You can achieve this by giving Ghostscript a customized page size that is the size of the bounding box. Then you can insert some extra PostScript code to move the image so that it starts at the 0,0 coordinate (adjusting the bounding box accordingly). Ghostscript then displays or converts the image just within the desired area, and no cropping is needed.

The bounding box transformations can be achieved using the `epsffit` program from the `psutils` collection discussed in Section 11.3.1 on p. 435; the page size change is most easily done using a Level 2 PostScript operator `setpagedevice`. Thus a PostScript file which starts:

```
%!PS-Adobe-2.0 EPSF-2.0
%%BoundingBox: 135 528 284 668
...
```

needs to be transformed to something like:

```
%!PS-Adobe-2.0 EPSF-2.0
%%BoundingBox: 0 0 149 140
<< /PageSize [149 140] >> setpagedevice
gsave -135 -528 translate
...
grestore
```

Here we have worked out the width and height of the enclosing rectangle (149 × 140 units), moved the origin down to 0,0 on the page, and set the page size. The `setpagedevice` command is not strictly legal in Encapsulated PostScript, but as long as you use this file only in the controlled environment of Ghostscript, you are safe enough. Figure 11.8 shows a simple Perl script that performs the necessary changes to a PostScript file for Ghostscript to process, without any need for `epsffit`.

```
$bbneeded=1;
$bbpatt="[0-9\\.\\-]";
while (<>) {
    if ( /%%BoundingBox: (\s$bbpatt+)\s($bbpatt+)\s($bbpatt+)\s($bbpatt+)/ )
    {
        if ($bbneeded) {
            $width = $3 - $1;
            $height = $4 - $2;
            $xoffset = 0 - $1;
            $yoffset = 0 - $2;
            print "%%BoundingBox: 0 0 $width $height\n";
            print "<< /PageSize [$width $height] >> setpagedevice\n";
            print "gsave $xoffset $yoffset translate\n";
            $bbneeded=0;
        }
    }
    else { print; }
}
print "grestore\n";
};
```

Figure 11.8: A Perl script to transform an EPS file for Ghostscript.

Now that Ghostscript is rendering only the desired area, you can use its builtin bitmap output facilities. The Unix or DOS command line:

```
gs -dNOPAUSE -q -r100 -sDEVICE=tiffg4 -sOutputFile=foo.tif foo.ps -c quit
```

generates a TIFF fax group 4 image at 100dpi of just the imaged area of the PostScript file `foo.ps` without any further ado. Ghostscript version 4 added anti-aliasing facilities; using the `Netpbm` tools under Unix, you can create a variant GIF image, using the command line:

```
gs -r100 -dNOPAUSE -q -sOutputFile=- \
-sDEVICE=pnm -dTextAlphaBits=4 \
-dGraphicsAlphaBits=4 foo.ps -c quit | \
ppmtogif -interlace \
-transparent \#ffffff > equation.gif
```

Figure 11.9 shows the result of the transformation without anti-aliasing (it does not of course look very good on this typeset page). There is one remaining problem: World Wide Web browsers can usually align images top, middle or bottom but what if an image has some characters with descenders below the base line? Bottom alignment of the images places the bottom of the descenders on the baseline; top alignment is ridiculous, and middle alignment

$$-\Phi_0 \frac{\partial}{\partial \varphi} (\Phi_1 \alpha \sin \varphi) - \Phi_1 \frac{\partial}{\partial \varphi} (\Phi_0 \alpha \sin \varphi) - \mathcal{A}_1 \left[\Phi_0 + \frac{\partial}{\partial \alpha} (\alpha \Phi_0) \right] \sin \varphi = -\alpha \Phi_0 f \sin \varphi$$

Figure 11.9: $\text{\LaTeX} \rightarrow \text{dvi} \rightarrow \text{EPS} \rightarrow \text{GIF}$.

is not quite right either. The answer is to use middle alignment and make $\text{\TeX}$ lie to dvips (and thence down the chain) about the extent of the character; making its depth equal to its height and then middle aligning it in the Web browser has the desired effect.

It is imperative, of course, that Web-making readers do not take these examples as “recipes” without both a precise specification of the desired Web page and an understanding of some basic image-processing techniques. The aim here has simply been to show how relatively trivial and efficient it is to create bitmap output from $\text{\LaTeX}$ and dvips using the free facilities of Ghostscript.

11.7 PSfrag — adding labels to included pictures

The PSfrag package by Michael Grant, Craig Barratt, and David Carlisle solves a common problem in graphics file inclusion. It provides a mechanism whereby $\text{\TeX}$ can be made to typeset labels, equations, and other complex material that override the textual elements in a PostScript graphic file. It works by getting the PostScript interpreter to identify strings in your output file, and then providing replacements.

The PSfrag package defines the command `\psfrag` and it allows you to “tag” a word in the PostScript file at the position where you want $\text{\LaTeX}$ material to be typeset. The replacement for the tag in the $\text{\LaTeX}$ document is defined by the arguments of the command.

```
\psfrag{tag}[posn][psposn][scale][rot]{ $\text{\LaTeX}$  material}
```

The *tag* is replaced by the $\text{\LaTeX}$ material. The first two optional arguments are used to define how the bounding boxes of the $\text{\LaTeX}$ text and the PostScript text line up, using the same syntax as the standard `\makebox` command. The third and fourth optional arguments *scale* and *rot* change the scaling and rotation of the typeset text.

As an example, consider the following PostScript file:

```
%!
%%BoundingBox:100 100 172 136
1 setlinewidth
100 100 moveto
/Times-Roman findfont
12 scalefont setfont
(nabla) show
100 120 moveto
(ALPHA) show
showpage
```

This produces:

ALPHA

nabla

when processed normally; If you put the command:

```
\psfrag{ALPHA}{ $\alpha$  \mathcal{ALPHA}}
\psfrag{nabla}{ $\nabla$ }
```

in your $\text{\LaTeX}$ file that includes the graphic and run it again, you get:

$\alpha \mathcal{ALPHA}$

∇

All `\psfrag` calls that precede an `\includegraphics` command (or equivalent) are used for all subsequently included PostScript files. This permits the definition of global as well as local `\psfrag` substitutions.

This method requires no editing of the PostScript file, as it is all done by the PostScript interpreter;³ it does, however, assume certain things about the way text is presented in the PostScript output from a program. Some packages may treat every letter of a word as a separate string to output, while even in the simple output from programs like dvips, it is not obvious which words come through as complete strings. For instance, while most of the words in the following piece of PostScript are amenable to processing by PSfrag, several are broken in unexpected places:

```
118 w(Y)l(ellowGreen)p SpringGreen 1734 1156 V Black
104 w(SpringGreen)p OliveGreen 46 1255 V Black 85 1305
a(OliveGreen)p RawSienna 384 1255 V Black 127 w(RawSienna)p
Sepia 721 1255 V Black 179 w(Sepia)p Brown 1059 1255
V Black 226 w(Brown)p Tan 1396 1255 V Black 245 w(T)g(an)p
Gray 1734 1255 V Black 260 w(Gray)p Black Black eop
```

It is also possible to get unwanted effects—if you mark the string “A” as a tag to be translated as α , then all distinct strings which the generating program puts out as “A” are translated. On the other hand, if you tag “A” as α and “B” as β , then “AB” is *not* translated.

If the included graphics file is to be resized on inclusion, it is important to understand the distinction between:

```
\resizebox{2in}{!}{\includegraphics{file.ps}}
```

and (using the `graphicx` package):

```
\includegraphics[width=2in]{file.ps}
```

³Earlier versions required a pre-processing stage.

Using the current graphics packages drivers, in the former case PSfrag elements are scaled with the drawing, and in the latter case they are not. Since both sorts of behavior may be needed at different times, this is to be regarded as a feature, not a problem.

PSfrag is principally useful in conjunction with fairly simple drawing programs such as xfig or idraw and plotting packages like MATLAB or gnuplot. In the latter programs, strings can typically be specified for axis labels on graphs and can be set by specifying unique tag identifiers and defining the L^AT_EX replacements. When a large number of plots are being prepared in some analysis program, then PSfrag lets you automate typesetting the text. However, although PSfrag is a clever piece of work that does its job well, it is not ideal for processing large-scale, arbitrarily complicated PostScript. The PostScript file should ideally be designed *with PSfrag in mind*. Note, also, that systematic use of PSfrag requires a good understanding of both the PostScript language and the application generating the figures.

A

Technical appendixes

A.1 emT_EX \special commands

Four \special commands are supported by the emT_EX drivers that have also been implemented by drivers like dvips. The first command includes a PCX or MSP bitmap picture with the syntax:

```
\special{em:graph filename}
```

to which the dvips implementation has added a scaling option. No space is used in the dvi file, so the macro writer must leave the appropriate white space.

The other commands let the user define and link arbitrary points on the page with straight lines:

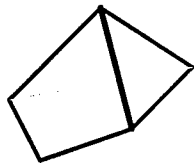
```
\special{em:point n}  
\special{em:line xmodifier, ymodifier, width}  
\special{em:moveto}  
\special{em:lineto}  
\special{em:linewidth width}
```

A numbered point defined with the `point` gets assigned the coordinates of the current point (which can be set, for instance, in the `picture` environment). The `line` command then links any two points with a line of *width* thickness (this parameter is optional). The default line width can be set with `linewidth`. Line thickness is expressed in normal T_EX units. The

modifier in the `line` command can be one of `h`, `v` or `p`; these determine whether two lines join with a horizontal, vertical or perpendicular cut.

Instead of defining numbered points, you can move to a new point with `moveto` and draw a line from the previous point with `lineto`.

These simple `\specials` have the great advantage that they can be used in the standard picture environment without worrying about line thickness or angle. As shown in the example below, the standard color package commands have their expected effect on the lines:



```
\setlength{\unitlength}{5mm}
\begin{picture}(10,10)
\put(2,2){\special{em:point 1}}
\put(5,3){\special{em:point 2}}
\put(7,5){\special{em:point 3}}
\put(4,7){\special{em:point 4}}
\put(1,4){\special{em:point 5}}
\special{em:linewidth 2pt}
\color{blue}\special{em:line 1,2}\special{em:line 2,3}}
\special{em:line 3,4} \special{em:line 4,5}
\special{em:line 5,1}
\color{yellow}\special{em:line 3,5,0.5pt}
\color{red}\special{em:line 4,2,3pt}
\end{picture}
```

A-1-1

A.2 `tpic` \special conventions

As various examples in Chapter 1 show, the `tpic` conventions define a powerful set of drawing commands. The `tpic` \special commands are all two-letter keywords followed by zero or more arguments. All distance arguments are integers excepts where specifically noted below. Angles are specified in radians. Unfortunately, in the coordinate system of the `dvi` language, the y -axis starts at the top of the page and works down, and you should bear this in mind when writing macro packages with the `tpic` \special commands.

Table A.1 summarizes the graphic primitives required by a `tpic` conformant `dvi`-processor; we now give details on the necessary parameters.

pn s Set the pen size to s , in thousands of an inch.

pa x y Add point (x, y) to a "path" which is being stored in the `dvi` interpreter. The values of x and y are in thousands of an inch, and their position is relative to the position of the *next* drawing \special (such as an `fp`). A set of coordinates is built in memory, but output to the page only when a drawing command is given, and only at that time are the absolute coordinates calculated.

fp Print a previously defined path using the current pen size. The number of path elements

Table A.1: Summary of `tpic` \special requirements.

\special	name	effect
pn		set the line width
pa		add an x, y position to the path
fp	(ip)	print current path as succession of straight lines
da		as above, but using a dashed line
dt		as above, but using a dotted line
sp		print current path as a spline curve
ar	(ia)	given a start and end angle, draw an arc or circle
sh		shade previously defined closed object

is reset to zero. If shading has been specified and if the path is closed, the enclosed area is shaded.

ip This is the same as `fp` except that the path is not actually drawn; shading of the defined region is performed if specified.

da f This is the same as `fp` except that a dashed line is drawn with dashes of length f inches (f is a real number).

dt f This is the same as `fp` except that a dotted line is drawn, with f inches between each dot (f is a real number).

sp d This is the same as `fp` except that a spline is drawn through the points. The spline is only guaranteed to go through the first and last points, its precise route depending on the relative positions of any intermediate points. The argument d is an optional real value. If $d = 0$ or the argument is omitted, the spline is drawn normally, as a solid line. If $d > 0$, the line is dashed with a dash width of d ; if $d < 0$, the line is dotted with a dot width of $-d$.

ar x y r_x r_y s e Draw an arc with center at (x, y) from starting angle s to ending angle e . If the arc is a complete circle or ellipse, then r_x and r_y determine the x and y radii, respectively. Otherwise, an arc from s to e is drawn, and x and y are identical (whatever the value of r_x is).

ia x y r_x r_y s e This is the same as `ar`, but the path is not actually drawn. Shading of the defined region is performed if specified.

sh s Shade the interior of the next closed object; this requires that at least three `pa` commands be given followed by an `fp` or `ip` command or by an `ar` or `ia` command. The value s is a real number from 0 to 1. Zero implies a completely white interior (including erasing anything underneath), while 1 implies completely black. A value of 0.5 is a default "normal" gray shading; if s is not specified a value of 0.5 is used. Shading applies to the interior of the object only; the borders of the object are still drawn with

Table A.2: Command-line options for bm2font.

option	meaning	default
-f name	name of picture in TeX	
-h number	horizontal resolution in pixels per inch	300
-v number	vertical resolution	300
-l number	length of mapline in bytes (pure bitmaps only)	
-py-or-n	height of generated characters 1 inch?	y
-ey-or-n	stretch EGA pictures?	y
-iy-or-n	inversion of pixels?	n
-gy-or-n	gray pixels in bitmap?	n
-wy-or-n	let white be light gray?	y
-dy-or-n	distribute errors?	y
-sy-or-n	separate gray dots?	n
-ry-or-n	repeat each gray pixel?	n
-unumber	pixels for gray rectangle	8
-cnumber	vertical pixels for rectangle	8
-xnumber	bits per sample	$0 < x < 9$
-bnumber	reduce halftone colors	0
-tnumber	gradation value in%	70
-znumber	area of gradation in %	70
-mnumber	width of picture on paper	in mm
-nnumber	height of picture on paper	in mm
-jy-or-n	clip white space?	y
-kcharacter	color: c, m, y, or k	k

the current pen (this is particularly important for dvi drivers which do not support the sh command.) The pixels generated by the shading value are ANDed with the pixels already in the area, and subsequent text pixels are ORed. Thus, text followed by a “sh 0” command covering the same area disappears. Any shading value greater than zero adds the shading to the existing text, but leaves the text visible (of course, a shading value of 1 is completely black, so that text is not distinguishable.)

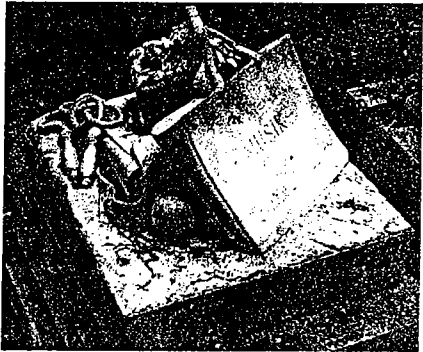
Note: the sh command is optional in dvi-driver implementations of tpic; drivers that do not implement it should silently ignore it.

The following three commands are considered obsolete but probably should be implemented for backwards compatibility:

- wh This is the same as “sh 0”.
- bk This is the same as “sh 1”.
- tx This command specified a texture to be used in shading instead of the default one. Only used by one existing device driver; new drivers should be implemented to ignore it.



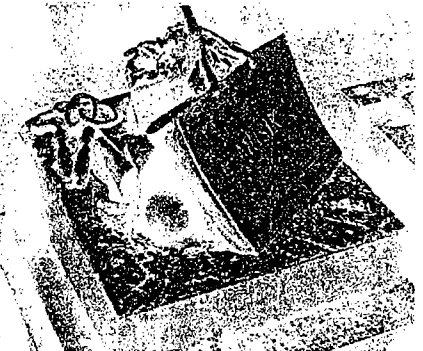
(a) no options



(b) no options



(c) options -b4 -gy



(d) option -iy

Figure A.1: Examples of using bm2font.

A.3 The bm2font program

The command-line options for bm2font are listed in Table A.2 on the facing page. The result of three of these command-line switches (“reduction of halftone colors”, “gray pixels in bitmap”, and “pixel-inversion”) are demonstrated in Fig. A.1. A significant limitation of the program is that the output files become impossibly large, and use too many TeX fonts, if you try to work at typesetter resolutions (above 600 dpi). The examples here were generated at 600 dpi, a quarter of the resolution used in the rest of the book.

A.4 The dvips color separation header file

This is the contents of the file `colorsep.pro`, distributed with dvips, which redefines PostScript color operators to generate CMYK separations. It starts by defining some useful commands:

```
1  %!
2  /seppages 0 def
3  userdict begin
4  /Min {% 3 items on stack, find the smallest
5  2 copy lt { pop }{ exch pop } ifelse
6  2 copy lt { pop }{ exch pop } ifelse
7  } def
```

We need to convert color specified in the gray, HSB or RGB modes to CMYK for the four separations. Gray is straightforward; HSB is simply a variant way of expressing RGB:

```
8  /SetGray {
9  1 exch sub systemdict begin adjustdot setgray end
10 } def
11 /sethsbcolor {systemdict begin
12   sethsbcolor currentrgbcolor end
13   userdict begin setrgbcolor end}def
```

RGB itself is harder. Adobe Systems (1990), p. 305, says: to convert RGB to CMYK, we subtract the amount of each color from 1, and find the minimum black:

```
14 /ToCMYK
15 {
16   3 { 1 exch sub 3 1 roll } repeat
17   3 copy Min
18   blackUCR sub
19   dup 0 lt {pop 0} if
20   /percent_UCR exch def
```

Now we subtract that black undercolor from each color and work out the black itself:

```
21   3 { percent_UCR sub 3 1 roll } repeat
22   percent_UCR 1.25 mul % 1 exch sub
23   } def
```

The PostScript stack should now have the C, M, Y, and K values.

The CM command prints cropmarks:

```
24 /cX 18 def
25 /CM{gsave TR 0 cX neg moveto 0 cX lineto stroke
26   cX neg 0 moveto cX 0 lineto stroke grestore}def
```

The critical part is the definition of `bop-hook`, which is executed at the start of each page; this goes around a cycle of four pages, doing a different separation on each. Below we assume that each page is duplicated four times using the `-b 4` option of dvips:

```
27 /bop-hook{cX dup TR
28   seppages 1 add
29   /seppages exch def
30   seppages 5 eq { /seppages 1 def } if
31   seppages 1 eq {
32     /ColorName (CYAN) def
33     CYAN setupcolor
34     /WhichColor 3 def } if
35   seppages 2 eq {
36     /ColorName (MAGENTA) def
37     MAGENTA setupcolor
38     /WhichColor 2 def } if
39   seppages 3 eq {
40     /ColorName (YELLOW) def
41     YELLOW setupcolor
42     /WhichColor 1 def } if
43   seppages 4 eq {
44     /ColorName (black) def
45     black setupcolor
46     /WhichColor 0 def } if
```

Set crop marks on each page with the separation name printed:

```
47 gsave .3 setlinewidth
48 3 -7 moveto
49 /Helvetica findfont 6 scalefont setfont
50 ColorName show
51 0 0 CM
52 vsize cX 2 mul sub dup hsize cX 2 mul sub dup isls{4 2 roll}if 0 CM
53 exch CM 0
54 exch CM
55 grestore 0 cX -2 mul TR isls
56 {cX -2 mul 0 TR}if
57   } def end
58 /separations 48 dict def
59 separations begin
60   /cmykprocs [
61     { pop pop pop SetGray } % cyan
62     { pop pop exch pop SetGray } % magenta
63     { pop 3 1 roll pop pop SetGray } % yellow
64     { 4 1 roll pop pop pop SetGray } % black
65   ] def
66   /rgbprocs [ %def
```

```

67 { ToCMYK pop pop pop SetGray }
68 { ToCMYK pop pop exch pop SetGray }
69 { ToCMYK pop 3 1 roll pop pop SetGray }
70 { ToCMYK 4 1 roll pop pop pop SetGray }
71 ] def
72 /screenangles [ %def
73   105 % cyan
74   75 % magenta
75   0 % yellow
76   45 % black
77 ] def
78 end

```

The command `setupcolor` takes 0, 1, 2, or 3 as its argument, for cyan, magenta, yellow, and black, respectively:

```

79 /CYAN 0 def      /MAGENTA 1 def
80 /YELLOW 2 def    /black 3 def
81 /setupcolor{ %def
82   userdict begin
83     dup separations /cmypprocs get exch get
84     /setcmypcolor exch def
85     dup separations /rgbprocs get exch get
86     /setrgbcolor exch def
87     separations /screenangles get exch get
88     currentscreen
89     exch pop 3 -1 roll exch
90     setscreen
91     /setscreen { pop pop pop } def

```

We redefine `setgray` so that it shows only on the black separation:

```

92 /setgray {
93   WhichColor 0 eq
94   {systemdict begin adjustdot setgray end}
95   {pop systemdict begin 1 setgray end}
96   ifelse}def
97 end
98 } bind def
99 /adjustdot { dup 0 eq { } { dup 1 exch sub .1 mul add} ifelse } def

```

Finally, we set the percentages of undercolor removal:

```

100 /magentaUCR .3 def
101 /yellowUCR .07 def
102 /blackUCR .4 def

```

A.5 Catalogue of typefaces with *Fontname* abbreviations

This catalogue lists *Fontname* family abbreviations for a large number of fonts; it also gives a suggested 8-letter directory name in brackets. Where the same font has different names between suppliers, the variants are listed preceded by the supplier abbreviation (see Table 10.4 on p. 383) and a colon.

a1 (arial) Arial	au (auriol) Auriol	bp (bodocamp) BodoniCampanile	cn (century) Century
a2 (amasis) Amasis	b:Freeform721	b:Modern735	co (cochin) Cochin
a3 (adlib) AdLib	av (avenir) Avenir	br (berling) Berling	cp (copperpl) Copperplate
a4 (algorith2) AlternateGothicNo2	az (azdztrot) AkzidenzGrotesk	b:Revival565BT	cq (cheq) Cheq
a5 (allegro) Allegro	b:Gothic725BT	bs (bakersig) BakerSignet	cr (courier) Courier
a6 (amelia) Amelia	b0 (blackoak) Blackoak	bt (boton) Boton	m:MCourierPS
a7 (amertext) AmericanText	b1 (blado) Blado	bu (brush) Brush	m:CourierTwelveMT
a8 (aurora) Aurora	b2 (binnyos) BinnyOldStyle	bv (baskervi) Baskerville	u:NumbusMonL
a9 (adsans) Adsans	b3 (braggado) Braggadocio	bw (broadway) Broadway	cs (cntursbk) CenturySchoolbook
b:Humanist970BT	b4 (bison) Bison	by (berkeley) Berkeley	ct (cheltenh) Cheltenham
aa (aachen) Aachen	b:Brush738BT	c0 (carolina) Carolina	cu (centuros) CenturyOldStyle
ab (aboeckli) ArnoldBoecklin	b5 (binghams) BinghamScript	c1 (calisto) Calisto	cv (clairvx) Clairvaux
ac (acaslon) ACaslon	b:Freehand591BT	c2 (casln224) CaslonTwoTwentyFour	cw (cottonwd) Cottonwood
ad (agaramon) AGaramond	b6 (block) Block	c3 (caslon3) CaslonThree	cx (cavton) Caxton
ag (avantgar) AvantGarde	b:Gothic821	c4 (chicago) Chicago	cy (city) City
m:MAvantGardeGothicPS	b7 (bernhard) Bernhard	c5 (casln540) CaslonFiveForty	b:SquareSlabserif711BT
u:URWGothicL	BernhardModern	c6 (clarion) Clarion	cz (choc) Choc
ah (ashleysc) AshleyScript	b8 (blippo) Blippo	c7 (compacta) Compacta	b:Staccato555BT
ai (abadi) Abadi	b9 (bremen) Bremen	c8 (calvert) Calvert	d0 (davida) DavidaBold
al (albertus) Albertus	bb (bembo) Bembo	c9 (codex) Codex	dc (domcasl) Dom
b:Flareserif821BT	b:Aldine401BT	b:Calligraphic421BT	DomCasual DomDiagonal
am (amercana) Americana	bc (banco) Banco	ca (caslon) Caslon	dd (ducberry) DucDeBerry
b:Americana	bd (bodoni) Bodoni	cb (cooper) Cooper	de (dellarob) DellaRobbia
an (anna) Anna	be (bell) Bell	cc (cascade) Cascade	b:Calligraphic810BT
ao (atqolive) AntiqueOlive	bf (biffo) Biffo	b:Freehand471BT	d1 (dolores) Dolores
b:Incised901BT	bg (benguiat) Benguiat	cd (clarendo) Clarendon	dm (delima) Delima
ap (apollo) Apollo	bh (bauhaus) Bauhaus	ce (centenni) Centennial	dr (doric) Doric
aq (antiqua) Antiqua	bi (birch) Birch	cf (clearfac) Clearface	ds (dorchesc) DorchesterScript
u:URWAntiqua	bk (bookman) Bookman	cg (charlema) Charlemagne	dt (dante) Dante
m:BookAntiqua	m:MBookmanPS	ch (charter) Charter	e0 (embassy) Embassy
ar (arcadia) Arcadia	u:URWBookmanL	ci (candida) Candida	e1 (empire) Empire
as (aldus) Aldus	b1 (belwe) Belwe	c1 (cloister) Cloister	ea (lectra) Electra
at (atypewri) AmericanTypewriter	bm (bulmer) Bulmer	cm (cm) Computer Modern	b:Transitional521BT
	bn (bernard) Bernard		
	bo (balloon) Balloon		

eb (baskerbe) BaskervilleBE	ft (fettefrk) FetteFraktur Fraktur	hd (headline) Headline	it (italia) Italia	m3 (mirarae) Miraraeae	ni (nimrod) Nimrod	ot (coronet) Coronet b:Ribbon131BT	pp (laudatio) Laudatio
ec (applecha)	fu (futura) Futura	hg (hiroshig) Hiroshige	iw (ironwood) Ironwood	m4 (mrearl) MisterEarl	nk (neograph) Neographik	ov (octavian) Octavian	pw (perrywoo) Perrywood
e (apple) Chancery	fx (felix) FelixTitling	hi (hvinsera)	iy (iowanold)	m5 (murrayhl) MurrayHill	n1 (neuland) Neular.d	ox (onyx) Onyx	px (popplpfx)
ee (egyptian) Egyptian	fy (flyer) Flyer	h1 (hvinsera)	lowanOldStyle	m6 (mittlsft) Mittelschrift	b:Informal011BT	oy (olympian) Olympian	py (playbill) Playbill
egyptianExtended	g0 (gando) Gando	hm (charme) Charme	jf (jeffersn) Jefferson	m7 (matura) Matura	nm (nimbus) Nimbus	i:OlympicLH	pz (poliphil) Poliphilus
ef (egyptien) EgyptienneF	g1 (gothic13)	hn (hvneue) HelveticaNeue	jn (janson) Janson	m8 (monotyos)	nn (nclarend)	oz (ozhandcr)	qr (quorum) Quorum
eg (stgaramo)	g1 (gothic13)	ho (horleyos)	jo (joanna) Joanna	MonotypeOldStyle	np (nsplantin) NewsPlantin	OzHandicraft	qt (boulevard) Boulevard
StempelGaramond	g2 (giddyup) Giddyup	hr (hvround)	jp (juniper) Juniper	ma (mandate) Mandate	nr (nberolin) NewBerolina	p0 (poplar) Poplar	r0 (romana) Romana
b:OriginalGaramondBT	g3 (garamon3)	hs (hseati) Hanseatic	kb (kabel) Kabel	b:Freehand521BT	ns (timesnps)	p1 (palacesc) PalaceScript	r1 (rotis) RotisSemi
eh (engsf) Engschrift	g3 (garamon3)	ht (hoeffert) HoeferText	kg (bankgoth) BankGothic	mc (monaco) e:Monaco	n (timesnewromanPS)	p2 (pepita) Pepita	r2 (cantoria) Cantoria
ei (esprit) Esprit	b:AmecianGaramondBT	hv (helvetic) Helvetica	k1 (klang) Klang	md (meridien) Meridien	nt (timesnew)	p3 (palette) Palette	ra (raleigh) Raleigh
ek (eckmann) Eckmann	g4 (gymodern)	u:NimbusSans	kn (kino) Kino	me (mercursc)	TimesNewRoman	b:Brush445BT	ro (rotis) Rotis
b:Freeform710	g5 (gycatalog)	u:NimbusSanL	ko (bookmano)	MercuriusScript	m:TimesNR	p4 (pioneer) PioneerITC	rm (carmina) Carmina
e1 (ellingtn) Ellington	g6 (gorilla) GorillaITC	b:Swiss721BT	kr (korinna) Korinna	mg (marigold) Marigold	n:SonoranSerif	p5 (piranesi) Piranesi	rp (reporter) Reporter
en (engraver) Engravers	g9 (ugaramon)	b:Monospace821BT	ks (kuenstsc)	mh (machine) Machine	nu (nuptial) Nuptial	p6 (blueprin) Blueprint	rq (russelsq) RussellSquare
eo (amerigo) Amerigo	garamondRetrospective	b:Swiss911BT d:Sans	ku (kaufmann) Kaufmann	mi (medici) Medici	nz (neuzeits) Neuzeits	pa (parkave) ParkAvenue	rs (arrus) Arrus
b:AmerigoBT	gb (gothicbb) GothicBBB	Newton	11 (liberty) Liberty	m1 (melior) Melior	b:Geometric706BT	pb (ptbarnum) PtBarnum	ru (runic) Runic
er (eras) Eras	ge (geneva) e:Geneva	n:SonoranSansSerif	12 (libra) Libra	b:ZapfElliptical711BT	mn (minion) Minion	pc (poetica)	rv (revue) Revue
es (eurostil) Eurostile	gg (garthgra) GarthGraphic	hw (shadow) Shadow	13 (lucia) Lucia	mo (modern) Modern	mp (memphis) Memphis	PoeticaChancery	rw (rockwell) RockwellSlate
b:Square721BT k:Minima	gj (granjon) Granjon	hx (huxleyvr)	14 (lucian) Lucian	b:Modern20BT	b:GeometricSlab703BT	s:Scriptoria	b:GeometricSlab712BT
et (ehrhadt) Ehrhardt	b:ElegantGaramondBT	i0 (commscr)	1b (lubaling) LubalinGraph	mq (mesquite) Mesquite	mr (madrone) Madrone	o1 (oranda) Oranda	ry (ryumin) Ryumin
eu (euler) Euler	g1 (galliard) Galliard	i1 (impress) Impress	1c (lucida) Lucida	ms (mistrall) Mistrall	ms (madrone) Madrone	o2 (orbib) OrbitB	s0 (sanmarco) SanMarco
ex (excelsior) Excelsior	gm (garamond) Garamond	i2 (impuls) Impuls	1f (life) Life	b:Staccato222BT	mt (minister) Minister	o3 (colonna) Colonna	s1 (scotchro) ScotchRoman
f0 (flemishs) FlemishScript	go (gyoldsty)	ia (insignia) Insignia	1g (lettrgth) LetterGothic	mv (malvern) Malvern	na (newaster) NewAster	o4 (oceansan) OceanSans	s2 (script) Script
f1 (fournier) Fournier	gp (glypha) Glypha	ib (ibycus) Ibycus	ISILGothic	mz (amazone) Amazone	nb (nbaskerv)	o7 (oldsty7) OldStyleSeven	s3 (seagull) Seagull
f2 (facade) Facade	gr (gloucest) Gloucester	ic (impact) Impact	1h (lubright) LucidaBright	n0 (normande) Normande	nc (nbiner) Binner	oa (ocra) OCR-A	s4 (shotgun) Shotgun
f3 (festival) Festival	gs (gillsans) GillSans	if (inflex) Inflex	1i (latin) Latin	n1 (biner) Binner	na (newaster) NewAster	ob (ocrb) OCR-B	ShotgunBlanks
f4 (footligh) Footlight	m:GillAltOneMT	ig (scgaramo)	1n (lino) Lino	na (newaster) NewAster	nb (nbaskerv)	oc (concorde) Concorde	s5 (snell) Snell
f5 (french) French	b:Humanist521BT	SimonciniGaramond	1o (lithos) Lithos	nb (nbaskerv)	nc (nbiner) Binner	oe (oldengli) OldEnglish	s6 (stuyvesa) Stuyvesant
fa (formata) Formata	gt (goudytxt) GoudyText	b:ItalianGaramond	1s (lucsans) LucidaSans	ng (newgth) NewsGothic	nc (nbiner) Binner	b:English157	s7 (sassoon) Sassoon
fe (forte) Forte	gu (guardi) Guardi	i1 (imprint) Imprint	1t (lutetia) Lutetia	ni (nimrod) Nimrod	na (newaster) NewAster	oi (ondine) Ondine	sb (sabon) Sabon
ff (fairfld) Fairfield	gv (giovanni) Giovanni	il (caecilia) Caecilia	1w (leawood) Leawood	nk (neograph) Neographik	nb (nbaskerv)	Mermaid	b:ClassicalGaramondBT
b:Transitional551BT	gy (goudy) Goudy	im (imago) Imago	1x (lucidfax) LucidaFax	n1 (biner) Binner	nc (nbiner) Binner	b:FormalScript421BT	sc (slimbach) Slimbach
fg (frankgth)	h0 (handelgo)	in (industri) Industria	1y (lydian) Lydian	na (newaster) NewAster	nc (nbiner) Binner	o1 (colmcill) Colmcille	sf (serifa) Serifa
FranklinGothic	hb (hobo) Hobo Tramp	io (ionic) Ionic	m0 (monospac) Monospace	nb (nbaskerv)	nc (nbiner) Binner	om (omnia) Omnia	sg (serifgth) SerifGothic
fi (figaro) Figaro	hc (herculnm) Herculanum	b:News701BT	m1 (monolisc)	ng (newgth) NewsGothic	nc (nbiner) Binner	on (corona) Corona	sh (shannon) Shannon
f1 (flora) Flora		ip (impressm) Impressum	MonolineScript		nc (nbiner) Binner	op (optima) Optima	sk (skia) e:Skia
fn (fenice) Fenice		is (italnos) ItalianOldStyle			nd (ncaledon)	d:Optimum	s1 (stencil) Stencil
fo (folio) Folio					nf (nofret) Nofret	b:ZapfHumanist601BT	sm (spectrum) Spectrum
fq (frizquad) FrizQuadrata					ng (newgth) NewsGothic	Omega	sn (spartan) Spartan
fr (frutiger) Frutiger						or (orator) Orator	
b:Humanist777BT						os (oldstyle) OldStyle	
fs (falstaff) Falstaff							

so (sonata) Sonata	tj (trajan) Trajan	um (umbra) Umbra	wi (wingding) Wingdings
sp (serpenti) Serpentine	tk (tekton) Tekton	un (univers) Univers	wk (wilke) Wilke
sz (snellrnd)	tl (castella) Castellar	b:ZurichBT	wn (windsor) Windsor
SnellRoundhand	tm (times) Times	ur (centaur) Centaur	wr (wittenbe)
ss (stschnei)	u:NimbusRoman	b:Venetian301BT	WitternbergerFrakt
StempelSchneidler	u:NimbusRomNo9L	ut (utopia) Utopia	ws (weiss) Weiss
st (stone) Stone	b:Dutch801BT	uw (usherwoo) Usherwood	ww (willow) Willow
sv (souvenir) Souvenir	to (torino) Torino	uy (universi) University	yg (centuryg)
sw (swing) Swing	b:Industrial736BT	v0 (vineta) Vineta	CenturyGothic
sx (syntax) Syntax	tp (tempo) Tempo	va (activa) Activa	yh (strayhor) Strayhorn
sy (symbol) Symbol	tv (trumpmed)	vc (vectora) Vectora	ym (stymie) Stymie
m:MSymbolPS	TrumpMediaeval	vd (vandijck) VanDijck	za (zantiqua) ZAntiqua
u:StandardSymL	b:Kuenstler480BT	vi (victoria) Victoria	zc (zapfchan)
t0 (tango) Tango	tw (twentyc)	vj (veljovic) Veljovic	ZapfChancery
t1 (thunderb) Thunderbird	TwentiethCentury	vl (versail) Versailles	m:ZapfChanceryPS
t2 (typo) TypoUpright	ty (typewrit) Typewriter	vr (vagrnd) VAGRounded	u:URWChanceryL
tb (bebodoni) BBodoni	u0 (columbus) Columbus	w0 (weddingt) WeddingT	zd (zapfding) ZapfDingbats
te (timeseur) TimesEuropa	ub (babodoni) BauerBodoni	wb (walbaum) Walbaum	m:ZapfDingbatsPS
tf (tiffany) Tiffany	ug (benguth)	wd (weideman) Weidemann	u:Dingbats
tg (tradegth) TradeGothic	BenguiatGothic	wg (wlklgh)	zg (neuzeitg) NeuzeitGrotesk
ti (techphon) TechPhonetic	uh (cushing) Cushing	WilhelmKlingsporGotisch	zt (zeitgeis) Zeitgeist
	ul (universa) Universal		

A.6 Font encoding tables

Table A.3: Font encoding table, by name.

Name	Glyph	EC	8r	ISO Latin 1	Adobe Standard	Windows ANSI	Mac	PDF
A	À	65	65	65	65	65	65	65
AE	Æ	198	198	198	225	198	174	198
Aacute	Á	193	193	193		193	231	193
Abreve	Â	128						
Acircumflex	Â	194	194	194		194	229	194
Adieresis	Ä	196	196	196		196	128	196
Agrave	À	192	192	192		192	203	192
Aogonek	Ą	129						
Aring	Å	197	197	197		197	129	197

Name	Glyph	EC	8r	ISO Latin 1	Adobe Standard	Windows ANSI	Mac	PDF
Atilde	Ã	195	195	195		195	204	195
B	B	66	66	66	66	66	66	66
C	C	67	67	67	67	67	67	67
Cacute	Ć	130						
Ccaron	Č	131						
Ccedilla	Ç	199	199	199		199	130	199
D	D	68	68	68	68	68	68	68
Dcaron	Ď	132						
E	E	69	69	69	69	69	69	69
Eacute	É	201	201	201		201	131	201
Ecaron	Ě	133						
Ecircumflex	Ê	202	202	202		202	230	202
Edieresis	Ë	203	203	203		203	232	203
Egrave	È	200	200	200		200	233	200
Eng	Ǝ	141						
Eogonek	Ę	134						
Eth	Ð	208	208	208		208		208
F	F	70	70	70	70	70	70	70
G	G	71	71	71	71	71	71	71
Gbreve	Ğ	135						
Germandbls	Š	223						
H	H	72	72	72	72	72	72	72
I	I	73	73	73	73	73	73	73
Ij	İ	156						
Iacute	Í	205	205	205		205	234	205
Icircumflex	Î	206	206	206		206	235	206
Idieresis	Ï	207	207	207		207	236	207
Idotaccent	İ	157						
Igrave	Î	204	204	204		204	237	204
J	J	74	74	74	74	74	74	74
K	K	75	75	75	75	75	75	75
L	L	76	76	76	76	76	76	76
Lacute	Ł	136						
Lquoteright	Ł	137						
Lslash	Ł	138	6		232			149
M	M	77	77	77	77	77	77	77
N	N	78	78	78	78	78	78	78
Nacute	Ń	139						
Ncaron	Ň	140						

Name	Glyph	EC	8r	ISO Latin 1	Adobe Standard	Windows ANSI	Mac	PDF
Ntilde	Ñ	209	209	209		209	132	209
O	Ó	79	79	79	79	79	79	79
OE	Œ	215	140		234	140	206	150
Oacute	Ô	211	211	211		211	238	211
Ocircumflex	Ö	212	212	212		212	239	212
Odieresis	Û	214	214	214		214	133	214
Ograve	Ò	210	210	210		210	241	210
Ohungarumlaut	Ő	142						
Oslash	Ø	216	216	216	233	216	175	216
Otilde	Õ	213	213	213		213	205	213
P	P	80	80	80	80	80	80	80
Q	Q	81	81	81	81	81	81	81
R	R	82	82	82	82	82	82	82
Racute	Ŕ	143						
Rcaron	Ř	144						
S	S	83	83	83	83	83	83	83
Sacute	Ś	145						
Scaron	Š	146	138			138		151
Scedilla	Ş	147						
T	T	84	84	84	84	84	84	84
Tcaron	Ť	148						
Tcedilla	Ț	149						
Thorn	Þ	222	222	222		222		222
U	U	85	85	85	85	85	85	85
Uacute	Ú	218	218	218		218	242	218
Ucircumflex	Û	219	219	219		219	243	219
Udieresis	Ü	220	220	220		220	134	220
Ugrave	Ù	217	217	217		217	244	217
Uhungarumlaut	Ű	150						
Uring	Û	151						
V	V	86	86	86	86	86	86	86
W	W	87	87	87	87	87	87	87
X	X	88	88	88	88	88	88	88
Y	Y	89	89	89	89	89	89	89
Yacute	Ý	221	221	221		221		221
Ydieresis	ÿ	152	159			159	217	152
Z	Z	90	90	90	90	90	90	90
Zacute	Ź	153						
Zcaron	Ž	154	14					153

Name	Glyph	EC	8r	ISO Latin 1	Adobe Standard	Windows ANSI	Mac	PDF
Zdotaccent	Ž	155						
a	a	97	97	97	97	97	97	97
acute	á	225	225	225		225	135	225
abreve	ă	160						
acircumflex	â	226	226	226		226	137	226
acute	ı	1	180	180	194	180	171	180
adieresis	ä	228	228	228		228	138	228
ae	æ	230	230	230	241	230	190	230
agrave	à	224	224	224		224	136	224
ampersand	&	38	38	38	38	38	38	38
aogonek	ą	161						
aring	å	229	229	229		229	140	229
asciicircum	^	94	94	94	94	94	94	94
asciitilde	~	126	126	126	126	126	126	126
asterisk	*	42	42	42	42	42	42	42
at	@	64	64	64	64	64	64	64
atilde	ã	227	227	227		227	139	227
b	b	98	98	98	98	98	98	98
backslash	\	92	92	92	92	92	92	92
bar		124	124	124	124	124	124	124
blank							202	
braceleft	{	123	123	123	123	123	123	123
braceright	}	125	125	125	125	125	125	125
bracketleft	[	91	91	91	91	91	91	91
bracketright	]	93	93	93	93	93	93	93
breve	˘	8	11	150	198	24	249	24
brokenbar		166	166			166		166
bullet	•	149			183		165	
c	c	99	99	99	99	99	99	99
cacute	ć	162						
caron	ˇ	7	16	159	207	25	255	25
ccaron	č	163						
ccedilla	ç	231	231	231		231	141	231
cedilla	¸	11	184	184	203	184	252	184
cent	¢	162	162	162	162	162	162	162
circumflex	ˆ	2	136	147	195	136	246	26
colon	:	58	58	58	58	58	58	58
comma	,	44	44	44	44	44	44	44
compoundwordmark		23						
copyright	©	169	169			169	169	169

Name	Glyph	EC	8r	ISO Latin 1	Adobe Standard	Windows ANSI	Mac	PDF
currency	¤		164	164	168	164	219	164
d	d	100	100	100	100	100	100	100
dagger	†		134		178	134	160	129
daggerdbl	‡		135		179	135	224	130
dbar	ḍ	158						
degree	°		176	176		176	161	176
dieresis	¨	4	168	168	200	168	172	168
divide	÷		247	247		247	214	247
dollar	\$	36	36	36	36	36	36	36
dotaccent	·	10	1	151	199	27	250	27
dotlessi	ı	25	17	144	245		245	154
dotlessj	ȷ	26	18					
dquoteright	”	164						
e	e	101	101	101	101	101	101	101
acute	é	233	233	233		233	142	233
ecaron	š	165						
ecircumflex	ê	234	234	234		234	144	234
edieresis	ë	235	235	235		235	145	235
egrave	è	232	232	232		232	143	232
eight	8	56	56	56	56	56	56	56
ellipsis	...		133		188	133	201	131
emdash	—	22	151		208	151	209	132
endash	–	21	150		177	150	208	133
eng	ŋ	173						
eogonek	ę	166						
equal	=	61	61	61	61	61	61	61
eth	ð	240	240	240		240		240
exclam	!	33	33	33	33	33	33	33
exclamdown	¡	189	161	161	161	161	193	161
f	f	102	102	102	102	102	102	102
ff	ff	27	19					
ffi	ffi	30	20					
ffl	ffl	31	21					
fi	fi	28	2		174		222	147
five	5	53	53	53	53	53	53	53
fl	fl	29	3		175		223	148
florin	₣		131		166	131	196	134
four	4	52	52	52	52	52	52	52
fraction	/		4		164		218	135
g	g	103	103	103	103	103	103	103

Name	Glyph	EC	8r	ISO Latin 1	Adobe Standard	Windows ANSI	Mac	PDF
gbreve	ḡ	167						
germandbls	ſc	255	223	223	251	223	167	223
grave	`	0	30	145	193		96	96
greater	>	62	62	62	62	62	62	62
guillemotleft	«	19	171	171	171	171	199	171
guillemotright	»	20	187	187	187	187	200	187
guilsingleft	<	14	139		172	139	220	136
guilsingright	>	15	155		173	155	221	137
hungarumlaut	˝	104	104	104	104	104	104	104
hyphen	-	5	5	157	205	28	253	28
i	i	127	173		45	173	45	45
iacute	í	105	105	105	105	105	105	105
icircumflex	î	237	237	237		237	146	237
idieresis	ï	238	238	238		238	148	238
igrave	ì	239	239	239		239	149	239
ij	ij	236	236	236		236	147	236
j	j	188						
k	k	106	106	106	106	106	106	106
l	l	107	107	107	107	107	107	107
lacute	ĺ	108	108	108	108	108	108	108
less	<	168						
logicalnot	¬	60	60	60	60	60	60	60
lquoteright	”	172	172			172	194	172
lslash	ł	169						
m	m	170	7		248			155
macron	ˉ	109	109	109	109	109	109	109
minus	-	9	175	175	197	175	248	175
mu	μ		12	45				138
multiply	×		181	181		181	181	181
n	n		215	215		215		215
nacute	ń	110	110	110	110	110	110	110
nbspace		171						
ncaron	ň			160				
nine	9	172						
ntilde	ñ	57	57	57	57	57	57	57
numbersign	#	241	241	241		241	150	241
o	o	35	35	35	35	35	35	35
oacute	ó	111	111	111	111	111	111	111
ocircumflex	ô	243	243	243		243	151	243
		244	244	244		244	153	244

Name	Glyph	EC	8r	ISO Latin 1	Adobe Standard	Windows ANSI	Mac	PDF
odieresis	ö	246	246	246		246	154	246
oe	œ	247	156		250	156	207	156
ogonek	˛	12	8	158	206	29	254	29
ograve	ò	242	242	242		242	152	242
ohungarumlaut	ő	174						
one	1	49	49	49	49	49	49	49
onehalf	½		189	189		189		189
onequarter	¼		188	188		188		188
onesuperior	¹		185	185		185		185
ordfeminine	ª		170	170	227	170	187	170
ordmasculine	º		186	186	235	186	188	186
oslash	ø	248	248	248	249	248	191	248
otilde	õ	245	245	245		245	155	245
p	p	112	112	112	112	112	112	112
paragraph	¶		182	182	182	182	166	182
parenleft	(	40	40	40	40	40	40	40
parenright	)	41	41	41	41	41	41	41
percent	%	37	37	37	37	37	37	37
period	.	46	46	46	46	46	46	46
periodcentered	˙		183	183	180	183	225	183
perthousand	‰	24	137		189	137	228	139
plus	+	43	43	43	43	43	43	43
plusminus	±		177	177		177	177	177
q	q	113	113	113	113	113	113	113
question	?	63	63	63	63	63	63	63
questiondown	¿	190	191	191	191	191	192	191
quotedbl	"	34	34	34	34	34	34	34
quotedblbase	„	18	132		185	132	227	140
quotedblleft	“	16	147		170	147	210	141
quotedblright	”	17	148		186	148	211	142
quoteleft	‘	96	145	96	96	145	212	143
quoteright	’	39	146	39	39	146	213	144
quotesinglbase	’	13	130		184	130	226	145
quotesingle	’		31		169	39	39	39
r	r	114	114	114	114	114	114	114
racute	ř	175						
rcaron	ř	176						
registered	®		174	174		174	168	174
ring	°	6	9	154	202	30	251	30
s	s	115	115	115	115	115	115	115

Name	Glyph	EC	8r	ISO Latin 1	Adobe Standard	Windows ANSI	Mac	PDF
sacute	ś	177						
scaron	š	178	154			154		157
scedilla	ş	179						
section	§	159	167	167	167	167	164	167
semicolon	;	59	59	59	59	59	59	59
seven	7	55	55	55	55	55	55	55
sfthyphen	-			173				
six	6	54	54	54	54	54	54	54
slash	/	47	47	47	47	47	47	47
space			32	32	32	160	32	32
sterling	£	191	163	163	163	163	163	163
t	t	116	116	116	116	116	116	116
tcedilla	ţ	181						
thorn	þ	254	254	254		254		254
three	3	51	51	51	51	51	51	51
threequarters	¾		190	190		190		190
threesuperior	³		179	179		179		179
tilde	~	3	152	148	196	152	247	31
tquoteright	ʹ	180						
trademark	™		153			153	170	146
two	2	50	50	50	50	50	50	50
twosuperior	²		178	178		178		178
u	u	117	117	117	117	117	117	117
uacute	ú	250	250	250		250	156	250
ucircumflex	û	251	251	251		251	158	251
udieresis	ü	252	252	252		252	159	252
ugrave	ù	249	249	249		249	157	249
uhungarumlaut	ű	182						
underscore	_	95	95	95	95	95	95	95
uring	ü	183						
v	v	118	118	118	118	118	118	118
visiblespace		32						
w	w	119	119	119	119	119	119	119
x	x	120	120	120	120	120	120	120
y	y	121	121	121	121	121	121	121
yacute	ý	253	253	253		253		253
ydieresis	ÿ	184	255	255		255	216	255
yen	¥		165	165	165	165	180	165
z	z	122	122	122	122	122	122	122
zacute	ź	185						

Name	Glyph	EC	8r	ISO Latin 1	Adobe Standard	Windows ANSI	Mac	PDF
zcaron	ž	186	15					158
zdotaccent	ẏ	187						
zero	0	48	48	48	48	48	48	48

Table A.4: Font encoding table, by number.

Decimal	Octal	Hexadecimal	EC	TeXBase I	ISO Latin 1	Standard	Windows ANSI	Mac Roman	PDF
0	0	0	grave						
1	1	1	acute	dotaccent					
2	2	2	circumflex	fi					
3	3	3	tilde	fi					
4	4	4	dieresis	fraction					
5	5	5	hungarumlaut	hungarumlaut					
6	6	6	ring	Lslash					
7	7	7	caron	lslash					
8	10	8	breve	ogonek					
9	11	9	macron	ring					
10	12	a	dotaccent						
11	13	b	cedilla	breve					
12	14	c	ogonek	minus					
13	15	d	quotesingbase						
14	16	e	guilsinglleft	Zcaron					
15	17	f	guilsinglright	zcaron					
16	20	10	quotedblleft	caron					
17	21	11	quotedblright	dotlessi					
18	22	12	quotedblbase	dotlessj					
19	23	13	guillemotleft	ff					
20	24	14	guillemotright	ffi					
21	25	15	endash	ffi					
22	26	16	emdash						
23	27	17	compoundwordmark						
24	30	18	perthousand				breve		breve
25	31	19	dotlessi				caron		caron
26	32	1a	dotlessj				circumflex		circumflex
27	33	1b	ff				dotaccent		dotaccent
28	34	1c	fi				hungarumlaut		hungarumlaut
29	35	1d	fi				ogonek		ogonek
30	36	1e	ffi				ring		ring
31	37	1f	ffi				tilde		tilde
32	40	20	visiblespace	space	space	space	space	space	space
33	41	21	exclam	exclam	exclam	exclam	exclam	exclam	exclam
34	42	22	quotedbl	quotedbl	quotedbl	quotedbl	quotedbl	quotedbl	quotedbl
35	43	23	numbersign	numbersign	numbersign	numbersign	numbersign	numbersign	numbersign
36	44	24	dollar	dollar	dollar	dollar	dollar	dollar	dollar
37	45	25	percent	percent	percent	percent	percent	percent	percent

Decimal	Octal	Hexadecimal	EC	TeXBase I	ISO Latin 1	Standard	Windows ANSI	Mac Roman	PDF
39	47	27	quoteright	quoteright	quoteright	quoteright	quotesingle	quotesingle	quotesingle
40	50	28	parenleft	parenleft	parenleft	parenleft	parenleft	parenleft	parenleft
41	51	29	parenright	parenright	parenright	parenright	parenright	parenright	parenright
42	52	2a	asterisk	asterisk	asterisk	asterisk	asterisk	asterisk	asterisk
43	53	2b	plus	plus	plus	plus	plus	plus	plus
44	54	2c	comma	comma	comma	comma	comma	comma	comma
45	55	2d	hyphen	hyphen	minus	hyphen	hyphen	hyphen	hyphen
46	56	2e	period	period	period	period	period	period	period
47	57	2f	slash	slash	slash	slash	slash	slash	slash
48	60	30	zero	zero	zero	zero	zero	zero	zero
49	61	31	one	one	one	one	one	one	one
50	62	32	two	two	two	two	two	two	two
51	63	33	three	three	three	three	three	three	three
52	64	34	four	four	four	four	four	four	four
53	65	35	five	five	five	five	five	five	five
54	66	36	six	six	six	six	six	six	six
55	67	37	seven	seven	seven	seven	seven	seven	seven
56	70	38	eight	eight	eight	eight	eight	eight	eight
57	71	39	nine	nine	nine	nine	nine	nine	nine
58	72	3a	colon	colon	colon	colon	colon	colon	colon
59	73	3b	semicolon	semicolon	semicolon	semicolon	semicolon	semicolon	semicolon
60	74	3c	less	less	less	less	less	less	less
61	75	3d	equal	equal	equal	equal	equal	equal	equal
62	76	3e	greater	greater	greater	greater	greater	greater	greater
63	77	3f	question	question	question	question	question	question	question
64	100	40	@	at	at	at	at	at	at
65	101	41	A	A	A	A	A	A	A
66	102	42	B	B	B	B	B	B	B
67	103	43	C	C	C	C	C	C	C
68	104	44	D	D	D	D	D	D	D
69	105	45	E	E	E	E	E	E	E
70	106	46	F	F	F	F	F	F	F
71	107	47	G	G	G	G	G	G	G
72	110	48	H	H	H	H	H	H	H
73	111	49	I	I	I	I	I	I	I
74	112	4a	J	J	J	J	J	J	J
75	113	4b	K	K	K	K	K	K	K
76	114	4c	L	L	L	L	L	L	L
77	115	4d	M	M	M	M	M	M	M
78	116	4e	N	N	N	N	N	N	N
79	117	4f	O	O	O	O	O	O	O
80	120	50	P	P	P	P	P	P	P
81	121	51	Q	Q	Q	Q	Q	Q	Q
82	122	52	R	R	R	R	R	R	R
83	123	53	S	S	S	S	S	S	S
84	124	54	T	T	T	T	T	T	T
85	125	55	U	U	U	U	U	U	U
86	126	56	V	V	V	V	V	V	V
87	127	57	W	W	W	W	W	W	W
88	130	58	X	X	X	X	X	X	X
89	131	59	Y	Y	Y	Y	Y	Y	Y

Decimal	Octal	Hexadecimal	EC	TeXBase1	ISO Latin 1	Standard	Windows ANSI	Mac Roman	PDF
89 131 59	Y	Y	Y	Y	Y	Y	Y	Y	Y
90 132 5a	Z	Z	Z	Z	Z	Z	Z	Z	Z
91 133 5b	[	[	bracketleft	bracketleft	bracketleft	bracketleft	bracketleft	bracketleft	bracketleft
92 134 5c	\	\	backslash	backslash	backslash	backslash	backslash	backslash	backslash
93 135 5d	]	]	bracketright	bracketright	bracketright	bracketright	bracketright	bracketright	bracketright
94 136 5e	~	~	asciicircum	asciicircum	asciicircum	asciicircum	asciicircum	asciicircum	asciicircum
95 137 5f	`	`	underscore	underscore	underscore	underscore	underscore	underscore	underscore
96 140 60	~	~	quoteleft	quoteleft	quoteleft	quoteleft	grave	grave	grave
97 141 61	a	a	a	a	a	a	a	a	a
98 142 62	b	b	b	b	b	b	b	b	b
99 143 63	c	c	c	c	c	c	c	c	c
100 144 64	d	d	d	d	d	d	d	d	d
101 145 65	e	e	e	e	e	e	e	e	e
102 146 66	f	f	f	f	f	f	f	f	f
103 147 67	g	g	g	g	g	g	g	g	g
104 150 68	h	h	h	h	h	h	h	h	h
105 151 69	i	i	i	i	i	i	i	i	i
106 152 6a	j	j	j	j	j	j	j	j	j
107 153 6b	k	k	k	k	k	k	k	k	k
108 154 6c	l	l	l	l	l	l	l	l	l
109 155 6d	m	m	m	m	m	m	m	m	m
110 156 6e	n	n	n	n	n	n	n	n	n
111 157 6f	o	o	o	o	o	o	o	o	o
112 160 70	p	p	p	p	p	p	p	p	p
113 161 71	q	q	q	q	q	q	q	q	q
114 162 72	r	r	r	r	r	r	r	r	r
115 163 73	s	s	s	s	s	s	s	s	s
116 164 74	t	t	t	t	t	t	t	t	t
117 165 75	u	u	u	u	u	u	u	u	u
118 166 76	v	v	v	v	v	v	v	v	v
119 167 77	w	w	w	w	w	w	w	w	w
120 170 78	x	x	x	x	x	x	x	x	x
121 171 79	y	y	y	y	y	y	y	y	y
122 172 7a	z	z	z	z	z	z	z	z	z
123 173 7b	{	{	braceleft	braceleft	braceleft	braceleft	braceleft	braceleft	braceleft
124 174 7c			bar	bar	bar	bar	bar	bar	bar
125 175 7d	}	}	braceright	braceright	braceright	braceright	braceright	braceright	braceright
126 176 7e	~	~	asciitilde	asciitilde	asciitilde	asciitilde	asciitilde	asciitilde	asciitilde
127 177 7f	-	-	hyphen						
128 200 80	Å	Å	Abreve					Adieresis	
129 201 81	Ä	Ä	Aogonek					Aring	dagger
130 202 82	Ć	Ć	Cacute	quotesingbase			quotesingbase	Ccedilla	daggerdbl
131 203 83	Č	Č	Ccaron	florin			florin	Eacute	ellipsis
132 204 84	Ď	Ď	Dcaron	quotedblbase			quotedblbase	Ntilde	emdash
133 205 85	Ě	Ě	Ecaron	ellipsis			ellipsis	Odieresis	endash
134 206 86	Ę	Ę	Eogonek	dagger			dagger	Udieresis	florin
135 207 87	Ğ	Ğ	Gbreve	daggerdbl			daggerdbl	aacute	fraction
136 210 88	Ł	Ł	Lacute	circumflex			circumflex	agrave	guilsingleft
137 211 89	Ł	Ł	Lquoteright	perthousand			perthousand	acircumflex	guilsingright
138 212 8a	Š	Š	Lslash	Scaron			Scaron	adieresis	minus

Decimal	Octal	Hexadecimal	EC	TeXBase1	ISO Latin 1	Standard	Windows ANSI	Mac Roman	PDF
140 214 8c	N	N	Ncaron	OE			OE	aring	quotedblbase
141 215 8d	Ŋ	Ŋ	Eng					ccedilla	quotedblleft
142 216 8e	Ő	Ő	Ohungarumlaut					ecacute	quotedblright
143 217 8f	Ŕ	Ŕ	Racute					egrave	quoteleft
144 220 90	Ŗ	Ŗ	Rcaron		dotlessi			ecircumflex	quoteright
145 221 91	Š	Š	Sacute	quoteleft	grave		quoteleft	edieresis	quotesingbase
146 222 92	Ŝ	Ŝ	Scaron	quoteright	acute		quoteright	iacute	trademark
147 223 93	Š	Š	Scedilla	quotedblleft	circumflex		quotedblleft	igrave	fi
148 224 94	Ť	Ť	Tcaron	quotedblright	tilde		quotedblright	icircumflex	fi
149 225 95	Ŧ	Ŧ	Tcedilla	bullet	macron			idieresis	Lslash
150 226 96	Ū	Ū	Uhungarumlaut	endash	breve		endash	ntilde	Oe
151 227 97	Ů	Ů	Uring	emdash	dotaccent		emdash	oacute	Scaron
152 230 98	Ÿ	Ÿ	Ydieresis	tilde	dieresis		tilde	ograve	Ydieresis
153 231 99	Ž	Ž	Zacute	trademark			trademark	ocircumflex	Zcaron
154 232 9a	Ž	Ž	Zcaron	scaron	ring		scaron	odieresis	dotlessi
155 233 9b	Ž	Ž	Zdotaccent	guilsingright	cedilla		guilsingright	otilde	Lslash
156 234 9c	ı	ı	IJ	oe			uacute	oe	oe
157 235 9d	ı	ı	Idotaccent		hungarumlaut			ugrave	scaron
158 236 9e	ı	ı	Idotaccent		ogonek			ucircumflex	zcaron
159 237 9f	ı	ı	section	Ydieresis			Ydieresis	udieresis	
160 240 a0	ı	ı	abreve		nbspspace		space	dagger	
161 241 a1	ı	ı	aogonek	exclamdown	exclamdown	exclamdown	exclamdown	degree	exclamdown
162 242 a2	ı	ı	acacute	cent	cent	cent	cent	cent	cent
163 243 a3	ı	ı	ccaron	sterling	sterling	sterling	sterling	sterling	sterling
164 244 a4	ı	ı	dquoteright	currency	currency	fraction	currency	section	currency
165 245 a5	ı	ı	ecaron	yen	yen	yen	yen	bullet	yen
166 246 a6	ı	ı	eogonek	brokenbar	brokenbar	florin	brokenbar	paragraph	brokenbar
167 247 a7	ı	ı	gbreve	section	section	section	section	germandbls	section
168 250 a8	ı	ı	lacute	dieresis	dieresis	currency	dieresis	registered	registered
169 251 a9	ı	ı	lquoteright	copyright	copyright	quotesingle	copyright	copyright	copyright
170 252 aa	ı	ı	lslash	ordfeminine	ordfeminine	quotedblleft	ordfeminine	trademark	ordfeminine
171 253 ab	ı	ı	macute	guillemotleft	guillemotleft	guillemotleft	guillemotleft	acute	guillemotleft
172 254 ac	ı	ı	ncaron	logicalnot	logicalnot	guilsingleft	logicalnot	dieresis	logicalnot
173 255 ad	ı	ı	eng	hyphen	sfthyphen	guilsingright	hyphen	notequal	
174 256 ae	ı	ı	ohungarumlaut	registered	registered	fi	registered	AE	registered
175 257 af	ı	ı	racute	macron	macron	fi	macron	Osash	macron
176 260 b0	ı	ı	rcaron	degree	degree		degree	infinity	degree
177 261 b1	ı	ı	sacute	plusminus	plusminus	endash	plusminus	plusminus	plusminus
178 262 b2	ı	ı	scaron	twosuperior	twosuperior	dagger	twosuperior	lessequal	twosuperior
179 263 b3	ı	ı	scedilla	threesuperior	threesuperior	daggerdbl	threesuperior	greaterqual	threesuperior
180 264 b4	ı	ı	tquoteright	acute	acute	periodcentered	acute	yen	acute
181 265 b5	ı	ı	tcedilla	mu	mu		mu	mu	mu
182 266 b6	ı	ı	uhungarumlaut	paragraph	paragraph	paragraph	paragraph	partialdiff	paragraph
183 267 b7	ı	ı	uring	periodcentered	periodcentered	bullet	periodcentered	summation	periodcentered
184 270 b8	ı	ı	ydiereis	cedilla	cedilla	quotesingbase	cedilla	product	cedilla
185 271 b9	ı	ı	zacute	onesuperior	onesuperior	quotedblbase	onesuperior	pi	onesuperior
186 272 ba	ı	ı	zcaron	ordmasculine	ordmasculine	quotedblright	ordmasculine	integral	ordmasculine
187 273 bb	ı	ı	zdotaccent	guillemotright	guillemotright	guillemotright	guillemotright	ordfeminine	guillemotright
188 274 bc	ı	ı	ij	onequarter	onequarter	ellipsis	onequarter	ordmasculine	onequarter
189 275 bd	ı	ı	exclamdown	onehalf	onehalf	perthousand	onehalf	Omega	onehalf

Decimal	Octal	Hexadecimal	EC Glyph	EC	TeXBase1	ISO Latin 1	Standard	Windows ANSI	Mac Roman	PDF
190 276	be	z	questiondown	threequarters	questiondown	threequarters	ae	threequarters		
191 277	bf	£	sterling	questiondown	questiondown	questiondown	oslash	questiondown		
192 300	c0	À	Agrave	Agrave	Agrave	Agrave	questiondown	Agrave		
193 301	c1	Á	Aacute	Aacute	Aacute	grave	exclamdown	Aacute		
194 302	c2	Â	Acircumflex	Acircumflex	Acircumflex	acute	logicalnot	Acircumflex		
195 303	c3	Ã	Atilde	Atilde	Atilde	circumflex	radical	Atilde		
196 304	c4	Ä	Adieresis	Adieresis	Adieresis	tilde	florin	Adieresis		
197 305	c5	Å	Aring	Aring	Aring	macron	approxequal	Aring		
198 306	c6	Æ	AE	AE	AE	breve	Delta	AE		
199 307	c7	Ç	Ccedilla	Ccedilla	Ccedilla	dotaccent	guillemotleft	Ccedilla		
200 310	c8	È	Egrave	Egrave	Egrave	dieresis	guillemotright	Egrave		
201 311	c9	É	Eacute	Eacute	Eacute		ellipsis	Eacute		
202 312	ca	Ê	Ecircumflex	Ecircumflex	Ecircumflex	ring	blank	Ecircumflex		
203 313	cb	Ë	Edieresis	Edieresis	Edieresis	cedilla	Agrave	Edieresis		
204 314	cc	Ì	Igrave	Igrave	Igrave		Atilde	Igrave		
205 315	cd	Í	Iacute	Iacute	Iacute	hungarumlaut	Otilde	Iacute		
206 316	ce	Î	Icircumflex	Icircumflex	Icircumflex	ogonek	OE	Icircumflex		
207 317	cf	Ï	Idieresis	Idieresis	Idieresis	caron	oe	Idieresis		
208 320	d0	Ð	Eth	Eth	Eth	emdash	endash	Eth		
209 321	d1	Ñ	Ntilde	Ntilde	Ntilde		emdash	Ntilde		
210 322	d2	Ó	Ograve	Ograve	Ograve		quotedblleft	Ograve		
211 323	d3	Ô	Oacute	Oacute	Oacute		quotedblright	Oacute		
212 324	d4	Ö	OCircumflex	OCircumflex	OCircumflex		quoteleft	OCircumflex		
213 325	d5	Õ	Otilde	Otilde	Otilde		quoteright	Otilde		
214 326	d6	Ö	Odieresis	Odieresis	Odieresis		divide	Odieresis		
215 327	d7	Œ	OE	multiply	multiply		lozenge	multiply		
216 330	d8	Ø	Oslash	Oslash	Oslash		ydieresis	Oslash		
217 331	d9	Ù	Ugrave	Ugrave	Ugrave		Ydieresis	Ugrave		
218 332	da	Ú	Uacute	Uacute	Uacute		fraction	Uacute		
219 333	db	Û	Ucircumflex	Ucircumflex	Ucircumflex		currency	Ucircumflex		
220 334	dc	Ü	Udieresis	Udieresis	Udieresis		guilsingleft	Udieresis		
221 335	dd	Ý	Yacute	Yacute	Yacute		guilsingright	Yacute		
222 336	de	Þ	Thorn	Thorn	Thorn		fi	Thorn		
223 337	df	SS	Germandbls	germandbls	germandbls		fi	germandbls		
224 340	e0	à	agrave	agrave	agrave		daggerdbl	agrave		
225 341	e1	á	aacute	aacute	aacute	AE	periodcentered	aacute		
226 342	e2	â	acircumflex	acircumflex	acircumflex		quotesingbase	acircumflex		
227 343	e3	ã	atilde	atilde	atilde	ordfeminine	quotedblbase	atilde		
228 344	e4	ä	adieresis	adieresis	adieresis		perthousand	adieresis		
229 345	e5	å	aring	aring	aring		Acircumflex	aring		
230 346	e6	æ	ae	ae	ae		Ecircumflex	ae		
231 347	e7	ç	ccedilla	ccedilla	ccedilla		Aacute	ccedilla		
232 350	e8	è	egrave	egrave	egrave	Lslash	Edieresis	egrave		
233 351	e9	é	eacute	eacute	eacute	Oslash	Egrave	eacute		
234 352	ea	ê	ecircumflex	ecircumflex	ecircumflex	OE	Iacute	ecircumflex		
235 353	eb	ë	edieresis	edieresis	edieresis	ordmasculine	Icircumflex	edieresis		
236 354	ec	ì	igrave	igrave	igrave		Idieresis	igrave		
237 355	ed	í	iacute	iacute	iacute		Igrave	iacute		
238 356	ee	î	icircumflex	icircumflex	icircumflex		Oacute	icircumflex		
239 357	ef	ï	idieresis	idieresis	idieresis		OCircumflex	idieresis		

Decimal	Octal	Hexadecimal	EC Glyph	EC	TeXBase1	ISO Latin 1	Standard	Windows ANSI	Mac Roman	PDF
240 360	f0	ð	eth	eth	eth		eth	apple		eth
241 361	f1	ñ	ntilde	ntilde	ntilde		ae	ntilde	Ograve	ntilde
242 362	f2	ò	ograve	ograve	ograve			ograve	Uacute	ograve
243 363	f3	ó	oacute	oacute	oacute			oacute	Ucircumflex	oacute
244 364	f4	ô	ocircumflex	ocircumflex	ocircumflex			ocircumflex	Ugrave	ocircumflex
245 365	f5	ö	otilde	otilde	otilde	dotlessi		otilde	dotlessi	otilde
246 366	f6	ø	odieresis	odieresis	odieresis			odieresis	circumflex	odieresis
247 367	f7	œ	oe	divide	divide			divide	tilde	divide
248 370	f8	ø	oslash	oslash	oslash	lslash		oslash	macron	oslash
249 371	f9	ù	ugrave	ugrave	ugrave	oslash		ugrave	breve	ugrave
250 372	fa	ú	uacute	uacute	uacute	oe		uacute	dotaccent	uacute
251 373	fb	û	ucircumflex	ucircumflex	ucircumflex	germandbls		ucircumflex	ring	ucircumflex
252 374	fc	ü	udieresis	udieresis	udieresis			udieresis	cedilla	udieresis
253 375	fd	ý	yacute	yacute	yacute			yacute	hungarumlaut	yacute
254 376	fe	þ	thorn	thorn	thorn			thorn	ogonek	thorn
255 377	ff	ß	germandbls	ydieresis	ydieresis			ydieresis	caron	ydieresis

A.7 fontinst reference information

A.7.1 Encoding files

An *encoding file* is a TeX document with the following structure, where the *encoding commands* describe the font encoding:

```
\relax
  ignored material
\encoding
  encoding commands
\endencoding
  ignored material
```

Since the encoding file ignores any material between `\relax` and `\encoding`, an encoding file can also be processed by L^AT_EX normally; the fontdoc package is used to typeset the encoding statements. For example:

```
\documentclass{article}
\usepackage{fontdoc}

\begin{document}
This file documents an encoding.
\encoding
...
\endencoding
And that is the end
\end{document}
```

The *encoding commands* describe “slots” or numerical positions in the encoding, using the following commands:

`\nextslot{number}`

Sets the number of the next slot. If there is no `\nextslot` command, the number is the successor of the previous slot.

`\setslot{glyph} slot commands \endsetslot`

Sets the slot of *glyph*. The *slot commands* describe the glyph and give its usage in T_EX.

`\inputetx{file}`

Inputs the *encoding commands* of file .etx.

The encoding files can set the variable `\codingscheme` (the font coding scheme), and the integers (using `\setint`) `\boundarychar` (the slot of the boundary character), and `\letterspacing` (the extra space to be added between every glyph); additionally:

`\fontdimen{n}`

Sets the value of the *n*th font dimension.

The encoding file generates an integer variable for each slot in the encoding that can be referred to elsewhere; the slot number for *glyph* is available as `\int{glyph}`.

The possible *slot commands* are:

`\usedas{type}{control sequence}`

Sets the T_EX type for the character in this slot, with the *type* taken from: *accent*, *char*, *mathaccent*, *mathbin*, *mathclose*, *mathdelim*, *mathopen*, *mathord*, *mathpunct*, *mathrel*, *mathvariable*

`\nextlarger{glyph}`

Sets a *nextlarger* entry from the current slot to the *glyph*; this is used in creating math fonts.

`\varchar varchar commands \endvarchar`

Sets a *varchar* entry for the current slot, using the commands:

`\vartop{glyph}`

Sets the top *glyph* of the *varchar*

`\varmid{glyph}`

Sets the middle *glyph* of the *varchar*

`\varbot{glyph}`

Sets the bottom *glyph* of the *varchar*

`\varrep{glyph}`

Sets the repeated *glyph* of the *varchar*

A.7.2 Metric files

A *metric file* is a T_EX document which describes the glyphs in a font. It has the following structure:

```
\relax
ignored material
\metrics
metric commands
\endmetrics
ignored material
```

The *metric commands* are as follows:

`\setglyph{name} glyph commands \endsetglyph`

Builds the glyph called *name* using the *glyph commands*, if it is not already defined.

`\resetglyph{name} glyph commands \endsetglyph`

Defines the glyph called *name* using the *glyph commands*, even if it has already been defined.

`\unsetglyph{name}`

Makes the glyph called *name* undefined.

`\setrawglyph{name}{font}{size}{slot}{width}{height}{depth}{italic correction}`

Sets a glyph called *name* from the *font* that has the given design size, slot, width, height, depth, and italic correction. This command is usually generated automatically from an afm or pl file.

`\setkern{glyph}{glyph}{amount}`

Sets a kern between the two glyphs.

`\setleftkerning{glyph}{glyph}{amount}`

Sets the proportion (out of 1000) by which the first glyph should copy the left kern of the second glyph; thus the following commands say that the glyph *Asmall* (small caps A) should copy its left kerning from that of the normal A in a proportion of 85%, but that the glyph *IJ* should have just the same left kerning values as the normal I.

```
\setleftkerning{Asmall}{A}{850}
```

```
\setleftkerning{IJ}{I}{1000}
```

`\setrightkerning{glyph}{glyph}{amount}`

Sets the amount by which the first glyph should copy the right kern of the second glyph.

`\inputmtx{file}`

Inputs the *metric commands* of file .mtx.

The *metrics files* can also set `\ascender` (height of tallest lower-case letter), `\capheight` (height of tallest capital letter), `\descender` (depth of lowest lower-case letter), `\italicslant` (ratio of italic slant, given in units of rightward movement for each 1000 units of upward movement), `\minimumkern` (amount below which kern is ignored), `\monowidth` (set if font is monowidth), `\underlinethickness` (width of underline rule), `\xheight` (x-height), and `\designsize` (design size of the font).

The *glyph commands* are as follows:

```
\glyph{glyph}{scale}
```

Sets the named glyph at the given scale. This:

- Advances the current glyph width.
- Sets the current glyph height to be at least the height of the named glyph, adjusted for the current vertical offset.
- Sets the current glyph depth to be at least the depth of the named glyph, adjusted for the current vertical offset.
- Sets the current glyph italic correction to be the same as the set glyph.

The named glyph must have already been defined, otherwise an error occurs. For example:

```
\setglyph{fi}
\glyph{f}{1000}
\glyph{i}{1000}
\endsetglyph
\setglyph{Asmall}
\glyph{A}{700}
\endsetglyph
```

`\glyphrule{integer}{integer}`

Sets a rule of the given width and height; for example:

```
\setglyph{underline}
\glyphrule{333}{40}
\endsetglyph
```

`\glyphspecial{text}`

Sets a device-dependent `\special`; for example:

```
\setglyph{crest}
\glyphspecial{Filename: crest.eps}
\endsetglyph
```

`\glyphwarning{text}`

Sets a warning `\special` and produces a warning message each time the glyph is used; for example:

```
\setglyph{missingglyph}
\glyphrule{500}{500}
\glyphwarning{missing glyph}
\endsetglyph
```

`\movert{integer expression}`

Moves right by the given amount and advances the current glyph width; for example:

```
\setglyph{Asmall}
\movert{50}
\glyph{A}{700}
\movert{50}
\setendglyph
```

`\moveup{integer expression}`

Moves up by the given amount and advances the current vertical offset. Each glyph should always end with vertical offset zero; for example:

```
\setglyph{onehalf}
\moveup{500}
\glyph{one}{700}
\moveup{-500}
\glyph{slash}{1000}
\moveup{-200}
\glyph{two}{700}
```

```
\moveup{200}
\endsetglyph
```

`\push glyph commands \pop`

Performs the *glyph commands* without adjusting the current position or glyph width; for example:

```
\setglyph{aacute}
\push
\movert{\div{\sub{\width{a}}{\width{acute}}}{2}}
\glyph{acute}{1000}
\pop
\glyph{a}{1000}
\endsetglyph
```

`\resetwidth{integer expression}`

Sets the width of the current glyph.

`\resetheight{integer expression}`

Sets the height of the current glyph.

`\resetdepth{integer expression}`

Sets the depth of the current glyph.

`\resetitalic{integer expression}`

Sets the italic correction of the current glyph.

Inside the definition of *glyph*, you can use expressions such as `\width{glyph}`, which refers to the glyph defined so far. For example, a display summation can be defined as a text summation $\sum$ scaled 120% with 0.5pt extra height and depth with the commands:

```
\setglyph{sumdisplay}
\glyph{textsum}{1200}
\resetheight{\add{\height{sumdisplay}}{50}}
\resetdepth{\add{\depth{sumdisplay}}{50}}
\endsetglyph
```

Within a `\resetglyph`, these expressions refer to the previous definition of the glyph. For example, you can add sidebearings to the letter “A” with:

```
\resetglyph{A}
\movert{25}
\glyph{A}{1000}
\movert{25}
\endresetglyph
```

A.7.3 Low-level fontinst commands

A large number of utility command are available for writing fontinst instruction files:

`\ifiscommand{command}\then`

Expands to `\iftrue` if the command *command* is defined, and `\iffalse` otherwise.

`\ifisdim{dim}\then`
Expands to `\iftrue` if the dimension variable *dim* is defined, and `\iffalse` otherwise.

`\ifisglyph{glyph}\then`
Expands to `\iftrue` if the glyph variable *glyph* is defined, and `\iffalse` otherwise.

`\ifisint{int}\then`
Expands to `\iftrue` if the integer variable *int* is defined, and `\iffalse` otherwise.

`\ifisstr{str}\then`
Expands to `\iftrue` if the string variable *str* is defined, and `\iffalse` otherwise.

`\needsfontinstversion{version}`
Issues a warning if the current version of the `\fontinst` package is less than *version*.

`\resetcommand{command}{definition}`
A synonym for `\def`.

`\resetdim{dim}{dimension}`
The dimension variable *dim* is defined to be the current value of *dimension*.

`\resetint{int}{integer expression}`
The integer variable *int* is defined to be the current value of *integer expression*.

`\resetstr{str}{string}`
The string variable *str* is defined to be the current value of *string*.

`\setcommand{command}{definition}`
If the command *command* is currently undefined, it is defined to be *definition*. This uses the same syntax as the TeX `\def` command.

`\setdim{dim}{dimension}`
If the dimension variable *dim* is currently undefined, it is defined to be the current value of *dimension*.

`\setint{int}{integer expression}`
If the integer variable *int* is currently undefined, it is defined to be the current value of *integer expression*.

`\setstr{str}{string}`
If the string variable *str* is currently undefined, it is defined to be the current value of *string*.

`\unsetcommand{command}`
Makes *command* an undefined command.

`\unsetdim{dim}`
Makes *dim* an undefined dimension.

`\unsetint{int}`
Makes *int* an undefined integer.

`\unsetstr{str}`
Makes *str* an undefined string.

`\int{int}`
The value of the integer variable *int*.

`\width{glyph}`
The width of the glyph variable *glyph*.

`\height{glyph}`
The height of the glyph variable *glyph*.

`\depth{glyph}`
The depth of the glyph variable *glyph*.

`\italic{glyph}`
The italic correction of the glyph variable *glyph*.

`\kerning{left}{right}`
The kerning between the *left* and *right* glyph variables.

`\neg{integer expression}`
The negation of the *integer expression*.

`\add{integer expression}{integer expression}`
The sum of the two *integer expressions*.

`\sub{integer expression}{integer expression}`
The first *integer expression* minus the second.

`\mul{integer expression}{integer expression}`
The multiplication of the two *integer expressions*.

`\div{integer expression}{integer expression}`
The first *integer expression* divided by the second.

`\scale{integer expression}{integer expression}`
The first *integer expression* times the second, divided by 1000.

`\declaresize{size}{fd-size-range}`
This declares a new *size*, and gives the *fd* commands for it. For example, `fontinst.sty` declares the following sizes:

```
\declaresize{}{<->}
\declaresize{5}{<5>}
\declaresize{6}{<6>}
\declaresize{7}{<7>}
\declaresize{8}{<8>}
\declaresize{9}{<9>}
\declaresize{10}{<10>}
\declaresize{11}{<10.95>}
\declaresize{12}{<12>}
\declaresize{14}{<14.4>}
\declaresize{17}{<17.28>}
\declaresize{20}{<20.74>}
\declaresize{25}{<24.88>}
```

`\declareencoding{string}{etx}`
This declares which *etx* file corresponds to which encoding string. For example, `fontinst.sty` declares the following encoding strings:

```
\declareencoding{EXTENDED TEX FONT ENCODING - LATIN}{T1}
\declareencoding{TEX TEXT}{OT1}
\declareencoding{TEX TYPEWRITER TEXT}{OT1TT}
\declareencoding{TEX MATH ITALIC}{OML}
\declareencoding{TEX MATH SYMBOLS}{OMS}
\declareencoding{TEX MATH EXTENSION}{OMX}
```


A.8 Ghostscript drivers

DOS and Windows screen display

ega	EGA (640×350, 16-color)
vga	VGA (640×480, 16-color)
s3vga	SuperVGA with S3 86C911 chip (e.g., Diamond Stealth board)
svgalib	Generic SuperVGA in 800×600, 16-color mode
tseng	SuperVGA using Tseng Labs ET3000/4000 chips, 256-color modes
tvga	Trident SuperVGA, 256-color modes
vesa	SuperVGA with VESA standard API driver

OS/2 screen display

os2pm	OS/2 Presentation Manager
os2dll	OS/2 DLL bitmap
os2prn	OS/2 printer

Unix and VMS screen display

lvga256	Linux vgalib, 256-color VGA modes
vgalib	Linux PC with vgalib
x11	X Windows System version 11, release >=4
x11alpha	X Windows grayscale

Printers

ap3250	Epson AP3250 printer
bj10e	Canon BubbleJet BJ10e
bj200	Canon BubbleJet BJ200
bjc600	Canon Color BubbleJet BJC-600, BJC-4000, and BJC-70
bjc800	Canon Color BubbleJet BJC-800
ccr	CalComp Raster format
cdeskjet	HP DeskJet 500C with 1 bit/pixel color
cdjcolor	HP DeskJet 500C with 24 bit/pixel color and high-quality color (Floyd-Steinberg) dithering; suitable for DeskJet 540C
cdjmono	HP DeskJet 500C printing black only; suitable for DeskJet 510, 520, and 540C (black only)
cdj500	HP DeskJet 500C (same as cdjcolor)
cdj550	HP DeskJet 550C/560C
deskjet	HP DeskJet and DeskJet Plus
djet500	HP DeskJet 500
djet500c	HP DeskJet 500C alternate driver (does not work on 550C or 560C)
dnj650c	HP DesignJet 650C
epson	Epson-compatible dot matrix printers (9- or 24-pin)
eps9mid	Epson-compatible 9-pin, interleaved lines (intermediate resolution)
eps9high	Epson-compatible 9-pin, interleaved lines (triple resolution)
epsonc	Epson LQ-2550 and Fujitsu 3400/2400/1200 color printers
ibmpro	IBM 9-pin Proprinter
imagen	Imagen ImPress printers
iwhi	Apple Imagewriter in high-resolution mode
iwlo	Apple Imagewriter in low-resolution mode
iwlq	Apple Imagewriter LQ in 320×216 dpi mode
jep3852	IBM Jetprinter ink-jet color printer (Model 3852)

laserjet	HP LaserJet
la50	DEC LA50 printer
la70	DEC LA70 printer
la70t	DEC LA70 printer with low-resolution text enhancement
la75	DEC LA75 printer
la75plus	DEC LA75plus printer
lbp8	Canon LBP-8II laser printer
lips3	Canon LIPS III laser printer in English (CaPSL) mode
ln03	DEC LN03 printer
lj250	DEC LJ250 Companion color printer
ljet2p	HP LaserJet IId/IIp/III with TIFF compression
ljet3	HP LaserJet III with Delta Row compression
ljet3d	HP LaserJet IIId with duplex capability
ljet4	HP LaserJet 4 (defaults to 600 dpi)
lj4dith	HP LaserJet 4 with Floyd-Steinberg dithering
ljetplus	HP LaserJet Plus
lp2563	HP 2563B line printer
m8510	C.Itoh M8510 printer
necp6	NEC P6/P6/P60 printers at 360×360 dpi
nwp533	Sony Microsystems NWP533 laser printer
oce9050	OCE 9050 printer
oki182	Okidata MicroLine 182
okiibm	Okidata MicroLine IBM-compatible printers
paintjet	alternate HP PaintJet color printer
pj	HP PaintJet driver
pjetxl	alternate HP PaintJet XL driver
pjxl	HP PaintJet XL color printer
pjxl300	HP PaintJet XL300 color printer; suitable for PaintJet 1200C
r4081	Ricoh 4081 laser printer
sj48	StarJet 48 inkjet printer
sparc	SPARCprinter
st800	Epson Stylus 800 printer
stcolor	Epson Stylus Color
t4693d2	Tektronix 4693d color printer, 2 bits per RGB component
t4693d4	Tektronix 4693d color printer, 4 bits per RGB component
t4693d8	Tektronix 4693d color printer, 8 bits per RGB component
tek4696	Tektronix 4695/4696 inkjet plotter
xes	Xerox XES printers (2700, 3700, 4045, etc.)

Fax systems¹

dfaxhigh	DigiBoard, Inc.'s DigiFAX software format (high resolution)
dfaxlow	DigiFAX low (normal) resolution
faxg3	Group 3 fax, with EOLs but no header or EOD
faxg32d	Group 3 2D fax, with EOLs but no header or EOD
faxg4	Group 4 fax, with EOLs but no header or EOD
tiffcrle	TIFF "CCITT RLE 1-dim" (Group 3 fax with no EOLs)
tiffg3	TIFF Group 3 fax (with EOLs)

¹ All these drivers adjust the page size to match one of the three CCITT standard sizes (U.S. letter with A4 width, A4, or B4).

tiffg32d TIFF Group 3 2D fax
tiffg4 TIFF Group 4 fax

Bitmap file formats

bmpmono Monochrome MS Windows BMP file format
bmp16 4-bit (EGA/VGA) BMP file format
bmp256 8-bit (256-color) BMP file format
bmp16m 24-bit BMP file format
cgmono Monochrome (black-and-white) CGM
cgm8 8-bit (256-color) CGM
cgm24 24-bit color CGM
cif CIF file format for VLSI
miff24 ImageMagick MIFF format, 24-bit direct color, RLE compressed
mgrmono 1-bit monochrome MGR devices
mgrgray2 2-bit gray scale MGR devices
mgrgray4 4-bit gray scale MGR devices
mgrgray8 8-bit gray scale MGR devices
mgr4 4-bit (VGA) color MGR devices
mgr8 8-bit color MGR devices
pcxmono PCX file format, monochrome (1-bit black and white)
pcxgray PCX file format, 8-bit gray scale
pcx16 PCX file format, 4-bit planar (EGA/VGA) color
pcx256 PCX file format, 8-bit chunky color
pcx24b PCX file format, 24-bit color (3 8-bit planes)
pbm Portable Bitmap (plain format)
pbmraw Portable Bitmap (raw format)
pgm Portable Graymap (plain format)
pgmraw Portable Graymap (raw format)
pgnm Portable Graymap (plain format), optimizing to PBM if possible
pgnmraw Portable Graymap (raw format), optimizing to PBM if possible
pnm Portable Pixmap (plain format) (RGB), optimizing to PGM or PBM if possible
pnmraw Portable Pixmap (raw format) (RGB), optimizing to PGM or PBM if possible
ppm Portable Pixmap (plain format) (RGB)
ppmraw Portable Pixmap (raw format) (RGB)
pngmono Monochrome Portable Network Graphics (PNG)
pnggray 8-bit gray Portable Network Graphics (PNG)
png16 4-bit color Portable Network Graphics (PNG)
png256 8-bit color Portable Network Graphics (PNG)
png16m 24-bit color Portable Network Graphics (PNG)
psmono PostScript (Level 1) monochrome image
sgirgb SGI RGB pixmap format
tiff12nc TIFF 12-bit RGB, no compression
tiff24nc TIFF 24-bit RGB, no compression (NeXT standard format)
tiff1zw TIFF LZW (monochrome)
tiffpack TIFF PackBits (monochrome)

High-level formats

pdfwrite Portable Document Format

B

Getting all the goodies

We hope that, when reading this book, you will actually be tempted to get one or more of the packages we describe onto your computer and try them out yourself, see how useful they are to your work—or perhaps just have some fun.

Most of the packages and programs described here are freely available on the Internet via ftp (file transfer protocol) or via a WWW (World Wide Web) interface from one of the CTAN (Comprehensive TeX Archive Network (Greenwade, 1993)) sites. Presently there are two main CTAN sites, one in Cambridge, UK and one in Heidelberg, Germany. In addition there are many mirror sites in different parts of the world. Also, several CD-ROMs containing snapshots of the CTAN archives are available, and provide a viable alternative for those with slow links (or without access) to the Internet. In particular, the *TeX Live* CD-ROM is available from various TeX User Groups (see Burbank and Goossens (1996) and <http://www.tug.org/usergroups.html> for an up-to-date list). The TeX Users Group (TUG), the UK TeX Users Group (UK-TUG), and the French TeX Users Group (GUTenberg), in collaboration with members of other groups and helpful individuals, produced this plug-and-play CD-ROM using Thomas Esser's teTeX TeX distribution for Unix, based on Karl Berry's Web2c. Its support tree adheres to the TDS (TeX Directory Structure) tree structure (TUG Working Group on the TeX Directory Structure (TWG-TDS), 1995). It contains a preinstalled runnable system for over twenty Unix-based platforms. Because it is formatted to the ISO 9660 standard, the CD-ROM can also be used on MSDOS and MS-Windows, although the setup is not optimized for such environments. For full details about this CD-ROM, see <http://www.tug.org/texlive.html> or refer to the back of this book.

The first part of this appendix introduces various ways of getting access to the CTAN archives on the Internet, while the second part lists where the packages and utilities described in this book can be found on the Internet.

B.1 Connecting to CTAN

For those of you with a reasonably fast connection to the Internet, downloading the latest version of a package directly from a CTAN archive or one of its mirror sites is probably the quickest alternative. Nevertheless, we feel it is still worthwhile to buy one of the more recent TeX CD-ROMs mentioned in the previous section, since they provide a convenient and complete reference. In particular, all packages described in this book are available on the TeX Live CD-ROM.

Once you decide to download a particular package, you should connect to the Internet; note that although network connections get faster all the time, it is probably still wise to connect to a site that is not too distant geographically from your location. Nowadays, the most convenient way to make an Internet connection is via a Web browser, especially since many TeX sites have developed a user-friendly interface to the CTAN archives.

B.1.1 Finding files on the archive

A first example of a Web interface is Peter Flynn's CTAN search engine <http://www.ucc.ie/cgi-bin/ctan> (see Fig. B.1). To the query entry ("pstricks" in our example) the search engine will return a list of all files in the CTAN archive matching the given search criterion. DANTE, the German TeX Users Group, offers a similar search engine at <http://www.dante.de/cgi-bin/ctan-index>. A simple interface to the directory structure is available <http://www.tex.ac.uk/tex-archive/>; this interface automatically displays a text of the "README" file if one is present at a given node.

A catalogue of TeX- and L^AT_EX-related packages maintained by Graham Williams can be consulted at <http://www.dit.csiro.au/~gjlw/texpkgs.html> (see Fig. B.2). This interface is especially attractive when you are "surfing" the archive and want to know what a certain package does. Moreover, the interface lets you choose the site from which to download the software, so that you can optimize your connection. When an extended description of the package is available on the Internet, its URL is presented together with the relevant entry. Several instances of this catalogue, which is regularly updated by its author, are available on the Internet.

These interfaces are very handy if you want to check the characteristics of one or more specific files (date of last modification, size, purpose etc.), but they are not particularly convenient for transferring complete packages consisting of many (sometimes hundreds of) files; more about this later.

B.2 Getting a package from the archive

Although in previous examples we used the Netscape browser, you can use any Web browser as an interface to the CTAN archives. In this section we use Microsoft's Internet Explorer under Microsoft Windows to go through the various steps of finding and transferring a complete package.

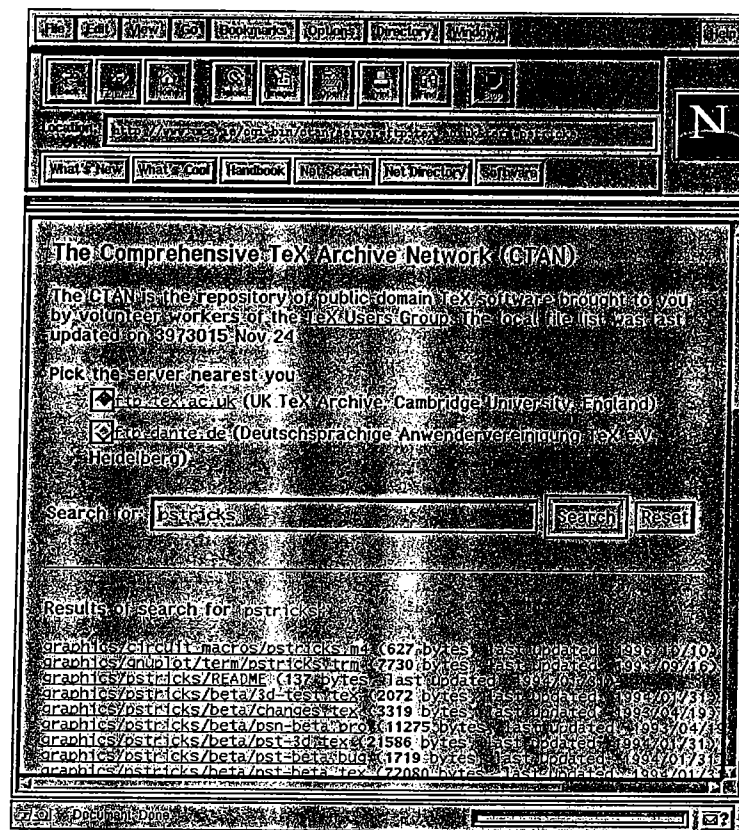


Figure B.1: Peter Flynn's CTAN Web interface

After connecting to the query interface of the WWW interface of the DANTE CTAN Internet node at <http://www.dante.de> we ask for filenames containing the string "graphicx" (shown inside an oval in the top part of Fig. B.3 on p. 501). The middle part of the figure shows the result of the search. We then select the entry shown with an arrow and the browser displays the file in question (bottom part of figure). You can then save it to disk.

If you want to transfer more than a single file (say, a complete package consisting of one or more complete directories), then you must position yourself in the directory one level

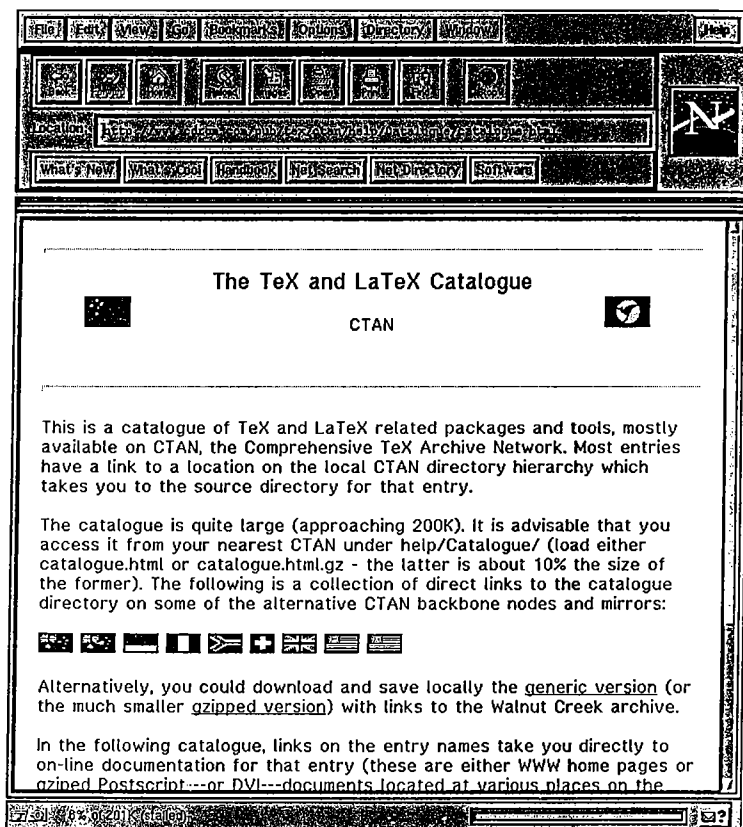


Figure B.2: Graham Williams' CTAN catalogue on the Web.

above the directory (tree) in question. In the example in Fig. B.4, you can type in the *Address* box containing the current URL the name of the directory with the extension `.zip` or `.tar.gz` appended (the ftp servers of the CTAN nodes support dynamic creation of archive files of type `.zip`, `.zoo`, `.tar`, and its gzip-compressed form). This transfers the desired files packed into an archive. The result for our example is a transfer as a zip archive of the complete directory containing all files associated with the graphics package. Thus at the top

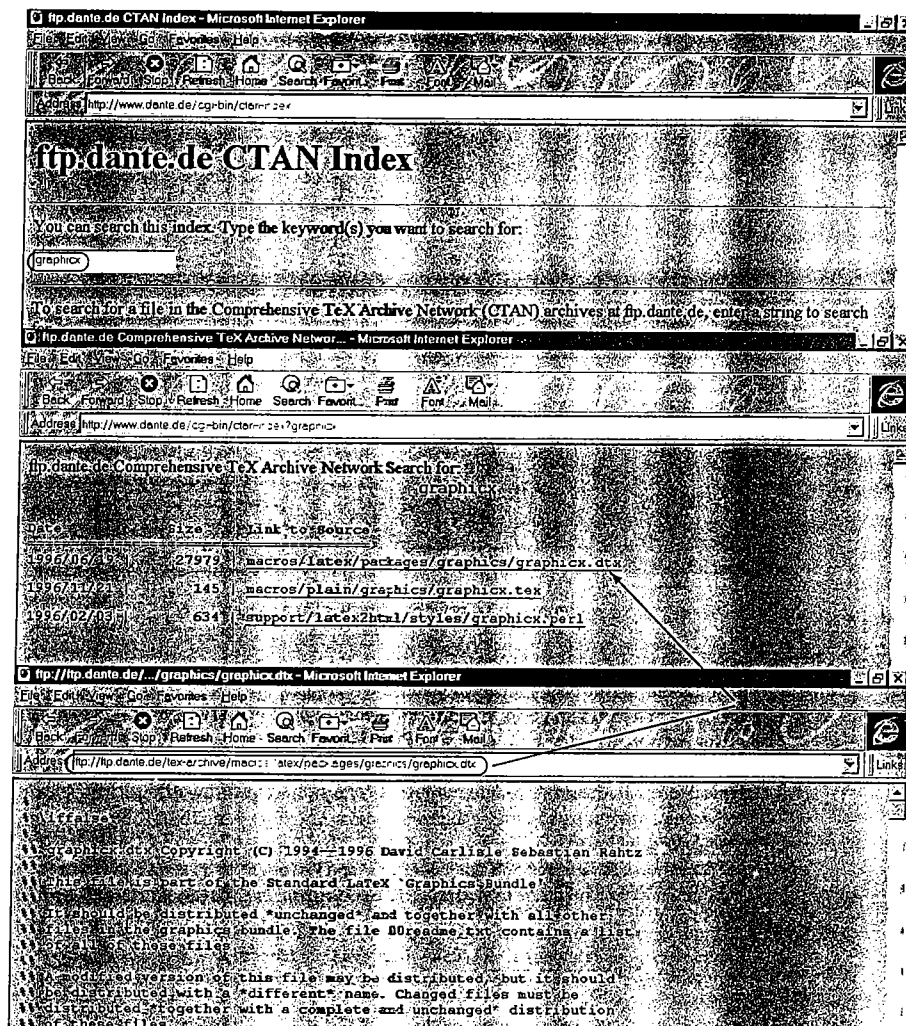


Figure B.3: Viewing CTAN files on the Web.

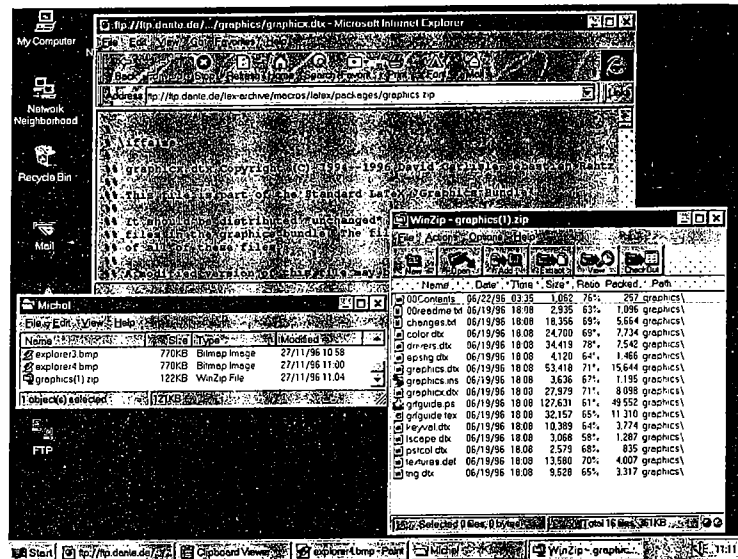


Figure B.4: Transfer of a directory as a zip archive.

of Fig. B.4 inside the oval, you see the command sent to the ftp server. At the bottom left you see the `graphics.zip` file in your directory, while at the bottom right you see the contents of the `.zip` archive after transfer to your PC.

You can also connect directly via ftp to any of the CTAN sites or mirrors. Below is a typical interactive session (commands input by the user are underlined>:

```
> ftp ftp.dante.de
Connected to sun.dante.de.
220 sun.dante.de FTP server ...
Name (ftp.dante.de:goossens): ftp
331 Guest login ok, send your complete e-mail address as password.
Password: goossens@cern.ch (use your email address!)
230-Welcome, archive user! This is an FTP server for the DANTE Archive.
...
ftp> quote site index graphicx
200-index graphicx
200-NOTE. This index shows at most 20 lines. for a full list of files,
200-retrieve /tex-archive/FILES.byname
```

```
200-1996/06/19 |      27979 | macros/latex/packages/graphics/graphicx.dtx
200-1996/11/21 |       145 | macros/plain/graphics/graphicx.tex
200-1996/02/03 |       634 | support/latex2html/styles/graphicx.perl
200 (end of 'index graphicx')
ftp> bir
200 Type set to I.
ftp> cd ctan:
250-Machine specific implementations --> systems
250-Original Knuthian sources: --> systems/knuth.
250-LaTeX styles, plain macros, MusicTeX: --> macros.
250-LaTeX2e: --> macros/latex.
...
ftp> cd macros/latex/packages
250 CWD command successful.
ftp> get graphics.zip
200 PORT command successful.
150 Opening BINARY mode data connection for /bin/ZIP.
226 Transfer complete.
124742 bytes received in 4.88 seconds (24.94 Kbytes/s)
ftp> get graphics.tar.gz
200 PORT command successful.
150 Opening BINARY mode data connection for /bin/tar.
226 Transfer complete.
117666 bytes received in 5.57 seconds (20.61 Kbytes/s)
ftp> quit
221 Goodbye.
```

After connecting to the site (`ftp.dante.de` above), you should specify `ftp` as login name and type *your* email address as password. Then you can send a query with the command “`query site index <term>`”, where *term* is the query string. We submitted the same query for the term “`graphicx`” as in Fig. B.3 above, and the result is, of course, identical in contents to the one in the middle part of that figure. We now decide to transfer the `graphics` package, so we first position ourselves at the root of the CTAN archive tree by issuing the command “`cd ctan:`” (with the colon at the end!). From our query we know where in the tree the files we want are located, so we change to the directory just one level above (`cd macros/latex/packages`). Then we transfer the directory twice, once as a `.zip` and once as a compressed `.tar` archive (just to show how to specify the commands). The command “`quit`” ends the ftp session.

B.3 List of packages and programs

In the following table we provide either a full Internet Uniform Resource Locator (URL), a location in the CTAN archives or an indication of which package contains the resource. The CTAN locations are prefixed with CTAN:

Table B.1: Location of the packages and programs mentioned in this book.

Name	Location
2up	part of seminar distribution
abc2mtex	CTAN:support/abc2mtex
Acrobat Distiller	http://www.adobe.com/prodindex/acrobat/main.html
Acrobat Reader	http://www.adobe.com/prodindex/acrobat/readstep.html
Adobe Acrobat	http://www.adobe.com/prodindex/acrobat/main.html
Adobe File Utilities	http://www.adobe.com/prodindex/fileutilities/main.html
Adobe Illustrator	http://www.adobe.com/prodindex/illustrator/main.html
Adobe Photoshop	http://www.adobe.com/prodindex/photoshop/main.html
Adobe Type Manager	http://www.adobe.com/prodindex/atm/main.html
afm2tfm	part of dvips distribution
aldratex	CTAN:macros/generic/dratex
aliphath	part of X _Y TeX package
amstex	CTAN:macros/amstex
ArcInfo	http://www.esri.com/base/products/arcinfo/arcinfo.html
array	CTAN:macros/latex/packages/tools
aurora	CTAN:support/aurora
AutoCAD	http://www.autodesk.com/products/autocad/autocad.htm
avant	part of PSNFSS package
	CTAN:fonts/psfonts/adobe/avantgar
	CTAN:fonts/psfonts/monotype/avantgar
axodraw	CTAN:graphics/axodraw
bakoma	CTAN:fonts/cm/type1/bakoma
bar	CTAN:macros/latex209/contrib/barkom
basker	part of PSNFSS package
	CTAN:fonts/psfonts/adobe/basker
bdfchess	CTAN:fonts/chess/bdfchess
bembo	part of PSNFSS package
	CTAN:fonts/psfonts/monotype/bembo
	CTAN:fonts/psfonts/xadobe/bembo
	CTAN:fonts/psfonts/xmonotype/bembo
bg	CTAN:macros/latex/contrib/other/bg
bit2spr	CTAN:graphics/bit2spr
bitpxl	CTAN:dviware/bitpxl
bm2font	CTAN:graphics/bm2font
bookman	part of PSNFSS package
	CTAN:fonts/psfonts/adobe/bookman

Location of the packages and programs mentioned in this book *continued*.

Name	Location
	CTAN:fonts/psfonts/monotype/bookman
boxes	part of METAPOST distribution
bridge	CTAN:macros/latex/contrib/other/bridge
carom	part of X _Y TeX package
cchess	CTAN:macros/latex/contrib/other/cchess
ccycle	part of X _Y TeX package
chancery	part of PSNFSS package
charter	part of PSNFSS package
	CTAN:fonts/charter
	CTAN:fonts/psfonts/bitstream/charter
ChemDraw	http://www.camsci.com/products/chemdraw/chemdraw.html
chemstr	part of X _Y TeX package
chemsym	CTAN:macros/latex/contrib/other/chemsym
chem	part of Unix Documentors Workbench
cheq	CTAN:fonts/cheq
chesskey	CTAN:fonts/chess/bdfchess
chess	CTAN:fonts/chess
circ	CTAN:macros/generic/diagrams/circ
CMacTeX	CTAN:systems/mac/cmactex
colordvi	CTAN:dviware/dvips
colortab	CTAN:macros/generic/misc
colortbl	CTAN:macros/latex/contrib/supported/carlisle
color	CTAN:macros/latex/packages/graphics
Corel Draw	http://www.corel.com/products/graphicsandpublishing/draw7/index.htm
crosswrd	CTAN:macros/latex/contrib/other/crosswrd
curves	CTAN:macros/latex/contrib/supported/curves
cwpuzzle	CTAN:macros/latex/contrib/other/gene/crossword
dcolumn	CTAN:macros/latex/packages/tools
draftcopy	CTAN:macros/latex/contrib/supported/draftcopy
dratex	CTAN:macros/generic/dratex
dvi2ps	CTAN:dviware/dvi2ps
dviaw	CTAN:drivers/beebe
dvicopy	part of T _E X distribution
dvidrv	part of emT _E X T _E X distribution
dvipsone	http://www.YandY.com/yydvips.htm
dvips	CTAN:dviware/dvips
dvitomp	part of METAPOST distribution
dvitops	CTAN:dviware/dvitops
dviwindo	http://www.YandY.com/yydviwin.htm
dviwin	CTAN:dviware/dviwin
ectree	CTAN:macros/latex209/contrib/ectrip

Location of the packages and programs mentioned in this book *continued*.

Name	Location
Edmetrics	http://www.bluesky.com/textures.html
eeepic	CTAN:macros/latex/contrib/other/eeepic
emTeX	CTAN:systems/msdos/emtex
epic	CTAN:macros/latex/contrib/other/epic/
epsffit	part of psutils set
epsfig	CTAN:macros/latex/packages/graphics
epstool	CTAN:support/ghostscript/rjl
Excel	http://www.microsoft.com/msexcel/
extractres	part of psutils set
fancybox	part of PSTricks distribution
Feynarts	http://www.mathsource.com/cgi-bin/MathSource/Applications/Physics/0202-194
feynman	CTAN:macros/latex209/contrib/feynman
feynmf	CTAN:macros/latex/contrib/supported/feynmf
feynmp	part of feynmf package
fig2mf	CTAN:graphics/fig2mf
fixdlsrps	part of psutils set
fixfmeps	part of psutils set
fixmacps	part of psutils set
fixpsditps	part of psutils set
fixpspps	part of psutils set
fixscribeps	part of psutils set
fixtpps	part of psutils set
fixwfwps	part of psutils set
fixwpps	part of psutils set
fixwwps	part of psutils set
flow	CTAN:support/flow
foiltex	CTAN:macros/latex/contrib/supported/foiltex
fontdoc	part of fontinst package
fontinst	CTAN:fonts/utilities/fontinst
FrameMaker	see http://www.adobe.com/prodindex/framemaker/main.html
FreeHand	http://www.macromedia.com/software/freehand/index.html
garamond	part of PSNFSS package
	CTAN:fonts/psfonts/adobe/garamond
	CTAN:fonts/psfonts/monotype/garamond
getafm	part of psutils set
gftodvi	part of T _E X distribution
gftopk	part of T _E X distribution
gftopxl	part of T _E X distribution
gftype	part of T _E X distribution
Ghostscript	CTAN:support/ghostscript/aladdin

Location of the packages and programs mentioned in this book *continued*.

Name	Location
ghostview	CTAN:support/ghostscript/gnu
gnuplot	CTAN:graphics/gnuplot
go	CTAN:fonts/go
gpik	part of GNU groff distribution
graphics	CTAN:macros/latex/packages/graphics
graphicx	part of graphics package
graph	part of METAPOST distribution
grap	part of Unix Documentors Workbench
gsftopk	CTAN:fonts/utilities/gsftopk
GSview	CTAN:support/ghostscript/rjl
hcycle	part of X _Y M _T E _X package
helvet	part of PSNFSS package
	CTAN:fonts/psfonts/adobe/helvetica
hetaromh	part of X _Y M _T E _X package
hetarom	part of X _Y M _T E _X package
hline	CTAN:macros/latex/packages/tools
Hijaak	http://www.csd.net/%7Eearcturus/xhijks.htm
hp2xx	CTAN:support/hp2xx
hptomf	CTAN:support/hptomf
ifthen	CTAN:macros/latex/base
Image Alchemy	http://www.alchemy.com.au/welcome.html
includeres	part of psutils set
inputenc	CTAN:macros/latex/base
Internet Explorer	http://www.microsoft.com/ie/ie.htm
keyval	part of graphics package
latex2html	CTAN:support/latex2html
lcircuit	CTAN:graphics/lcircuit
letter	CTAN:macros/latex/base
locant	part of X _Y M _T E _X package
longtable	CTAN:macros/latex/packages/tools
lowcycle	part of X _Y M _T E _X package
lpr	part of Unix systems
lscape	part of graphics package
lucbr	part of PSNFSS package
m-chemic	part of ppctex package
m4	part of Unix distribution
Mac GS Viewer	http://www.glyphic.com/glyphic/projects/macgs.html
makeindex	part of T _E X distribution
makeinstance	CTAN:fonts/utilities/mm
makempx	part of METAPOST distribution
MakeTeXPK	part of Web2c-based T _E X distributions

Location of the packages and programs mentioned in this book *continued*.

Name	Location
MakeTeXTFM	part of Web2c-based TeX distributions
Maple	http://www.maplesoft.on.ca/Products/MapleV/MapleV.html
Mathematica	http://www.mathematica.com
mathptm	part of PSNFSS package
MATLAB	http://www.mathworks.com
METAFONT	part of TeX distribution
METAPOST	part of TeX distribution, or CTAN:graphics/metapost
mfpic	CTAN:graphics/mfpic
mftoepts	CTAN:graphics/MF-PS/mftoepts
midi2tex	CTAN:support/midi2tex
mmafm	CTAN:fonts/utilities/mm
mpp	CTAN:support/mpp
mptotex	part of METAPOST distribution
mtimes	part of PSNFSS package
multido	part of PSTricks distribution
MusicTeX	CTAN:macros/musictex
music	part of Unix Documentors Workbench
musixflx	part of MusiXTeX package
MusiXTeX	CTAN:macros/musixtex/taupin
nassflow	CTAN:macros/latex209/contrib/nassflow
Netscape	http://www.netscape.com
newcent	part of PSNFSS package
nfssfont.tex	CTAN:macros/latex/base
nimbus	part of PSNFSS package
	CTAN:fonts/psfonts/urw/nimbus
OzTeX	CTAN:systems/mac/oztex
palatino	part of PSNFSS package
	CTAN:fonts/psfonts/adobe/palatino
pbmtpk	CTAN:graphics/pbmtpk
pctex32	http://www.pctex.com
pctexhp	http://www.pctex.com
pctexps	http://www.pctex.com
pctexwin	http://www.pctex.com
pdf2ps	part of Ghostscript distribution
pict2e	CTAN:macros/latex/base
PjCTeX	CTAN:graphics/pictex
pic	part of Unix Documentors Workbench
pie-chart.sh	part of PSTricks distribution
pifont	part of PSNFSS package
pktogf	part of TeX distribution
pktype	part of TeX distribution

Location of the packages and programs mentioned in this book *continued*.

Name	Location
pltotf	part of TeX distribution
PMX	CTAN:support/pmx
pnmcrop	part of Portable Bitmap utilities
ppchtex	CTAN:macros/context/ppchtex
ppmtogif	part of Portable Bitmap utilities
ps2ascii	part of Ghostscript distribution
ps2epsi	part of Ghostscript distribution
ps2pdf	part of Ghostscript distribution
ps2pk	CTAN:fonts/utilities/ps2pk
psbook	part of psutils set
PSfrag	CTAN:macros/latex/contrib/supported/psfrag
psmerge	part of psutils set
PSNFSS	CTAN:macros/latex/packages/psnfss
psnup	part of psutils set
pspicture	CTAN:macros/latex/contrib/other/misc/pspicture.sty
psprint	CTAN:dvipware/psprint
psresize	part of psutils set
psselect	part of psutils set
pst-3d	part of PSTricks distribution
pst-char	part of PSTricks distribution
pst-coil	part of PSTricks distribution
pst-eps	part of PSTricks distribution
pst-fill	part of PSTricks distribution
pst-grad	part of PSTricks distribution
pst-node	part of PSTricks distribution
pst-plot	part of PSTricks distribution
pst-text	part of PSTricks distribution
pst-tree	part of PSTricks distribution
pstcol	part of graphics package
pstops	part of psutils set
pstotext	http://www.research.digital.com/SRC/virtualpaper/pstotext.html
pstpoly	part of PSTricks distribution
pstricks	part of PSTricks distribution
PSTricks	CTAN:graphics/pstricks
psutils	CTAN:support/psutils
ps_view	CTAN:support/ps_view
pxltpk	part of TeX distribution
rboxes	part of METAPOST distribution
rotating	CTAN:macros/latex/contrib/supported/rotating
Scientific Word	http://www.thomson.com/tcisoft/products/swver25.html
semcolor	part of seminar distribution

Location of the packages and programs mentioned in this book *continued*.

Name	Location
seminar	CTAN:macros/latex/contrib/other/seminar
showchar	part of psutils set
slidesec	part of seminar distribution
slides	CTAN:macros/latex/base
Slit _{TeX}	CTAN:macros/latex209/distrib/latex/general
sprite	CTAN:graphics/bit2spr
SPSS	http://www.spss.com/
t1utils	CTAN:fonts/utilities/t1utils
teTeX	CTAN:systems/unix/teTeX
TeX-PS	CTAN:macros/generic/TeX-PS
TeXcad	part of emTeX distribution
Textures	http://www.bluesky.com/textures.html
tftopl	part of T _{EX} distribution
times	part of PSNFSS package CTAN:fonts/psfonts/adobe/times
timing	CTAN:macros/latex/contrib/other/timing
trees	CTAN:macros/latex/contrib/supported/treesvr
trig	part of graphics package
troff	part of Unix Documentors Workbench
TrueTeX	http://styx.ios.com/~kinch/truetex.htm
utopia	part of PSNFSS package CTAN:fonts/psfonts/adobe/utopia CTAN:fonts/psfonts/xadobe/utopia CTAN:fonts/utopia
vftovp	part of T _{EX} distribution
vptovf	part of T _{EX} distribution
VTeX	http://www.micropress-inc.com
Web2c	CTAN:systems/web2c
xdvi	CTAN:dviware/xdvi
xfig	CTAN:graphics/xfig
X _{AM} TeX	CTAN:macros/latex/contrib/other/xymtex
Xy-pic	CTAN:macros/generic/diagrams/xy-pic

Bibliography

- Adobe Systems 1990. *PostScript Language Reference Manual*. 2nd edition. Reading, MA: Addison-Wesley.
- Aho, A. V., Sethi, R. and Ullman, J. D. 1986. *Compilers: Principles, Techniques and Tools*. Reading, MA: Addison-Wesley.
- Appelt, W. 1988. Typesetting chess. *TUGboat*, 9 (3), 284–287.
- Arbortext Inc. 1986. *Manual for DVIPS*. (See notes on LPEXT distributed with the software).
- Bentley, J. 1986. Little languages. *Communications of the ACM*, 29, 711–721.
- Bentley, J. and Kernighan, B. 1984. *GRAP — a language for typesetting graphs*. Computing Science Technical Report 114, AT&T Bell Laboratories, Murray Hill, NJ.
- Brüggemann-Klein, A. and Wood, D. 1989. Drawing trees nicely with T_{EX}. *Electronic publishing — origin, dissemination and design*, 3, 101–115.
- Burbank, M. and Goossens, M. 1996. Electronic news from the family. *TUGboat*, 17 (1), 37–42.
- Carlisle, D. 1995. *Packages in the 'graphics' bundle*. Electronic document distributed with the graphics package.
- Clark, A. 1987. Halftone output from T_{EX}. *TUGboat*, 8 (3), 270–274.
- Clark, J. 1989. *DVITOPS user manual*. Electronic document distributed with the package.
- Clark, M. (ed.) 1990. *T_{EX} Applications, Uses, Methods*. Chichester: Ellis Horwood.
- Déribéré, M. 1993. *La Couleur — Collection: que sais-je ?, Nr. 220*. Eighth edition. Paris: Presses universitaires de France.

- Duggan, A. 1994. Colour separation and PostScript. *TUGboat*, 15 (3), 213–217.
- Duplan, P. and Jauneau, R. 1986. *Maquette et mise en page*. Paris: Éditions du Moniteur.
- Eck, H. and Küblbeck, S. 1991. *Generating Feynman graphs and amplitudes with FeynArts*. Technical report, Physikalisches Institut am Hubland, D-8700 Würzburg. Available from MathSource server at the URL <http://www.wolfram.com/mathsource/>.
- Foley, J. D., van Dam, A., Feiner, S. K. and Hughes, J. F. 1990. *Computer graphics, principles and practice*. 2nd edition. Reading, MA: Addison-Wesley.
- Fujita, S. 1993. *X_YTeX: a macro package for typesetting chemical structural formulas*. Manual of X_YTeX for Version 1.00. Electronic document distributed with the package.
- Fujita, S. 1995. X_YTeX for drawing chemical structural formulas. *TUGboat*, 16 (1), 80–88.
- Gerritsen, F. 1988. *Evolution in color*. West Chester, PA: Schiffer Publishing Ltd.
- Girou, D. 1994. Présentation de PSTricks. *Cahiers GUTenberg*, 16, 21–70.
- Goossens, M. and van Herwijnen, E. 1992. The elementary Particle Entity Notation (PEN) scheme. *TUGboat*, 13 (2), 201–207.
- Goossens, M., Mittelbach, F. and Samarin, A. 1994. *The L^AT_EX companion*. Reading, MA: Addison-Wesley.
- Greenwade, G. D. 1993. The Comprehensive T_EX Archive Network (CTAN). *TUGboat*, 14 (3), 342–351.
- Greenwood, T. G. 1993. International cultural differences in software. *Digital Technical Journal*, 5 (16), 8–20.
- Gurari, E. M. 1994. *T_EX and L^AT_EX: Drawing and Literate Programming*. New York: McGraw-Hill.
- Haas, R. T. and O’Kane, K. C. 1987. Typesetting chemical structure formulas with the text formatter T_EX/L^AT_EX. *Computers and Chemistry*, 11 (4), 251–271.
- Hafner, J. L. 1992. FoilT_EX, a L^AT_EX-like system for typesetting foils. *TUGboat*, 13 (3), 347–356.
- Hagen, J. and Otten, A. F. 1996. P_PCHT_EX: typesetting chemical formulas in T_EX. *TUGboat*, 17 (1), 54–66.
- Hamilton Kelly, B. 1990. Some macros to draw crosswords. *TUGboat*, 11 (1), 103–119.
- Haralambous, Y. and Rahtz, S. 1995. L^AT_EX, hypertext and PDF, or the entry of T_EX into the world of hypertext. *TUGboat*, 16 (2), 162–173.
- Hassel, F. 1993. Schachfigurensatz mit T_EX und L^AT_EX. *Die T_EXnische Komödie*, 5 (4), 11–25. An English translation is distributed as an electronic document with the package.
- Hobby, J. D. 1992. *A user’s manual for MetaPost*. Computing Science Technical Report 162,

- AT&T Bell Laboratories.
- Hobby, J. D. 1993. *Drawing graphs with MetaPost*. Computing Science Technical Report 164, AT&T Bell Laboratories.
- Hoenig, A. 1997. *T_EX Unbound: Strategies for Fonts, Graphics, and More*. New York: Oxford University Press.
- International Union of Pure and Applied Chemistry 1979. *Nomenclature of Organic Chemistry*. Oxford: Pergamon.
- International Union of Pure and Applied Physics 1987. Symbols, units, nomenclature and fundamental constants in physics. *Physica*, 146A, 1–67.
- Itten, J. 1974. *Art of Colour*. New York: Von Nostrand Reinhold.
- Jackowski, B. 1995. A METAFONT-EPS interface. *TUGboat*, 16 (4), 388–395.
- Jackson, R., MacDonald, L. and Freeman, K. 1994. *Computer Generated Colour*. New York: Wiley.
- Jalbert, F. 1989. *MuT_EX user’s guide*. Electronic document distributed with the package.
- Judd, D. B. and Wyszecki, G. 1963. *Color in Business, Science, and Industry*. 2nd edition. New York: Wiley.
- Karney, C. 1988. *Getting T_EX labelling on graphics*. Electronic mail to texhax, 24 May 1989.
- Kernighan, B. 1984. *PIC — a graphics language for typesetting*. Computing Science Technical Report 116, AT&T Bell Laboratories, Murray Hill, NJ.
- Knappen, J. 1992. Changing the appearance of math. In Zlatuška (1992), pp. 212–216.
- Knuth, D. E. 1986a. *The T_EXbook*, volume A of *Computers and Typesetting*. Reading, MA: Addison-Wesley.
- Knuth, D. E. 1986b. *T_EX: The Program*, volume B of *Computers and Typesetting*. Reading, MA: Addison-Wesley.
- Knuth, D. E. 1986c. *The METAFONTbook*, volume C of *Computers and Typesetting*. Reading, MA: Addison-Wesley.
- Knuth, D. E. 1986d. *METAFONT: The Program*, volume D of *Computers and Typesetting*. Reading, MA: Addison-Wesley.
- Knuth, D. E. 1986e. *Computer Modern Typefaces*, volume E of *Computers and Typesetting*. Reading, MA: Addison-Wesley.
- Knuth, D. E. 1987. Fonts for digital halftones. *TUGboat*, 8 (2), 135–160.
- Knuth, D. E. 1989. Typesetting *Concrete Mathematics*. *TUGboat*, 10 (1), 31–36.
- Knuth, D. E. 1990. Virtual fonts: more fun for grand wizards. *TUGboat*, 11 (1), 13–23.

- Knuth, D. E. 1993. Icons for $\text{\TeX}$ and METAFONT. *TUGboat*, 14 (4), 387–389.
- Kunkel, G. 1990. *Graphic Design with PostScript*. Glenview, IL: Scott, Foresman.
- Kuykens, H. J. P. and Verbruggen, A. H. 1992. *MIDI2 $\text{\TeX}$, a MIDI to Music $\text{\TeX}$ translation program*. Electronic document distributed with the package.
- Lamport, L. 1994. *L $\text{\TeX}$: A Document Preparation System*. 2nd edition. Reading, MA: Addison-Wesley.
- Levine, M. J. S. 1990. A $\text{\TeX}$ graphics routine for drawing Feynman diagrams. *Computer Physics Communications*, 58, 181–198.
- MacLenan, M. P. and Burns, G. M. 1991. An approach to drawing circuit diagrams for text books. *TUGboat*, 12 (1), 66–69.
- McGilton, H. and Campione, M. 1992. *PostScript by Example*. Reading, MA: Addison-Wesley.
- Merz, T. 1996. *PostScript & Acrobat/PDF*. Berlin: Springer Verlag.
- Murray, J. D. and van Ryper, W. 1996. *Encyclopedia of Graphics File Formats*. 2nd edition. Sebastopol, CA: O'Reilly & Associates, Inc.
- Nemcsics, A. 1993. *Colour Dynamics — Environmental Colour Design*. New York: Ellis Horwood.
- Nieuwenhuizen, J. and Nienhuys, H.-W. 1996. *MusiX $\text{\TeX}$ pre-processor—using $\text{\TeX}$ and the MusiX $\text{\TeX}$ macro package to write parts and scores of music*. Electronic document distributed with the package.
- Norris, A. C. and Oakley, A. L. 1990. Electronic publishing and chemical text processing. In Clark (1990).
- Ohl, T. 1995. Drawing Feynman Diagrams with $\text{\TeX}$ and METAFONT. *Computer Physics Communications*, 90, 340–354.
- Ohl, T. 1996. *feynMF, Drawing Feynman Diagrams with $\text{\TeX}$ and METAFONT*. Version 1.05. Electronic document distributed with the package.
- Olejniczak-Burkert, R. 1989. *texpic—design and implementation of a picture graphics language in $\text{\TeX}$* . *TUGboat*, 10 (4), 627–637.
- Podar, S. 1986. *Enhancements to the picture environment of $\text{\TeX}$* . Technical Report 86-17, Dept. of Computer Science, State University of New York, Stony Brook, NY. version 1.2.
- Rahtz, S. 1987. The Protestant Cemetery, Rome: a study undertaken under the auspices of the Unione Internazionale degli Istituti di Archeologia, Storia e Storia dell'Arte in Roma. *Opuscula Romana*, 16, 149–167.
- Ramek, M. 1990. Chemical structure formulae and x/y diagrams with $\text{\TeX}$. In Clark (1990).

- Reid, G. C. 1988. *PostScript Language Program Design*. Reading, MA: Addison-Wesley.
- Reid, T. and Hosek, D. 1989. Report from the DVI Driver Standards Committee. *TUGboat*, 10 (2), 188–191.
- Rogers, D. 1989. Computer graphics and $\text{\TeX}$ —a challenge. *TUGboat*, 10 (1), 39–44.
- Rokicki, T. 1994. Advanced 'special' support in a dvi driver. *TUGboat*, 15 (3), 205–212.
- Rose, K. H. 1992. How to typeset pretty diagram arrows with $\text{\TeX}$ —design decisions used in Xy-pic. In Zlatuška (1992), pp. 183–190.
- Rose, K. H. 1995. *Xy-pic user's guide. Version 3.0*. Electronic document distributed with the package.
- Rose, K. H. and Moore, R. 1995. *Xy-pic reference manual. Version 3.2*. Electronic document distributed with the package.
- Rubinstein, Z. 1989. Chess printing via METAFONT and $\text{\TeX}$. *TUGboat*, 10 (2), 170–172.
- Salmon, R. and Slater, M. 1987. *Computer Graphics — Systems & Concepts*. Amsterdam: Addison-Wesley Europe.
- Schofer, A. and Steinbach, A. 1987. *Automatisierter Notensatz mit $\text{\TeX}$* . Technical report, Rheinische Friedrich-Wilhelms-Universität, Bonn.
- Simons, D. 1995. *PMX, a preprocessor for MusiX $\text{\TeX}$. Version 0.92*. Electronic document distributed with the package.
- Sofka, M. 1994. Color book production using $\text{\TeX}$. *TUGboat*, 15 (3), 228–238.
- Sowa, F. 1994. Printing colour pictures. *TUGboat*, 15 (3), 223–227.
- Tannert, S. and Tille, A. 1996. *The CIRC package*. Electronic document distributed with the package.
- Taupin, D. 1992. Music $\text{\TeX}$: using $\text{\TeX}$ to write polyphonic or instrumental music. In Zlatuška (1992), pp. 257–272.
- Taupin, D. 1993a. Music $\text{\TeX}$: using $\text{\TeX}$ to write polyphonic or instrumental music. *TUGboat*, 14 (3), 203–211.
- Taupin, D. 1993b. Using $\text{\TeX}$ and METAFONT to build complicated maps. *TUGboat*, 14 (3), 196–202.
- Taupin, D. 1995a. MusiX $\text{\TeX}$, even more beautiful than Music $\text{\TeX}$ for music typesetting. In W. Dol (ed.), *Proceedings of the 9th European $\text{\TeX}$ Conference, September 4–8 1995, Arnhem, The Netherlands*, pp. 351–358. Nederlandstalige $\text{\TeX}$ Gebruikersgroep.
- Taupin, D. 1995b. *Music $\text{\TeX}$, using $\text{\TeX}$ to write polyphonic and instrumental music, Version 5.14*. Electronic document distributed with the package.
- Taupin, D., Mitchell, R. and Egler, A. 1996. *MusiX $\text{\TeX}$, using $\text{\TeX}$ to write polyphonic and*

- instrumental music, Version T.42.* Electronic document distributed with the package.
- TUG Working Group on the TeX Directory Structure (TWG-TDS) 1995. A Directory Structure for TeX Files (Version 0.999). *TUGboat*, 16 (4), 401–413.
- Tutelaers, P. 1992. A font and a style for typesetting chess using L^AT_EX or TeX. *TUGboat*, 13 (1), 85–90.
- Unicode Consortium 1996. *Unicode Standard; worldwide character encoding, version 2.0*. Reading, MA: Addison-Wesley.
- Valiente Feruglio, G. 1994. Typesetting commutative diagrams. *TUGboat*, 15 (4), 466–484.
- van der Laan, C. G. 1989. Typesetting bridge via L^AT_EX. *TUGboat*, 10 (1), 113–116.
- van Zandt, T. 1993a. *PSTricks user's guide*. Unpublished documentation with the software, version 0.93a.
- van Zandt, T. 1993b. *seminar.sty A E_T_X style for slides and notes. User's Guide. Version 1.0*. Unpublished documentation with the software.
- van Zandt, T. 1993c. *Documentation for colortab.tex/colortab.sty: shading and coloring TeX tables. Version 0.9*. Unpublished documentation with the software.
- van Zandt, T. and Girou, D. 1994. Inside PSTricks. *TUGboat*, 15 (3), 239–246.
- Vanderburg, G. L. and Reid, T. J. 1987. \special issues. *TUGboat*, 8 (2), 291–300.
- Vermaseren, J. A. M. 1994. Axodraw. *Computer Physics Communications*, 83, 45–58.
- Vulis, M. 1996. V_TE_X user guide. Manual supplied with V_TE_X software.
- Waldschmidt, H. 1988. An editor for constructing graphics with TeX. *Computer Graphics Forum*, 6 (4), 359–364.
- Walshaw, C. 1996. *ABC2M_TE_X, an easy way of transcribing folk and traditional music*. Electronic document distributed with the package.
- Wanske, H. 1990. Notenproduktion im Umbruch. Gedanken zur gegenwärtigen und zukünftigen Musikalienherstellung. In H.-J. Koppitz (ed.), *Gutenberg-Jahrbuch 1990*, pp. 237–243. Mainz, Germany: Gutenberg-Gesellschaft, Internationale Vereinigung für Geschichte und Gegenwart der Druckkunst e.V.
- White, J. V. 1990. *Color for the Electronic Age*. New York: Watson-Guption Publications.
- Wichura, M. J. 1987. *The P_TE_X manual*. Number 6 in TeXniques: publications for the TeX community. Providence, RI: TeX Users Group.
- Wichura, M. J. 1988. Macros for drawing P_TC_Tures. *TUGboat*, 9 (2), 193–197.
- Zlatuška, J. (ed.) 1992. *EuroTeX '92: Proceedings of the 7th European TeX Conference, Prague, Czechoslovakia, September 14–18, 1992*. Brno: Masarykova Universita.

Index

This index is in three parts; first we list all the packages and programs mentioned in the book, and all the commands, options and notations used in each (the private package therein refers to all commands defined in this book). Second, starting on p. 534, we provide a summary index of authors, packages, programs, files etc. Finally, starting on p. 539 is a complete list of all commands described in the book (entries therein without a package/program reference are directly available in L^AT_EX).

Throughout the index bold face page numbers are used to indicate pages with important information about the entry, e.g., the precise definition of a command or a detailed explanation, while page numbers in normal type indicate a textual reference.

- | | | |
|----------------------------|--|--|
| 2up package, 433, 504 | \ notation, 266, 268 | -o option, 269 |
| abc language, 254, 264–269 | {...} notation, 268 | -t option, 269 |
| " notation, 268 | _ notation, 267 | -x option, 269 |
| ' notation, 266 | _ _ notation, 267 | Acrobat Distiller program, 415, 455, 504 |
| (...) notation, 268 | notation, 265, 266, 267, 268 | Acrobat Reader program, 442, 504 |
| (2 notation, 267 | : notation, 266 | Adobe Acrobat program, 25, 372, 375, 406, 453, 504 |
| (3 notation, 267 |] notation, 265, 266, 267, 268 | Adobe File Utilities program, 20, 504 |
| (4 notation, 267 | notation, 266 | Adobe Illustrator program, 1, 4, 25, 68, 93, 94, 347, 442, 452, 504 |
| , notation, 266 | C: notation, 266 | Adobe Photoshop program, 347, 504 |
| - notation, 267, 268, 269 | D: notation, 266 | Adobe Type Manager program, 372, 375, 379, 380, 381, 388, 389, 406, 414, 447, 453, 504 |
| . notation, 268 | E: notation, 265, 267 | afm2tfm program, 291, 387, 390–393, 504 |
| / notation, 267, 268 | H: notation, 266 | -c option, 391 |
| :: notation, 266 | header . tex file, 265 | -e option, 390 |
| : notation, 266 | K: notation, 265, 266 | -O option, 391 |
| < notation, 267, 268 | L: notation, 266, 267 | -p option, 390, 391 |
| << notation, 267 | M: notation, 265, 266 | -s option, 390 |
| = notation, 267 | music . tex file, 269 | -T option, 390, 391 |
| > notation, 267, 268 | O: notation, 266 | -t option, 390, 391 |
| >> notation, 267 | S: notation, 266 | |
| [...] notation, 268 | T: notation, 266 | |
| [1 notation, 266 | X: notation, 265, 266 | |
| [2 notation, 266 | z notation, 266, 268 | |
| [notation, 266 | abc2mtx program, 254, 255, 264–269, 270, 504 | |
| ^ notation, 267, 268 | -i option, 269 | |
| ^^ notation, 267 | | |

-u option, 391
 -v option, 391
 * notation, 392
 =: notation, 392
 =: | notation, 392
 =: | > notation, 392
 > notation, 392
 {} notation, 392
 | notation, 392
 |= notation, 392
 |= > notation, 392
 |= | notation, 392
 |= | > notation, 392
 |= | >> notation, 392
 || notation, 392
 LTGKERN notation, 392
 aldratex package, 17, 504
 aliphat package, 210, 219, 504
 \dtetrastereo, 219, 220
 \dtrigonal, 219
 \dtrigonal, 219
 \ethanestereo, 219, 220
 \ethylene, 219
 \Ethylene, 219
 \ethylene, 219
 \ltrigona, 219
 \ltrigonal, 219
 \square, 219
 \tetrahedral, 219
 \tetrastereo, 219, 220
 \Utrigonal, 219
 \utrigonal, 219
 alltt package
 alltt environment, 12
 amstex package, 207, 504
 Sb environment, 207
 ArcInfo program, 25, 504
 array package, 326, 504
 \extrarowheight length, 327, 328
 \newcolumntype, 327
 aurora program, 349, 504
 AutoCAD program, 25, 83, 94, 152, 504
 avant package, 353, 354, 504
 axodraw package, 228, 504
 bakoma package, 342, 504
 bar package, 17, 78, 504
 basker package, 354, 504
 bdfchess package, *see also* chess package, 284–285, 288, 504
 \cardmessage, 286, 287
 chess15 font, 285
 gameone environment, 284, 285–287
 gametwo environment, 284, 285–287
 \postcard, 285, 286, 287
 \postcardaddress, 285, 286, 287
 \postmove, 284, 286, 287
 \postply, 284, 286, 287
 \receiver, 285, 286, 287
 \sender, 285, 286, 287
 \showboardwithnotation, 284
 \showinversboard, 284
 bembo package, 354, 504
 bg package, 299–302, 504
 \bigboard, 300
 \blackbar, 299, 300
 \blackcube, 300
 \blackonmove, 300
 \blackpoint, 299
 \boardcaption, 299, 300, 301
 \dontindentwhite, 302
 \dontshowcube, 300, 302
 \dontshowmoves, 302
 \dontshownumbers, 300
 \fullboard, 300
 \fullincr, 302
 game environment, 299, 300, 301
 \halfboard, 300
 \halfincr, 302
 \indentwhite, 302
 \middlecube, 299, 300
 \move, 300, 301, 302
 \normalboard, 300, 301
 position environment, 299, 300, 302
 \printboard, 300, 301
 \rawboard, 300, 301
 \showcube, 299, 300
 \showmoves, 302
 \shownumbers, 299, 300

 \smallboard, 299, 300
 \takecube, 301
 \textmove, 301
 \togglenumbers, 300
 \whitebar, 300
 \whitecube, 300
 \whiteonmove, 299, 300, 301
 \whitepoint, 299
 bit2spr program, 13, 504
 -height option, 13
 -width option, 13
 bitpxl program, 8, 504
 bm2font program, 8, 466, 467, 504
 bookman package, 353, 354, 504
 boxes METAPOST package, 68, 69, 72, 505
 boxit, 69, 70–72
 boxjoin, 69, 70–73
 circleit, 69, 71–73
 circmargin, 69, 73
 defaultdx, 69
 defaultdy, 69
 drawboxed, 69, 70–72
 drawboxes, 69, 71
 drawunboxed, 69, 70, 73
 pic, 69, 71, 73
 bridge package, 303, 505
 bidding environment, 303, 305
 bridge.tex file, 303
 \crdima, 303, 304
 \hand, 303–306
 carom package, 210, 214, 505
 \anthracenev, 214
 \bzdrh, 211, 213, 214
 \bzdrv, 211, 212, 214
 \cyclohexaneh, 213, 214
 \cyclohexanev, 212, 213, 214
 \decalineh, 214
 \decalinev, 214
 \hanthracenv, 214
 \hphenanthrenev, 214
 \naphdrh, 214
 \naphdrv, 214
 \phenanthrenev, 214
 \steroid, 214, 215
 \steroidchain, 214
 \tetralineh, 214

 \tetralinev, 214
 chess package, 293, 294, 505
 cchess46 font, 292
 cchessboard.tex file, 292
 \largeboard, 294
 \normalboard, 294
 \piece, 293, 294, 295
 position environment, 293, 294, 295
 \smallboard, 294, 295
 \textpiece, 293
 ccycle package, 210, 218, 505
 \adamantane, 218
 \bicycheph, 218
 \bicychepv, 218
 \bornane, 218
 \chair, 218
 chancery package, 353, 354, 505
 charter package, 354, 505
 chem program, 24, 505
 ChemDraw program, 25, 505
 chemstr package, 210, 505
 \lmoiety, 212, 215
 \rmoiety, 212
 chemsym package, 206, 207, 208, 209, 505
 ~ notation, 207
 _ notation, 207
 \CH, 207
 collision option, 208
 \COOH, 207
 \H, 207
 \h, 207
 \kemtkn, 207
 \O, 207
 \OH, 207
 \OO, 207
 pertab.tex file, 208
 \PP, 207
 \pr, 207
 \re, 207
 SB environment, 207
 \SS, 207
 cheq package, 288, 505
 Cheq font, 288
 cheq font, 289, 290, 393
 cheq.map file, 288

 \CheqBKing, 288
 chessboard.tex file, 288
 config.cheq file, 288
 xcheq font, 291
 xcheq.tfm file, 291
 xchess.enc file, 291
 chess package, 278–282, 285, 288, 289, 292, 294, 295, 505
 * notation, 278, 279, 282
 + notation, 282
 - notation, 282
 : notation, 282
 | notation, 278, 280, 282
 \bbetter, 282, 283
 \bdecisive, 282
 \betteris, 282, 283
 \Black, 279, 281
 \board, 278, 279, 292
 \bupperhand, 282, 283
 \Chess, 285, 289, 292
 chess environment, 282
 chess10 font, 278, 289–291
 chess20 font, 278
 chess30 font, 278
 chessf10 font, 278
 \equal, 283
 \move, 279, 280, 281, 284, 286
 movecount counter, 279, 281
 \newgame, 279, 280
 nochess environment, 282
 \ply, 279, 280, 281, 284
 position environment, 279, 281
 \seppawns, 282, 283
 \showboard, 279, 281, 284, 286, 292
 \wbetter, 282, 283
 \wdecisive, 282, 283
 \White, 279, 281
 \Whitefalse, 279
 \Whitetrue, 279, 281
 \withattack, 282, 283
 \wupperhand, 282, 283
 chesskey package, 285, 505
 \key, 285
 circ package, 243–249, 505
 \-, 246
 \., 245
 \A, 244, 247
 \at, 246
 \atpin, 247, 248
 basic option, 243, 245
 \BLens, 246, 248
 box option, 243
 \BSplit, 246, 248
 \bundle, 246
 \C, 244
 \Cam, 246, 248
 \cc, 245, 247, 248
 \Cel, 244
 \centerto, 247
 circuit environment, 245, 247, 248
 \connection, 247
 \Cvar, 244
 \D, 244
 \dashed, 246
 \Dcap, 244
 \DIFF, 245
 \Dtext, 248
 \fff, 245
 \from, 246
 \frompin, 247
 gate option, 244
 \GND, 247
 \HBLens, 246
 \HSLens, 246
 \htopin, 246, 247
 \HVLens, 246
 \I, 244
 ic option, 244
 \IvaR, 244
 \JKMSFF, 245
 \junction, 245
 \L, 244
 \La, 244
 \laser, 246, 248
 \LED, 244
 \ln, 245
 \Mirror, 248
 \moverel, 246, 247
 \mfet, 244
 \nl, 247, 248
 \npn, 244, 247
 \NRFSF, 245
 \nv, 245
 \nvmos, 244

\oa, 248
 \OM, 248
 optics option, 244
 \pfet, 244
 \Pinhole, 246, 248
 \PM, 246
 \pnp, 244
 \Polar, 246, 248
 \pvmos, 244
 \Q, 244
 \R, 244, 247
 \Rvar, 244
 \S, 244
 \ScrL, 246, 248
 \ScrR, 246
 \ScrTL, 246
 \ScrTR, 246
 \shift, 246, 247, 248
 \SLens, 246, 248
 \U, 244
 \Uvar, 244
 \V, 244
 \VLens, 246
 \vtopin, 246, 247
 \wire, 246
 \wwire, 246
 \ZD, 244
 CMacTeX program, 168, 505
 color package, 7, 27, 82, 100, 153, 311,
 317–326, 348, 424, 464, 505
 black notation, 318, 319
 blue notation, 318, 319
 cmyk color model, 317
 \color, 318, 319, 320, 321, 329,
 331, 332
 color.cfg file, 317
 \colorbox, 322, 324, 325, 330, 331,
 333, 335, 427
 cyan notation, 318, 319
 \definecolor, 317, 319, 320, 322,
 330, 331, 334, 337
 dvips option, 318
 dvipsnames option, 318
 \fbboxrule length, 322
 \fbboxsep length, 322, 335
 \fcolorbox, 322
 gray color model, 317
 green notation, 318, 319
 magenta notation, 318, 319
 monochrome option, 317
 named color model, 317, 318, 320
 nodvipsnames option, 318
 \pagecolor, 321, 345
 red notation, 318, 319
 rgb color model, 317
 \textcolor, 318, 319, 322
 usenames option, 318, 320, 344
 white notation, 318, 319
 yellow notation, 318, 319
 colordvi package, 317, 505
 colortab package, 326, 505
 colortbl package, 323, 326–338, 505
 \arrayrulecolor, 329, 332, 333,
 334, 336, 337
 \columncolor, 326, 327, 328,
 333–337
 \doublerulesepcolor, 329
 \rowcolor, 328, 334, 336, 337
 Corel Draw program, 1, 93, 94, 347,
 505
 crossword package, 306–308, 505
 \ACROSStext, 308
 \clue, 306, 307–309
 crossword environment, 306,
 307, 308, 310
 crossword* environment, 308,
 309
 \DOWNtext, 308
 curves package, 17, 505
 cwpuzzle package, 309, 505
 * notation, 309
 {} notation, 309
 \Clue, 310
 Puzzle environment, 309
 PuzzleClues environment, 310
 \PuzzleLetters, 310
 \PuzzleNumbers, 310
 \PuzzleSolution, 310
 \PuzzleUnsolved, 310
 PuzzleWords environment, 310
 \Word, 310
 dcolumn package, 326, 505
 dpc program, 250
 draftcopy package, 430, 505
 dratex package, 5, 17, 505
 dvi2ps program, 29, 414, 505
 dvialw program, 29, 505
 dvcopy program, 360, 505
 dvdrv program, 415, 505
 dvilaser/PS program, 29, 437
 dvipdf program, 29
 dvips program, xxiii, 11, 19, 28, 29, 41,
 65, 68, 154, 288, 291, 317, 318,
 320, 342, 348, 349, 354, 361,
 362, 369, 370, 386, 387, 390,
 396, 408, 409, 414, 415–433,
 455–458, 460, 463, 468, 469,
 505
 -A option, 416
 -a option, 416, 419
 -B option, 417
 -b option, 420
 -C option, 420
 -c option, 420
 -D option, 417
 -d option, 420, 432
 -E option, 416, 418, 456, 458
 -e option, 418
 -F option, 416, 417, 421
 -f option, 416, 417
 -h option, 418, 429
 -i option, 416, 419, 420, 457
 -j option, 416, 418
 -K option, 416, 421
 -k option, 416, 418
 -l option, 416
 -M option, 416, 418
 -m option, 416, 417
 -mode option, 418
 -N option, 416, 421, 429
 -n option, 416
 -O option, 417
 -o option, 416, 417
 -P option, 416, 419
 -p option, 416
 -pp option, 416
 -q option, 416, 420
 -R option, 420
 -r option, 416, 417
 -S option, 419, 420, 457
 -s option, 416, 421
 + notation, 422
 .dvipsrc file, 415
 < notation, 428
 = notation, 416
 @ notation, 422
 @ option, 417, 422
 % notation, 420, 421
 %b notation, 421
 %d notation, 421
 %f notation, 421
 %m notation, 421
 %p notation, 421
 ' notation, 41, 420
 a option, 419
 b option, 420
 color.pro file, 320, 427
 colorsep.pro file, 348, 468
 config.lino file, 424
 config.ps file, 288, 415, 419
 D option, 417
 dvips.ini file, 416
 E option, 420, 431
 e option, 418
 f option, 417
 H option, 420
 h option, 418
 I option, 419
 i option, 419
 j option, 418
 K option, 421
 M option, 418, 421
 m option, 419
 N option, 421
 O option, 417
 o option, 417
 P option, 420
 p option, 419, 429
 PRINTER environment variable,
 416, 417, 419
 psfonts.map file, 68, 354, 362,
 419, 428, 429
 q option, 420
 R option, 419
 r option, 417
 S option, 421
 s option, 421
 T option, 421
 U option, 421
 V option, 421
 W option, 420
 X option, 417
 Y option, 417
 Z option, 418
 dvipsone program, 11, 19, 28, 29, 318,
 414, 505
 dvitomp program, 66, 505
 dvitops program, 29, 414, 505
 dviwin program, 11, 29, 505
 dviwindo program, 29, 388, 414, 453,
 505
 ectree package, 126, 505
 Edmetrics program, 388, 506
 eepic package, 18, 20, 211, 212, 506
 emTeX program, 28, 29, 505, 506, 510
 epic package, 17, 18, 20, 208, 211, 212,
 506
 \dottedline, 211
 epsffit program, 437, 442, 458, 506
 epsfig package, 50, 506
 epstool program, 457, 506
 Excel program, 25, 506
 extractres program, 437, 441, 506
 fancybox package, 342, 506
 Feynarts package, 228
 Feynarts program, 506
 feynman package, 228, 506
 feynmf package, 84, 228–239, 506
 \fmf, 229, 232, 237–239
 \fmfblob, 234
 \fmfblobn, 234
 \fmfbottom, 232
 \fmfbottomn, 232
 \fmfcmd, 239
 \fmfcurved, 232
 \fmfcyclen, 232
 \fmfdot, 229, 234, 237, 238
 \fmfdotn, 234
 fmffile environment, 230
 \fmffixed, 237, 238
 \fmffreeze, 234, 238, 239
 fmfgraph environment, 236, 238
 fmfgraph* environment, 229,
 236, 237
 \fmfi, 238, 239
 \fmfiequ, 238
 \fmfipair, 238
 \fmfipath, 238
 \fmfiw, 238, 239
 \fmflabel, 237
 \fmfleft, 229, 232, 238
 \fmfleftn, 232, 236, 237
 \fmfn, 232
 \fmfpen, 234
 \fmfpoly, 234
 \fmfrcyclen, 232
 \fmfright, 229, 232, 238
 \fmfrightn, 232, 236, 237
 \fmfstraight, 232
 \fmfsurround, 232
 \fmftop, 232
 \fmftopn, 232
 \fmfv, 234
 \fmfvn, 234
 fmpict.n file, 231
 fmpict.mf file, 230
 fmpict.mp file, 231
 fmpict.tfm file, 230
 \fmfpoly, 234
 feynmp package, *see* feynmf package,
 84, 229, 230, 239, 506
 fig2mf program, 83, 506
 fixdlsrps program, 437, 506
 fixfmps program, 437, 506
 fixmacps program, 437, 506
 fixpsdits program, 437, 506
 fixpspps program, 437, 506
 fixscribes program, 437, 506
 fixtps program, 437, 506
 fixwfwps program, 437, 506

fixwpps program, 437, 506
 fixwpps program, 437, 506
 flow program, 18, 19, 506
 foitex package, 317, 338, 506
 fontdoc package, 487, 506
 fontinst package, 383, 387, 393–404,
 410, 412, 487–493, 506
 \add, 493
 \ascender, 489
 \botaccent, 398
 \boundarychar, 488
 \bye, 393
 \capheight, 489
 \codingscheme, 488
 \comment, 398
 \cscglyph, 398
 \declareencoding, 493
 \declaresize, 493
 \depth, 492
 \descender, 489
 \designsize, 489
 \div, 493
 \encoding, 487
 \endencoding, 487
 \endinstallfonts, 394, 400
 \endmetrics, 398, 488
 \endresetglyph, 398
 \endsetglyph, 397, 489
 \endsetslot, 398, 400, 488
 \endvarchar, 488
 \fontdimen(*n*), 488
 \fontinst, 492
 \fromafm, 395, 400
 \glyph, 397, 489
 \glyphrule, 490
 \glyphspecial, 490
 \glyphwarning, 490
 \height, 492
 hlcrima.mtx file, 402
 \iffalse, 491, 492
 \ifiscommand, 491
 \ifisdime, 492
 \ifisglyph, 492
 \ifisint, 492
 \ifisstr, 492
 \iftrue, 491, 492
 \inputetx, 488
 \inputmtx, 489
 \installfamily, 394, 399, 401
 \installfont, 394, 395, 396, 399,
 400, 402
 \installfonts, 394, 399
 \int, 397, 488, 492
 \italic, 493
 \italicslant, 489
 \kerning, 397, 493
 kernoff.mtx file, 402
 kernon.mtx file, 402
 latin.mtx file, 393, 396, 397, 399
 \latinfamily, 393, 394
 \lc, 399
 \lclig, 399
 \letterspacing, 488
 lhax.mtx file, 402
 \ligature, 398
 \lmathit.mtx file, 402
 \metrics, 396, 488
 \minimumkern, 489
 \monocyr.mtx file, 401
 \monoot2.mtx file, 400
 \monowidth, 489
 \mover, 397, 490
 \moveup, 490
 \mul, 493
 \needsfontinstversion, 492
 \neg, 493
 \nextlarger, 488
 \nextslot, 398, 400, 488
 \pop, 491
 \ptmr8r.mtx file, 393
 \push, 491
 \reencodefont, 395, 399, 400
 \relax, 487, 488
 \resetcommand, 492
 \resetdepth, 398, 491
 \resetdim, 492
 \resetglyph, 398, 489, 491
 \resetheight, 491
 \resetint, 492
 \resetitalic, 491
 \resetstr, 492
 \resetwidth, 491
 \scale, 493
 \scalefont, 395
 \setcommand, 492
 \setdim, 492
 \setglyph, 397, 489
 \setint, 396, 488, 492
 \setkern, 489
 \setleftkerning, 489
 \setlefttrighthkerning, 397
 \setrawglyph, 489
 \setrightkerning, 489
 \setslot, 398, 400, 488
 \setstr, 492
 \slantfont, 395
 \special, 490
 \sub, 493
 \substitutenoisy, 395, 399
 \substitutesilent, 395, 399
 t1.etx file, 393, 396
 \then, 491, 492
 \topaccent, 398
 \transformfont, 395, 399, 400
 \underlinethickness, 489
 \unfakable, 397, 401
 \unsetcommand, 492
 \unsetdim, 492
 \unsetglyph, 489
 \unsetint, 492
 \unsetluc.mtx file, 402
 \unsetstr, 492
 \usedas, 488
 \varbot, 488
 \varchar, 488
 \varmid, 488
 \varrep, 488
 \vartop, 488
 \width, 491, 492
 \xheight, 489
 \xscalefont, 395
 \yscalefont, 395
 fontinst program, 404
 FrameMaker program, 437, 452, 506
 FreeHand program, 1, 506
 garamond package, 354, 506
 getafm program, 437, 506
 gftodvi program, 363, 506
 gftopk program, 20, 230, 363, 364, 506
 gftopxl program, 364, 506
 gftype program, 363, 506

Ghostscript program, xxiii, 375, 379,
 408, 431, 442–453, 455,
 457–460, 494, 506, 508, 509
 -c option, 443
 -D option, 443
 -d option, 443–445
 -f option, 443
 -g option, 444
 -I option, 444
 -P option, 444
 -P- option, 444
 -q option, 444
 -r option, 444
 -S option, 444
 -s option, 444, 445
 -u option, 444
 - option, 444
 -- option, 443
 devices.txt file, 450
 Fontmap file, 445, 446, 447
 GS_DEVICE environment variable,
 448
 GS_FONTPATH environment
 variable, 446
 gs_init.ps file, 445
 GS_LIB environment variable, 446
 gs_statd.ps file, 445
 ps2ai.ps file, 452
 ps2ascii.ps file, 451, 452
 stdin file, 444
 stdout file, 449
 TEMP environment variable, 448
 unix-lpr.txt file, 449
 ghostview program, 449, 450, 451, 507
 gnuplot program, 20, 21, 83, 462, 507
 go package, 295–298, 507
 \black, 296, 297, 298
 \gofontsize, 296, 297, 298
 gosign50 font, 296
 \inidiagram, 296, 297
 \inifullldiagram, 296, 298
 \letter, 296, 297, 298
 \pos, 296, 297, 298
 \showdiagram, 297, 298
 \showfullldiagram, 297
 \square, 296, 298
 \symbol, 296
 \textblack, 297, 298
 \textwhite, 298
 \triangle, 296, 297, 298
 \white, 296, 297, 298
 gpic program, 18, 22, 250, 251, 507
 -c option, 22
 -t option, 22, 251
 \graph, 251
 grap language, 24–25
 copy, 24
 frame, 24, 25
 ht, 25
 label, 24
 thru, 24
 ticks, 24
 wid, 25
 grap program, 24, 72, 507
 graph METAPOST package, 72, 74,
 75, 78, 83, 507
 augment, 77, 78–80
 autogrid, 75, 76, 79, 81–83
 begingraph, 74
 endgraph, 74
 format, 75, 76
 frame, 75, 76, 77, 79–82
 gdata, 76, 77, 79, 81–83
 gdotlabel, 74
 gdraw, 72, 74, 76–78, 80–82
 gdrawarrow, 74
 gdrawdblarrow, 74
 gfill, 75–81, 83
 glabel, 72, 74, 78–83
 grid, 75
 init_numbers, 75
 itick, 75, 76
 otick, 75, 76, 82
 plot, 74
 setcoords, 76
 setrange, 76, 78, 79, 82, 83
 graphics package, 7, 9, 27–32, 34,
 39–45, 48–50, 51, 64, 99, 153,
 231, 424, 500, 503, 507, 509, 510
 \DeclareGraphicsExtensions,
 39, 40
 \DeclareGraphicsRule, 34, 39,
 40, 41
 draft option, 28
 dvi2ps option, 29
 dvialw option, 29
 dvilaser option, 29
 dvipdf option, 29
 dvips option, 29
 dvipsone option, 29
 dvitops option, 29
 dviwin option, 29
 dviwindo option, 29
 emtex option, 29
 final option, 28
 graphics.cfg file, 28
 \graphicspath, 39
 hiderotate option, 29
 hideshow option, 29
 hiresbb option, 28
 \includegraphics, 27, 29, 30, 31,
 39–41, 342, 461
 \includegraphics*, 30
 ln option, 29
 oztex option, 29
 pctex32 option, 29
 pctexhp option, 29
 pctexps option, 29
 pctexwin option, 29
 psprint option, 29
 pupps option, 29
 \reflectbox, 42
 \resizebox, 31, 34, 43, 44, 99
 \resizebox*, 43, 44
 \rotatebox, 27, 41, 44, 130
 \scalebox, 31, 34, 42
 tcidvi option, 29
 textures option, 29
 truetex option, 29
 graphicx package, *see also* graphics
 and keyval package, 27–29,
 32–51, 98, 101, 461, 507
 angle key, 33, 34, 36, 37
 bb key, 32, 33, 34
 bblx key, 32
 bblly key, 32
 bburx key, 32
 bbury key, 32
 clip key, 32, 33, 34
 command key, 33, 34
 depth key, 33
 draft key, 33, 34

ext key, 33, 34
 height key, 33, 35–37
 hiresbb key, 33
 \includegraphics, 32, 34, 38
 \includegraphics*, 32
 keepaspectratio key, 33, 35, 36, 38
 natheight key, 33
 natwidth key, 33
 orient key, 39
 origin key, 33, 46, 47
 read key, 33, 34, 39
 \rotatebox, 33, 34, 39, 46, 47
 scale key, 33, 34, 35
 \setkeys, 38, 47
 totalheight key, 33, 37, 38
 trim key, 33, 34
 type key, 33, 34, 41
 units key, 46, 47
 viewport key, 33, 34
 width key, 33, 35–38
 x key, 46, 47
 y key, 46, 47
 gsftopk program, 418, 450, 453, 455, 507
 GSview program, 449, 450, 457, 507
 gunzip program, 41
 gzip program, 500
 hcycle package, 210, 218, 507
 \furanose, 218
 \pyranose, 218
 helvet package, 353, 354, 507
 hetarom package, 210, 216, 217, 220, 507
 \benzofurane, 217
 \benzofurane, 217
 \benzoxazole, 217
 \benzoxazole, 217
 \cinnoline, 217
 \cinnoline, 217
 \decahetero, 217
 \fivehetero, 216
 \fiveunitv, 220
 \fiveunitvi, 220
 \fourhetero, 216
 \imidazole, 217

\imidazole, 217
 \indole, 217
 \indole, 217
 \indolizine, 217
 \indolizine, 217
 \isobenzofurane, 217
 \isobenzofurane, 217
 \isoindole, 210, 217
 \isoindole, 217
 \isoquinoline, 217
 \isoquinoline, 217
 \isoxazole, 217
 \isoxazole, 217
 \nonahetero, 210, 217
 \oxazole, 217
 \oxazole, 217
 \pteridine, 217
 \pteridine, 217
 \purine, 210, 217
 \purine, 217
 \pyrazine, 214, 217
 \pyrazole, 217
 \pyrazole, 217
 \pyridazine, 217
 \pyridazine, 217
 \pyridine, 217, 220
 \pyridine, 217
 \pyrimidine, 217
 \pyrimidine, 217
 \pyrrole, 217
 \pyrrole, 217
 \quinazoline, 217
 \quinazoline, 217
 \quinoline, 217
 \quinoline, 217
 \quinoxaline, 217
 \sixhetero, 213, 216
 \sixunitv, 220
 \threehetero, 213, 216
 \triazine, 217
 \triazine, 217
 hetaromh package, *see* hetarom
 package for command names,
 210, 216, 220, 507
 hhlne package, 326, 329, 336, 418, 507
 \hhlne, 336
 Hijaak program, 20, 507
 hp2xx program, 20, 507

hptomf program, 20, 507
 idraw program, 1, 462
 ifthen package, 92, 198, 324, 325, 507
 \ifthenelse, 198
 \while, 113, 324, 325
 Image Alchemy program, 20, 507
 includeres program, 437, 441, 507
 inputenc package, 362, 507
 Internet Explorer program, 498, 507
 keyval package, 38, 51, 507
 \setkeys, 38
 latex2html program, 457, 507
 lcircuit package, 18, 507
 locant package, 210, 507
 longtable package, 208, 326, 329, 507
 longtable environment, 326
 lowcycle package, 210, 215, 507
 \cyclobutane, 216
 \cyclopentane, 215
 \cyclopentane, 215, 216
 \cyclopentane, 215
 \cyclopropane, 216
 \indane, 216
 \indane, 216
 lpr program, 449, 507
 lscape package, 51, 507
 landscape environment, 51
 lucbr package, 354, 507
 expert option, 354
 yy option, 354
 m-chemic package, 221–228, 507
 + notation, 225
 – notation, 225
 ADJ notation, 225, 226
 B notation, 223, 225
 C notation, 225
 CARBON notation, 222, 223
 CB notation, 222
 \chemical, 222, 223, 224–227, 228
 CRZ notation, 227
 DB notation, 225
 \definechemical, 224
 DR notation, 225
 EB notation, 225

EQUILIBRIUM notation, 223, 227
 ER notation, 225
 FIVE notation, 223
 FOUR notation, 223
 GIVES notation, 227
 MOV notation, 225
 NEWMAN notation, 223
 OFF notation, 227
 ONE notation, 223, 227
 PLUS notation, 227
 R notation, 223, 225
 ROT notation, 225, 226
 RZ notation, 223, 224, 225
 S notation, 225
 SB notation, 225
 \setupchemical, 222, 227
 SIX notation, 223
 SPACE notation, 227
 SR notation, 225
 \startchemical, 222, 223, 224, 226, 227
 \stopchemical, 222, 223, 224, 226, 227
 SUB notation, 225, 226
 TB notation, 225
 THREE notation, 223
 Z notation, 225
 ZO notation, 225, 227
 m4 program, 250, 251, 507
 Mac GS Viewer program, 450, 507
 MacPaint program, 3, 4
 makeindex program, 84, 507
 makeinstance program, 408, 507
 makempx program, 66, 507
 MakeTeXPK program, 415, 455, 507
 MakeTeXTFM program, 410, 508
 Maple program, 1, 508
 Mathematica program, 1, 25, 508
 mathptm package, 68, 353, 354, 356, 357, 401, 508
 MATLAB program, 1, 462, 508
 META language, 25, 53–83
 ++ notation, 55
 +-+ notation, 55
 -- notation, 56
 .. notation, 55, 56

abs, 57
 angle, 55
 beginchar, 60, 63
 boolean, 54
 clip, 57
 cosd, 55
 curl, 56
 currentpicture, 66
 cycle, 56, 57, 77–80
 dir, 56, 70–73
 direction, 56
 draw, 55, 56, 57, 70, 74, 81
 end, 63
 endchar, 60, 63
 endfor, 55, 57
 fill, 55, 70, 74
 floor, 55
 for, 55, 57
 fullcircle, 67, 81
 length, 55, 71, 73
 makepen, 54
 mode, 50
 mode_setup, 61
 normaldeviate, 55
 nullpen, 54
 numeric, 54
 origin, 76, 77
 pair, 54, 57, 64
 path, 54, 56, 57
 pen, 54
 pencircle, 54, 57, 73, 78
 pensquare, 82
 pickup, 57, 73, 78, 82
 picture, 54, 66, 67
 point, 71, 72, 73
 reflectedabout, 66
 rotated, 56, 57, 67, 78–81
 rotatedabout, 66
 rotatedaround, 57
 scaled, 57, 66, 67, 73, 78, 79, 81, 82
 shifted, 57, 66
 sind, 55
 sqrt, 55
 step, 57
 string, 54
 tension, 56, 71–73
 transform, 54

true, 57
 uniformdeviate, 55
 until, 57
 upto, 57
 vardef, 71, 72
 whatever, 76, 78, 82
 withpen, 74
 xpart, 54
 ypart, 54
 yscaled, 57
 METAPOST language, *see also*
 META language
 bbox, 65, 66, 79, 81
 beginfig, 63
 black, 64
 blue, 64
 bot notation, 64
 bpath, 70, 71, 73
 btex, 64, 65, 66, 67, 72, 74, 75, 78, 80
 buildcycle, 81
 color, 64, 73
 cutafter, 70, 71–73
 cutbefore, 70, 71–73
 dashed, 72, 73, 74, 78
 defaultfont, 64, 65, 73, 79, 81
 defaultscale, 64, 66, 72, 73, 79, 81–83
 dotlabel, 64, 65
 dotlabels, 65, 66
 drawarrow, 70, 71–73
 drawdblarrow, 70
 endfig, 63
 etex, 64, 65–67, 72, 74, 75, 78, 80
 green, 64
 image, 79, 81
 infont, 79, 81
 input, 68
 label, 64, 65, 71, 75
 labelarrow, 72
 labeloffset, 64, 65
 lft notation, 64
 llft notation, 64
 lrt notation, 64
 OUT notation, 74
 prologues, 67, 68
 red, 64
 rt notation, 64

scantokens, 77, 78–83
 thelabel, 65, 66, 67
 top notation, 64
 ulft notation, 64
 unfill, 57, 79, 81
 urt notation, 64
 verbatimex, 66, 67
 white, 64
 withcolor, 66, 67, 73, 74, 75–79, 81, 83
 withdots, 78
 METAPOST program, 66
 -T option, 66
 TEX environment variable, 66, 67
 mpic package, 25, 54, 84–92, 250, 508
 \arc, 86
 \arrow, 86, 88, 91
 \axes, 86, 87, 90
 \axisheadlen length, 86, 87
 \bclosed, 86, 88
 \btwnfcn, 89
 \cbclosed, 88
 \circle, 86
 \closegraphsfile, 85
 connect environment, 85, 88, 89
 coords environment, 92
 \curve, 86, 92
 \cyclic, 86
 \darkershade, 88
 \dashlen length, 87, 88
 \dashlineset, 88
 \dashspace length, 87, 88
 \dotlineset, 88
 \dotted, 86, 88
 \draw, 88, 90
 \ellipse, 87, 92
 \function, 89, 90
 \gclear, 88, 90
 \gfill, 86, 88, 90
 \hashlen length, 87
 \hatch, 87, 88
 \hatchspace length, 87, 88
 \headlen length, 87, 88
 \headshape, 88
 \lclosed, 88
 \lightershade, 88
 \lines, 86, 87, 91
 mpic environment, 85, 91
 \mpicunit length, 85, 87
 \opengraphsfile, 85
 \parafcn, 89, 92
 \pen, 86, 88, 90
 \plrfcn, 89
 \plrregion, 89
 \point, 87
 pointfilled boolean, 87
 \pointsize length, 87
 \polygon, 87
 \polylines, 87
 \rect, 87
 \reverse, 88
 \rotatepath, 89
 \sclosed, 88
 \sector, 87
 \setrender, 85
 \shade, 86, 87, 89
 \shadespace length, 87–89
 \tcaption, 90, 91
 \tlabel, 89, 91
 \turn, 90, 92
 \turtle, 87
 \whiledo, 92
 \xmarks, 87
 \ymarks, 87
 mftoops METAFONT package, 93, 508
 draw, 93
 draw_C, 93, 94
 EPS_mode_setup, 93
 fill, 93
 fill_C, 93, 94
 find_BB, 93
 fix_drawn_cmyk, 94
 fix_fill_cmyk, 94
 fix_line_width, 94
 mftoops.mf file, 93
 set_BB, 93
 write_postamble, 93
 write_preamble, 93
 Microsoft Word program, 437
 midi2tex program, 254, 255, 274, 275, 508
 -d option, 275
 -i option, 275
 -o option, 275
 -q option, 275

-s option, 275
 mmafm program, 408, 508
 MP language, 269–274
 ' notation, 269, 270, 272, 273
 (...) notation, 271, 272, 273
 + notation, 272
 , notation, 270
 - notation, 272
 / notation, 272
 : | notation, 270
 : | : notation, 270
 : | | : notation, 270
 [...] notation, 271, 272, 273
 [...] / notation, 272
 [...] notation, 272
 [v...] notation, 272
 ~ notation, 271, 273
 { notation, 273
 {...} notation, 271, 272
 } notation, 273
 _ notation, 271, 272
 ' notation, 269
 | notation, 270
 | . notation, 270–273
 || notation, 270
 ||| notation, 270
 \A, 270
 \alto, 269, 272
 \bass, 270, 273
 \Bes, 271–273
 bes notation, 270
 \clef, 269, 272, 273
 \downbow, 271
 \duration, 270, 271–273
 \F, 272
 gis notation, 270
 \grace, 271, 272
 \headleft, 271
 \headright, 271, 272, 273
 \key, 270, 271–273
 \meter, 272, 273
 \normal, 271
 r notation, 271, 272
 s notation, 271, 272
 \small, 271
 \stemdown, 271
 \stemno, 271
 \stemup, 271, 272

\tenor, 269
 \tiny, 271
 \tr, 271
 \treble, 269, 272
 \upbow, 271
 \xpletstyle, 272
 mpp program, 273, 508
 mptotex program, 66, 508
 mtimes package, 354, 508
 multido package, 139, 508
 \multido, 121, 139, 146
 music program, 24, 508
 MusicTeX package, *see also* MusixTeX
 package, 254, 263, 269, 274, 275, 508
 \elemskip length, 275
 musixflx program, 260, 263, 264, 508
 MusixTeX package, xxii–xxiv, 253, 254–264, 265, 267, 269, 273, 274, 508
 ' notation, 257
 ' ' notation, 257
 ' ' ' notation, 257
 . notation, 259
 < notation, 257
 = notation, 257
 > notation, 257
 & notation, 256, 261
 ~ notation, 257, 258
 _ notation, 257, 258
 ' notation, 257
 | notation, 256, 261
 \allabreve, 257
 \altoclef, 257
 \arpeggio, 257
 \backturn, 257
 \bar, 256, 259–261
 \bass, 261
 \bassclef, 257
 \boxit, 257
 \bracket, 257
 \ca, 258, 259, 260
 \caesura, 257
 \cbreath, 257
 \cca, 258, 259, 260
 \cccl, 257, 259
 \cccu, 257, 259
 \cccl, 257, 259
 \cccu, 257, 259
 \ccc, 257, 258
 \cccu, 257, 258
 \ccl, 257, 258, 259, 260
 \ccu, 257, 258, 259, 260
 \cdf, 257
 \cdsh, 257
 \cfl, 257
 \chl, 257
 \chu, 257
 \circleit, 257
 \cl, 257, 258
 \cna, 257
 \cql, 257
 \cqu, 257
 \csh, 257
 \cu, 257, 258, 259, 260
 \dcqu, 257
 \DEP, 257
 \dhqu, 257
 \doqu, 257
 \downbow, 257
 \downtrio, 257
 \dqu, 257
 \drumclef, 257
 \ds, 257, 259
 \duevolte, 257
 \elemskip length, 260, 263, 265, 267
 \endextract, 259, 261
 \endpiece, 259
 \enotes, 256, 259–261
 \fermatadown, 257
 \Fermataup, 257
 \fermataup, 257
 \fla, 258
 \flageolet, 257
 \generalmeter, 261
 \generalsignature, 258, 261
 \geometricsskip, 260
 \grcl, 257
 \grcu, 257
 \gregorianCclef, 257
 \gregorianFclef, 257
 \ha, 258
 \hl, 257, 258
 \Hpause, 257
 \hpause, 257, 259
 \hpausep, 257
 \hs, 257
 \hu, 257, 258, 259
 \ibbu, 262
 \ibl, 261, 262
 \ibu, 261, 262
 \instrumentnumber, 261
 \islurd, 262
 \isluru, 261, 262
 \kqu, 257
 \l, 257
 \l..., 259
 \leftrepeat, 257
 \leftrightrepeat, 257
 \lifthpause, 257
 \liftpause, 257
 \lppz, 257
 \lpz, 257
 \lpzst, 257
 \lsf, 257
 \lsfz, 257
 \lsqu, 257
 \lst, 257
 \meterC, 257
 \meterfrac, 261
 \meterplus, 257
 \metron, 257
 \Mordent, 257
 \mordent, 257
 music environment, 259, 260, 261
 \NOTes, 260
 \NOTes, 260
 \NOTes, 256, 260, 261
 \Notes, 256, 259, 260, 261
 \notes, 256, 260, 261
 \noteskip length, 260
 \NOTesp, 260
 \NOTesp, 260
 \Notesp, 260
 \notesp, 260
 \octfindown, 257
 \octfinup, 257
 \oldGclef, 257
 \oxqu, 257
 \PAUSE, 257
 \PAuse, 257
 \pause, 257, 259
 \pausep, 257
 \PED, 257

\pt, 259
 \qa, 258, 259, 260
 \qb, 261, 262
 \ql, 257, 258, 261, 262
 \qp, 257, 259
 \qqs, 257
 \qs, 257
 \qu, 257, 258, 259–262
 \qupp, 257
 \r..., 259
 \reverseallabreve, 257
 \reverseC, 257
 \rh, 259
 \rightrepeat, 257
 \roqu, 257
 \rq, 261
 \rsqu, 257
 \sDEP, 257
 \segnos, 257
 \setclef, 261
 \setstafes, 261
 \setvolta, 257
 \setvoltabox, 257
 \sh, 258
 \Shake, 257
 \shake, 257
 \Shakel, 257
 \Shakene, 257
 \Shakenw, 257
 \Shakesw, 257
 \sk, 260
 \slide, 257
 \smallaltoclef, 257
 \smallbassclef, 257
 \smallmusicsize, 261
 \smalltrebleclef, 257
 \sPED, 257
 \squ, 257
 \startextract, 259, 261
 \startpiece, 259
 \tbl, 261, 262
 \tbu, 261, 262
 \tgqu, 257
 \treble, 261
 \trebleclef, 257
 \Trille, 257
 \trille, 257
 \tslur, 261, 262
 \turn, 257
 \upbow, 257
 \uppz, 257
 \uptrio, 257
 \upz, 257
 \upzst, 257
 \usf, 257
 \usfz, 257
 \ust, 257
 \wh, 257, 258, 259
 \whp, 257
 \xqu, 257
 \yqu, 257
 \z..., 259
 \zbrev, 257
 \zh, 259
 \zlonga, 257
 \zmaxima, 257
 \znotes, 260
 \zq, 261
 \zqb, 261
 \zw, 259
 \zwq, 257
 nassflow package, 18, 508
 Netscape program, 498, 508
 newcent package, 353, 354, 508
 nfssfont.tex program, 289, 508
 nimbus package, 354, 508
 OzTeX program, 29, 414, 508
 palatino package, 353, 354, 508
 pbmtopk program, 8, 508
 pctex32 program, 29, 508
 pctexhp program, 29, 508
 pctexps program, 29, 318, 508
 pctexwin program, 29, 508
 pdf2ps program, 453, 508
 pic language, 22–23, 24, 26, 68, 150
 "..." notation, 22, 23
 .PE notation, 22, 250
 .PS notation, 22, 250
 .c notation, 23
 .n notation, 23
 .ne notation, 23
 .nw notation, 23

.se notation, 23
 .sw notation, 23
 ; notation, 23
 arc, 22
 arrow, 22
 atan, 22
 box, 22, 23
 circle, 22, 23
 cos, 22
 dashed, 23
 define, 23
 down, 22, 23
 ellipse, 22
 from, 23
 height, 23
 i notation, 22
 invis, 23
 left, 23
 libccf.m4 file, 251
 line, 22, 23
 move, 23
 rand, 22
 rect, 23
 right, 22, 23
 sin, 22
 spline, 22
 sqrt, 22
 then, 22
 to, 23
 width, 23
 pic program, 20, 22–24, 184, 250–252, 508
 pict2e package, 17, 508
 PCTeX package, 5, 14–16, 26, 222, 508
 pie-chart.sh program, 134, 508
 pifont package, 323, 353, 356, 508
 \ding, 323
 Piautolist environment, 358
 \Pifill, 357
 \Piline, 357
 Pilist environment, 358
 \Pisymbol, 357, 359
 pktogf program, 364, 508
 pkttype program, 364, 508
 PL/VPL language, 364–370
 CHARACTER, 367, 368
 CHECKSUM, 368

COMMENT, 365, 367
 DESIGNSIZE, 366
 FONTAT, 368
 FONTDIMEN, 365
 FONTSIZE, 368
 FONTNAME, 368
 KRn, 366
 LABEL, 366
 LIG, 366, 392
 LIGTABLE, 366, 367
 MAP, 368
 MAPFONT, 368
 MOVERIGHT, 368
 POP, 368
 PUSH, 368
 SPACE, 365
 SPECIAL, 369
 STOP, 366
 VTITLE, 368
 pltotf program, 365, 509
 PMX program, 255, 509
 pnmcrop program, 457, 509
 PostScript language
 .notdef, 291, 392
 #copies, 420
 bind, 437, 445
 BlendDesignMap, 406
 BlendDesignPositions, 406
 bop-hook, 420, 429, 431
 BoundingBox, 28, 30, 33, 40, 41, 63, 456
 CharStrings, 370
 CIEBasedA, 445
 CIEBasedABC, 445
 colorimage, 349
 ConvertDesignVector, 406, 409
 deletefile, 445
 DeviceGray, 445
 DeviceRGB, 445
 Encoding, 371
 end-hook, 429, 433
 endmulti, 433
 eop-hook, 429
 ExtendFont, 370, 428
 FontInfo, 373
 FullName, 371
 HiResBoundingBox, 28, 33

multi, 433
 NormalizeDesignVector, 406, 409
 pdfmark, 415
 quit, 443
 ReEncodeFont, 291, 370, 428
 renamefile, 445
 Resolution, 426
 scale, 431
 SDict, 426
 selectdevice, 448
 setgray, 345
 setpagedevice, 431, 458
 showpage, 433, 442, 443
 SlantFont, 428
 StandardEncoding, 371
 start-hook, 429, 433
 systemdict, 443, 444
 TeXBase1Encoding, 361, 369, 388, 428
 translate, 431
 userdict, 418, 425, 429, 443
 ppctex package, *see* m-chemic package, 221–228, 507, 509
 ppmtofig program, 457, 509
 private package
 \Battery, 150
 \bid, 305, 306
 \Bpara, 49, 50
 \circuit, 186, 187
 \club, 302, 304–306
 \crdxa, 305
 \diam, 302, 305
 \Down, 150, 151
 \gosign, 296
 \hand, 302
 \heart, 302, 305, 306
 \HR, 30, 43
 \Inductor, 150
 \Left, 150
 mainline environment, 282, 283
 \mathbb, 404
 \pictex.sty file, 16
 \ProcessVector, 152
 \renewfont, 289
 \Resistor, 150
 \Right, 150
 \showgrid, 98, 142
 \slidechapter, 344
 \SlideColors, 345
 \SlideFront, 344
 \spade, 302, 304–306
 \Switch, 150
 \Up, 150
 variation environment, 282, 283
 w.eps file, 30
 psascii program, 452, 509
 ps2epsi program, 452, 509
 ps2pdf program, 453, 509
 ps2pk program, 418, 453–455, 509
 TINPUTS environment variable, 455
 ps_view program, 450, 509
 psbook program, 437, 439, 509
 PSfrag package, 26, 460–462, 509
 \psfrag, 460, 461
 psmerge program, 437, 440, 509
 PSNFS package, 353–356, 359–362, 381, 386, 394, 504–508, 509, 510
 psnp program, 435, 437, 438, 439, 509
 pspicture package, 17, 48, 509
 psprint program, 29, 437, 509
 psresize program, 437, 440, 509
 psselect program, 437, 440, 509
 pst-3d package, 96, 166, 509
 embedangle key, 166
 normal key, 136, 137, 166
 \psshadow, 136, 166
 \pstilt, 136, 166
 \ThreeDput, 136, 137, 138, 166
 Tshadowangle key, 136, 166
 Tshadowcolor key, 166
 Tshadowsize key, 136, 166
 viewpoint key, 136, 166
 pst-char package, 96, 154, 509
 \endpscharclip, 113, 155
 \pscharclip, 113, 155, 156
 \pscharpath, 112, 154, 155
 pst-coil package, 96, 120, 154, 160, 509
 coilarm key, 120, 159
 coilarmA key, 159

coilarmB key, 159
 coilaspect key, 159
 coilheight key, 159
 coilinc key, 159
 coilwidth key, 120, 159
 \ncccoil, 120, 160
 \nczigzag, 120, 161
 \pcccoil, 163
 \pczigzag, 163
 \psCoil, 155
 \pscoil, 154, 155
 pst-eps package, 96, 509
 \PSTtoEPS, 132, 152
 pst-fill package, 96, 509
 \psboxfill, 96
 pst-grad package, 96, 100, 105, 154, 157, 509
 gradangle key, 105, 157
 gradbegin key, 105, 108, 157
 gradend key, 105, 108, 157
 gradlines key, 157
 gradmidpoint key, 157
 pst-node package, 96, 159, 509
 < notation, 114, 123
 > notation, 114, 122, 123
 ~ notation, 114, 123
 _ notation, 114, 123
 angle key, 161
 angleA key, 118, 123, 125, 128, 129, 132, 162
 angleB key, 114, 117, 118, 119, 124, 125, 128, 129, 131, 132, 162
 arcangle key, 121, 161
 arm key, 162
 armA key, 125, 162
 armB key, 129, 131, 162
 boxsize key, 162
 \circlednode, 116, 160, 162, 163
 \Cnode, 116, 159, 162, 163
 \cnode, 116, 159, 163
 \cnodeput, 159
 colsep key, 122, 123, 125, 163
 \dianode, 116, 160, 162, 163
 \dotnode, 117, 159, 162, 163
 emnode key, 162
 \fnode, 117, 159, 162, 164
 loopsize key, 119, 162
 mcol key, 162
 mnode key, 123, 124, 162
 mnodesize key, 163
 name key, 124, 125, 128, 162
 \naput, 114, 123, 131, 161
 \nbput, 114, 123, 131, 132, 161
 \ncangle, 118, 119, 125, 128, 129, 130, 131, 160
 \ncangles, 118, 119, 160
 \ncarc, 114, 118, 160, 161
 \ncarcbox, 162
 \ncbar, 118, 132, 160
 \ncbox, 162
 \nccircle, 120, 160
 \nccurve, 114, 118, 123, 160, 162
 \ncdiag, 118, 119, 160
 \ncdiagg, 118, 119, 160
 \ncLine, 160
 \ncline, 115, 116, 122, 123, 160
 \ncloop, 118, 119, 160, 162
 \ncput, 161
 ncurv key, 123, 162
 ncurvA key, 162
 ncurvB key, 162
 nodealign key, 162
 nodesep key, 161
 nodesepA key, 161
 nodesepB key, 161
 npos key, 131, 162
 \nput, 161
 nrot key, 132, 162
 offset key, 123, 161
 \ovalnode, 116, 160, 162, 164
 \pcangle, 163
 \pcarc, 121, 163
 \pcbar, 163
 \pccurve, 163
 \pcdiag, 163
 \pcline, 163
 \pcloop, 163
 \pnode, 115, 117, 159, 162, 164
 psmatrix environment, 121, 122-125
 \psspan, 122
 radius key, 116, 127
 ref key, 124, 128, 162
 \Rnode, 115, 159, 162, 163

\Rnode, 114, 115, 117-119, 159, 162, 164
 rowsep key, 122, 123, 125, 163
 shortput key, 114, 122, 162
 \taput, 114, 127, 161
 \tbput, 114, 127, 161
 \thput, 161
 \tlput, 114, 130, 161
 tpos key, 162
 \trinode, 116, 117, 160, 162, 164
 \trput, 114, 122, 130, 161
 \tvput, 161
 pst-plot package, 96, 165, 509
 axesstyle key, 134, 166
 \dataplot, 132, 134, 165
 Dx key, 166
 dx key, 166
 Dy key, 134, 166
 dy key, 134, 166
 \fileplot, 132, 152, 165
 labels key, 166
 \listplot, 132, 165
 Ox key, 166
 Oy key, 166
 \parametricplot, 165
 plotpoints key, 133, 166
 plotstyle key, 133, 134, 166
 \psaxes, 133, 134, 165
 \psplot, 133, 165
 \psxlabel, 165
 \psylabel, 165
 \readdata, 132, 134, 165
 \savedata, 132, 165
 showorigin key, 166
 ticks key, 166
 ticksize key, 166
 tickstyle key, 134, 166
 pst-text package, 96, 154, 509
 \endpsclip, 100, 101
 \pscharclip, 154
 \psclip, 100, 101
 \pstextpath, 109, 112, 154, 156
 pst-tree package, 96, 163, 509
 bbd key, 164
 bbh key, 164
 bbl key, 164
 bbr key, 164
 edge key, 128, 132, 164

\endpsTree, 163
 fansize key, 164
 levelsep key, 126, 129, 131, 164
 nodesep key, 126, 128, 130
 offsetA key, 131
 \psedge, 127, 128, 129, 131
 \psTree, 163
 \pstree, 125, 126-132, 163
 showbbox key, 132, 164
 \skiplevel, 164
 \skiplevels, 164
 \TC, 125, 126, 131, 163
 \Tc, 163
 \Tcircle, 131, 163
 \Tdia, 163
 \Tdot, 127, 163
 \Tf, 164
 \Tfan, 164
 thislevelsep key, 164
 thistreefit key, 164
 thistreenodesize key, 164
 thistreesep key, 164
 \Th, 131, 163
 tndepth key, 164
 tnheight key, 164
 tnpos key, 127, 164
 tnsep key, 164
 \Toval, 164
 \Tp, 131, 132, 164
 \TR, 126, 128, 163
 \Tr, 129, 131, 164
 treefit key, 164
 treeflip key, 164
 treemode key, 127-129, 131, 165
 treenodes key, 165
 treesep key, 165
 \Tspace, 163
 \Ttri, 164
 xbbd key, 129, 165
 xbbh key, 165
 xbb1 key, 132, 165
 xbb2 key, 130, 132, 165
 pstcol package, 97, 100, 153, 154, 318, 342, 344, 509
 pstops program, 436, 437, 509
 pstotext program, 452, 509
 pstpoly package, 145, 509

PSTricks package, see packages
 psticks, pst-3d, pst-char,
 pst-coil, pst-eps, pst-fill,
 pst-grad, pst-node, pst-plot,
 pst-text, and pst-tree, xxii,
 xxiii, 4, 5, 11, 19, 20, 81,
 95-166, 222, 250, 318, 326, 342,
 435, 506, 508, 509
 pstricks package, 97, 153, 154, 509
 (- notation, 103
 (-) notation, 103
 * notation, 106
 ** - notation, 103
 *** - notation, 103
 * - notation, 103
 * - * notation, 103
 + notation, 106
 - notation, 103, 122
 -) notation, 103
 - * notation, 103
 - ** notation, 103
 - < notation, 103
 - < < notation, 103
 - > notation, 103, 104, 118, 122,
 123, 131
 - > > notation, 103
 - > | notation, 103
 - > | * notation, 103
 - C notation, 103
 -] notation, 103
 - c notation, 103
 - cc notation, 103
 - o notation, 103
 - oo notation, 103
 - | notation, 103
 - | * notation, 103
 < notation, 122
 < - notation, 103, 123, 131
 < - > notation, 103, 104
 < < - notation, 103, 122
 < < - > notation, 103
 > - notation, 103
 > - < notation, 103
 > > - notation, 103
 > > - < notation, 103
 [- notation, 103
 [-] notation, 103
 | notation, 106

| * - notation, 103
 | * - | * notation, 103
 | - notation, 103
 | - > notation, 106
 | - | notation, 103
 | < * - notation, 103
 | < * - > | * notation, 103
 | < - notation, 103
 | < - > | notation, 103
 arcsep key, 158
 arcsepA key, 158
 arcsepB key, 158
 arrowinset key, 158
 arrowlength key, 104, 158
 arrows key, 100, 123, 124, 131,
 132, 157
 arrowscale key, 158
 arrowsize key, 158
 asterisk notation, 106
 border key, 157
 bordercolor key, 157
 boxsep key, 158
 bracketlength key, 158
 C - notation, 103
 c - notation, 103
 C - C notation, 103
 c - c notation, 103
 cc - notation, 103
 cc - cc notation, 103
 cornersize key, 157
 \cput, 102, 159
 curvature key, 158
 \curveto, 145
 dash key, 157
 \degrees, 99, 121, 135
 diamond notation, 106
 diamond* notation, 106
 dimen key, 99, 100, 144, 156
 dotangle key, 158
 dotscale key, 105, 158
 dotsep key, 157
 dotsize key, 144, 158
 dotstyle key, 105, 108, 133, 144,
 158
 doublecolor key, 157
 doubleline key, 103, 108, 157
 doublesep key, 103, 157
 \fill, 143, 145

fillcolor key, 104, 105, 107,
112, 113, 124, 134, 135, 142,
145, 156
fillstyle key, 104, 105, 107,
108, 112, 113, 124, 134, 142,
145, 157
framearc key, 104, 105, 123, 125,
131, 157
framesep key, 124, 158
gangle key, 158
\grestore, 143, 145
gridcolor key, 158
griddots key, 159
gridlabelcolor key, 159
gridlabels key, 159
gridwidth key, 158
\gsave, 143, 145
hatchangle key, 105, 157
hatchcolor key, 105, 157
hatchsep key, 157
hatchwidth key, 157
labelsep key, 123, 127, 158
liftpen key, 142, 145
linear key, 103, 157
linecolor key, 98, 100, 104–109,
112, 113, 123, 128, 133, 134,
136, 144, 156
linestyle key, 101, 103, 104,
108, 110, 129, 132, 133, 142, 157
\lineto, 145
linewidth key, 98, 100, 103, 104,
106, 108, 112, 142, 144, 156, 157
\moveto, 143
\mrestore, 143
\msave, 143
\multido, 139
\multips, 139
\multirput, 139
\newpath, 143
\newsobject, 140
\newsstyle, 142
o notation, 106
o- notation, 103
o-o notation, 103
oo- notation, 103
oo-oo notation, 103
oplus notation, 106
origin key, 156
otimes notation, 106
\parabola, 105, 106, 154
pentagon notation, 106
pentagon* notation, 106
\psaddtolength, 98
\psarc, 105, 106, 109, 142, 154, 158
\psarcn, 109, 154
\psbezier, 106, 155
\psccurve, 155
\pscicle, 104, 110, 155, 156
\pscirclebox, 102, 108, 121, 155,
160
\pscurve, 105, 109, 145, 155
\pscustom, 142
\psdblframebox, 108, 155
\psdiabox, 108, 155, 160
\psdiamond, 101, 104, 155
\psdots, 105, 155, 159
\psecurve, 155
\psellipse, 105, 155
\psframe, 100, 104, 155, 159
\psframebox, 101, 107, 108, 117,
124, 155
\psgrid, 100, 136, 155
\psline, 98, 99, 101, 102, 103, 142,
155, 156
\psovalbox, 108, 155, 160
\pspicture environment, 97, 100,
102, 155
\pspolygon, 104, 156
\psset, 98, 99–101
\pssetlength, 98, 152
\psshadowbox, 107, 156
\pstdriver, 154
\pstheader, 154
\pstriangle, 104, 156
\pstribox, 108, 156, 160
pstricks.con file, 153
\pstrotate, 154
\pstunit, 154
\pstVerb, 154
\pstverb, 154
\pstverscale, 154
\pswedge, 105, 135, 156
\pszigzag, 152, 156
\qdisk, 156
\qline, 156
\radians, 99
rbracketlength key, 158
\rcurveto, 145
\rlineto, 145
\rotate, 143
\rput, 101, 102, 110, 112, 114, 116,
139, 148
runit key, 99, 156
\scale, 143
\scalebox, 153
shadow key, 108, 112, 157
shadowangle key, 112, 157
shadowcolor key, 157
shadowsize key, 157
showpoints key, 103, 105, 106,
156
\SpecialCoor, 99, 121, 135
square notation, 106
square* notation, 106
\stroke, 143
style key, 142
subgridcolor key, 159
subgriddiv key, 100, 159
subgriddots key, 159
subgridwidth key, 159
swapaxes key, 156
tbsize key, 158
\translate, 143
triangle notation, 106
triangle* notation, 106
unit key, 99, 100, 136, 144, 156
\uput, 102, 158, 435
x notation, 106
xunit key, 99, 132–134, 156
yunit key, 99, 132–134, 156
psutils program, 413, 433–442, 458,
506, 507, 509, 510
pxltopk program, 364, 509
rbbox METAPOST package, 69, 509
rboxit, 69
rotating package, 44, 49, 50, 509
sidewaysfigure environment,
50
sidewaystable environment, 50
Scientific Word program, 29, 375, 509
Scribe program, 437
sem-page package
 \slideheight length, 343

\slideleftmargin, 343
\sliderrightmargin, 343
\slidewidth length, 343
semcolor package, 342, 509
 \scplain, 346
seminar package, 153, 338–346, 433,
504, 509, 510
 article option, 341
 \makeslideheading, 343
 \newpagestyle, 344
 \newsframe, 344, 345
 note environment, 341
 notes option, 341
 \noteslides, 341
 notesonly option, 341
 \onlideslides, 341
 \pagestyle, 340
 portrait option, 339
 \printlandscape, 342
 seminar.con file, 342, 346
 slide environment, 339, 341
 slide* environment, 339
 \slideslides, 339
 \slideheading, 343
 \slideplacement, 341
 slidesonly option, 341
Separator program, 347
showchar program, 437, 510
slidesec package, 343, 510
 \listofslides, 343
 \Slidecontents, 343
SLiTeX package, 338
sprite package, 12, 13, 510
 \endsprite, 13
 \sprite, 12, 13
SPSS program, 25, 510
t1utils program, 372, 510
teTeX program, 455, 497, 510
TeX-PS package, 432, 510
TeXcad program, 6, 18, 510
texpic package, 23
Textures program, 19, 28, 29, 318, 348,
388, 413–415, 453, 510
tftopl program, 365, 367, 400, 510
times package, 353, 354, 359, 510
timing package, 239–243, 510
 \conttimingcounter, 240
 \larw, 243
 \linethickness, 243
 \larw, 243
 \sline, 241, 243
 \ttl, 240
 \timadjust, 243
 \timescalefactor, 243
 timing environment, 239, 240,
 241
 \timingcounter, 240
 \tin, 240, 241
 \tnote, 240, 241
tpic program, 11, 18, 22, 250, 251, 415,
424, 427, 464–466
Tpscript program, 437
Transcript program, 437
trees package, 127, 510
trig package, 51, 510
troff program, 22, 66–68, 437, 510
TrueTeX program, 29, 375, 453, 510
utopia package, 354, 510
vftovp program, 367, 510
vptovf program, 367, 390, 393, 400,
510
VTeX program, 375, 414, 510
Web2c program, 385, 410, 497, 507,
508, 510
Windows Write program, 437
WordPerfect program, 437
xdvi program, 11, 28, 510
xfig program, 1, 6, 25, 83, 462, 510
X_YTeX package, *see* packages aliphatic,
 carom, ccycle, chemstr, hcycle,
 hetarom, hetaromh, and
 lowcycle, 208–221, 510
Xy-pic package, xxii, xxiii, 4, 5, 10, 19,
72, 167–204, 510
 ! notation, 172, 173, 185, 186, 189
 " notation, 189
 "... notation, 170
 ' notation, 180, 182, 189
 (...) notation, 170
 (...) notation, 181, 182
 * notation, 168–170, 171, 172, 173,
 175–177, 181, 185, 198
 ** notation, 170, 171, 172,
 175–177, 193
 + notation, 168, 171, 173, 175
 ++ notation, 171, 173, 182
 += notation, 173
 , notation, 169
 - notation, 173, 180, 184
 -- notation, 173
 /.../ notation, 172
 /~.../ notation, 173
 /.../ notation, 172
 /d.../ notation, 182
 /l.../ notation, 183, 184
 /r.../ notation, 170, 183
 /u.../ notation, 182
 : notation, 170, 184
 ; notation, 170, 177, 180
 < notation, 171, 199, 200
 <...> notation, 169
 <<<< notation, 182
 = notation, 170, 173, 179, 189
 > notation, 171, 199, 200
 ? notation, 171, 175, 176
 ?! notation, 171
 ?< notation, 171
 ?>>> notation, 171
 @ notation, 172, 178
 @*[F] notation, 183, 184
 @*[o] notation, 181, 182
 @*[r] notation, 181, 182
 @H notation, 183, 184
 @M notation, 183
 @R notation, 183
 @W notation, 183, 184
 @[*] notation, 172, 173
 @{+} notation, 172, 175, 176
 @{-} notation, 170, 171, 195, 196
 @{-} notation, 170, 171, 185, 187,
 193, 194
 @{.} notation, 170, 171, 193, 194
 @{<<} notation, 171
 @{=} notation, 176
 @{=} notation, 170, 192

@{>} notation, 171
 @{} notation, 172
 @{} notation, 172
 @{} notation, 179, 203
 [F] notation, 168, 169, 171–174,
 178, 179, 182, 183, 185, 195
 [o] notation, 171, 173, 175, 179,
 182, 185, 194
 & notation, 168, 175, 181, 184
 \<cross, 199
 \<twist, 199
 \<cc>compositemap, 188
 \<cc>lowertwocell, 188
 \<cc>twocell, 188
 \<cc>uppertwocell, 188
 ~ notation, 178, 180, 189, 197, 201
 ~ notation, 191, 202, 204
 ~* notation, 176, 191, 192, 194,
 202, 203
 ~** notation, 176
 ~: notation, 192, 193, 194, 202,
 203
 ~< notation, 191, 193, 194
 ~<< notation, 191, 192
 ~<> notation, 191, 192–194
 ~= notation, 191, 194, 195, 203
 ~> notation, 191, 192, 194, 202, 203
 ~>< notation, 191, 192
 ~>> notation, 191, 192
 \\, 168, 173, 181
 _ notation, 178, 180, 189, 201
 ' notation, 180, 182, 188, 189
 | notation, 180, 199, 200
 2cell option, 188
 0 notation, 170, 178
 1 notation, 178
 2 notation, 178
 3 notation, 178
 all option, 168, 178
 \ar, 168, 172, 178, 179–183, 185,
 189, 190, 195–198
 arc option, 195
 arrow option, 168, 178, 179, 180,
 181, 184, 190, 199
 C notation, 172
 color option, 168, 174
 \croplattice, 198
 \crv, 175, 176
 curve option, 168, 174, 175, 179,
 195, 199
 D notation, 172
 \drop, 197, 198
 \ellipse, 187, 195, 196, 197
 \endxy, 169, 180
 \entrymodifiers, 182
 frame option, 168, 174, 176, 177,
 179
 \frm, 172, 176, 177, 202
 graph option, 168, 184, 185, 201
 \hcap, 201
 import option, 174
 knot option, 178, 198, 199
 \knotholesize, 202, 203
 L notation, 172
 \labelstyle, 189, 199–203
 \latticeA, 198
 \latticeB, 198
 \latticebody, 197, 198
 \latticeX, 198
 \latticeY, 198
 Lc notation, 176
 line option, 168, 174
 matrix option, 168, 178, 180, 181,
 184
 \newdir, 170, 181, 182
 \newgraphescape, 185, 186, 187
 \objectmarginlength, 191
 \objectstyle, 189, 192, 194, 199,
 202, 203

\omit, 188, 189, 190
 poly option, 190, 202
 \POS, 180, 183, 185, 187
 ps option, 175
 R notation, 172
 \restore, 183, 184, 185, 187, 202
 rotate option, 168, 174, 186
 \rrtwocell, 189, 190
 \rtwocell, 188, 189
 \save, 183, 184, 185, 187, 202
 \SloppyCurves, 176
 \splinetolerance, 176
 tile option, 174
 tips option, 168, 174, 181
 \turnradius, 184
 \txt, 173
 U notation, 172
 \UseAllTwocells, 188, 189
 \UseCompositeMaps, 188
 \UseHalfTwocells, 188
 \UseTwocells, 188, 190
 \vcap, 202
 \vloop, 203, 204
 \vover, 199, 201, 203
 web option, 197
 \xtwocell, 188, 189, 190
 \xunderv, 202, 203, 204
 \xy, 169
 xy environment, 169, 173, 191
 \xybox, 192, 197, 198
 \xygraph, 184, 185–187, 201–203
 \xylattice, 197
 \xymatrix, 168, 181, 182, 183,
 188–190
 \xyoption, 168
 \xypolygon, 190, 191–194, 202
 \xypolynome, 194
 \xypolynode, 192, 194, 202, 203
 \xypolynum, 192

Summary Index

Authors

Aplevitch, Dwight, 250
 Appelt, Wolfgang, 277, 278, 288
 Arseneau, Donald, 143

Barratt, Craig, 460
 Berry, Karl, 61, 352, 381, 385, 386,
 497
 Birrell, Andrew, 452

Bleser, Joachim, 17
 Braams, Johannes, 18
 Brown, Terry, 18
 Carlisle, David, 7, 17, 51, 326, 460

Cartledge, Ross, 433
 Clark, Adrian, 9, 288
 Clark, James, 22, 414
 Costabel, Martin, 12
 Dahlgren, Mats, 207, 208
 Deutsch, L. Peter, 442
 Donath, Dirk, 20
 Duggan, Angus, 8, 435
 Egler, Andreas, 254
 Esser, Thomas, 497
 Flynn, Peter, 498
 Freeman, Graham, 349
 Fujita, Shinsaku, 208
 Girou, Denis, 96, 110, 134, 138,
 140, 143, 145
 Grant, Michael, 460
 Gurari, Eitan M., 17
 Haas, Roswitha T., 208
 Hafner, Jim, 317, 338
 Hagen, Hans, 222
 Hamilton Kelly, Brian, 306
 Hansen, Carsten, 285
 Hassel, Frank, 284, 285
 Heinze, Henning, 431
 Hetherington, Lee, 372
 Hobby, John, 25, 62, 68, 72
 Hoenig, Alan, 54, 57
 Horn, Berthold, 386
 Horn, Georg, 6
 Jackowski, Bogusław, 93, 432, 450
 Jalbert, François, 254
 Jeffrey, Alan, 51, 68, 356, 386, 387,
 393, 401, 403
 Johnstone, Adrian, 18
 Kane, Kevin C., 208
 Kelley, Colin, 20
 Kernighan, Brian, 22
 Kiffe, Thomas, 168
 Knuth, Donald, 9, 13, 53, 302, 351,
 414
 Kołodziejska, Hanna, 295
 Kuykens, H.J.P., 274
 Kwok, Conrad, 18
 Lamers, Jürgen, 285
 Lampion, Leslie, 7, 10
 Lang, Russell, 450, 457
 Leathrum, Thomas, 84
 Leilich, Jens, 239

Levine, Michael, 228
 Lockyear, Kris, 16
 MacKay, Pierre, 386
 Malyshev, Basil, 342
 Mattes, Eberhard, 11
 May, Ludwig, 239
 McJones, Paul, 452
 Merz, Thomas, 443
 Mitchell, Ross, 254
 Mittelbach, Frank, 7, 293, 308
 Moore, Ross, 19, 167, 185
 Morgan, Tim, 11
 Neugebauer, Gerd, 306, 309
 Neumann, Gustav, 8
 Newton, Isacc, 312
 Nieuwenhuizen, Jan, 269
 Ohl, Thorsten, 84, 229, 234
 Olejniczak-Burkert, Rolf, 23
 Otten, A.F., 222
 Pianowski, Piotr, 93, 450
 Podar, Sunil, 17
 Poskanzer, Jef, 8
 Rahtz, Sebastian, 7, 44, 50, 288,
 353, 387
 Ramek, Michael, 208
 Richer, Jacques, 292
 Richter, Jörg, 299
 Rokicki, Tom, 28, 68, 360, 390, 414
 Rose, Kristoffer H., 19, 167
 Rovner, Marc, 13
 Rowley, Chris, 7
 Rubinstein, Zalman, 277
 Ryćko, Marek, 93
 Schofer, Angelika, 254
 Simons, Don, 255
 Sowa, Friedrich, 8
 Steinbach, Andrea, 254
 Tannert, Sebastian, 243
 Taupin, Daniel, 254, 257
 Theisen, Timothy, 450
 Tille, Andreas, 243
 Tobin, Geoffrey, 84
 Trevorrow, Andrew, 414
 Tutelaers, Piet, 277, 278, 453
 van der Laan, Kees, 58, 59, 303, 305
 van Zandt, Timothy, 95, 326, 338,
 433
 Vermaseren, Jos, 228

Vojta, Paul, 455
 Vollmer, Jürgen, 430
 Vulis, Michael, 414
 Walshaw, Chris, 264
 Wanske, Helene, 253
 Wernthges, Heinz W., 20
 White, Jan, 330
 Wichura, Michael, 14
 Williams, Graham, 498
 Williams, Thomas, 20
 Young, Thomas, 312

Document classes

letter, 298, 507
 seminar, 311, 323, 338, 339, 342
 slides, 311, 338, 510
 xymtex, 210

Environment Variables

GS_DEVICE (Ghostscript), 448
 GS_FONTPATH (Ghostscript), 446
 GS_LIB (Ghostscript), 446
 PRINTER (dvips), 416, 417, 419
 T1INPUTS (ps2pk), 455
 TEMP (Ghostscript), 448
 TEX (METAPOST), 66, 67

Files and File Extensions

.dvipsrc (dvips), 415
 8r.enc, 361, 370, 428
 .abc, xxiv, 269
 .afm, 288, 291, 375, 378, 387,
 389–391, 393–396, 401,
 406–408, 453–455
 .amfm, 373, 375
 aurora.pro, 349
 black.pro, 349
 bop-hax.tex, 432
 bridge.tex (bridge), 303
 chessboard.tex (cchess), 292
 cheq.map (cheq), 288
 chessboard.tex (cheq), 288
 color.cfg (color), 317
 color.pro (dvips), 320, 427
 colorsep.pro (dvips), 348, 468
 config.cheq (cheq), 288
 config.lino (dvips), 424
 config.ps (dvips), 288, 415, 419
 cyan.pro, 349
 devices.txt (Ghostscript), 450

.dvi, 6, 7, 22, 25, 66, 317, 320, 360,
363, 364, 413, 414, 417, 419,
422, 428, 429, 453, 457
dvips.ini (dvips), 416
.eps, 40, 41
.eps.gz, 41
.epsi, 452
.etx, 396
.fd, 358, 359, 361, 362, 400
fmpict.n (feynmf), 231
fmpict.mf (feynmf), 230
fmpict.mp (feynmf), 231
fmpict.tif (feynmf), 230
fntguide.tex, 353
Fontmap (Ghostscript), 445, 446,
447
.gf, 60–62, 230, 363, 364, 370
.gif, 8
graphics.cfg (graphics), 28
gs_init.ps (Ghostscript), 445
gs_statd.ps (Ghostscript), 445
header.tex (abc), 265
hlcrima.mtx (fontinst), 402
kernoff.mtx (fontinst), 402
kernon.mtx (fontinst), 402
latin.mtx (fontinst), 393, 396,
397, 399
lhax.mtx (fontinst), 402
libcct.m4 (pic), 251
lmathit.mtx (fontinst), 402
.log, 230, 236
.ltx, xxiv
.m4, xxiv
magenta.pro, 349
.map, 291, 360–362, 386, 390
mftoeps.mf (mftoeps), 93
.midi, 254
missfont.log, 418
monocyr.mtx (fontinst), 401
monoot2.mtx (fontinst), 400
.mp, xxiv, 66
.mpp, xxiv, 273
.mpx, 66
.msp, 41
.mtx, 395, 396
multi.pro, 433, 434
music.tex (abc), 269
.mx1, 263, 264
.mx2, 263
ot1ptm.fd, 359, 361
.pbm, 8
.pcx, 8, 41
pertab.tex (chemsym), 208
.pfa, 288, 291, 362, 372, 389
.pfb, 362, 372, 389
.pfm, 389
.pic, xxiv
.pict, 41
pictex.sty (private), 16
.pk, 8, 25, 60, 61, 230, 288, 360, 362,
363, 364, 374, 413, 414,
417–421, 428, 433, 453–455
.pl, 288, 365, 367, 392, 396, 400
.png, 41
.pro, 408
.ps, 40, 417
.ps.bb, 40
.ps.gz, 40, 41
ps2ai.ps (Ghostscript), 452
ps2ascii.ps (Ghostscript), 451,
452
psfonts.ins, 353
psfonts.map (dvips), 68, 354,
362, 419, 428, 429
psnfss.map, 354
pstricks.con (pstricks), 153
ptmr8r.mtx (fontinst), 393
.ptx, xxiv
.pxl, 8, 364, 453
.pz, 41
seminar.cfg (seminar), 345
seminar.con (seminar), 342, 346
stdin (Ghostscript), 444
stdout (Ghostscript), 449
t1.etx (fontinst), 393, 396
tiptm.fd, 361
.tar, 500, 503
.tar.gz, 500
.tfm, 8, 62, 64, 288, 291, 361–363,
364, 365, 367, 368, 378, 385,
387, 389–391, 400, 418, 421, 455
.tif, 41
unix-lpr.txt (Ghostscript), 449
unsetluc.mtx (fontinst), 402
.vf, 361, 362, 367, 368, 390, 393,
400, 421

.vpl, 367, 369, 390, 391, 393, 400
w.eps (private), 30
xcheq.tif (cheq), 291
xchess.enc (cheq), 291
yellow.pro, 349
.zip, 500, 502, 503
.zoo, 500

Fonts

cchess46 (cchess), 292
Cheq (cheq), 288
cheq (cheq), 289, 290, 393
chess10 (chess), 278, 289–291
chess15 (bdfchess), 285
chess20 (chess), 278
chess30 (chess), 278
chessf10 (chess), 278
gosign50 (go), 296
xcheq (cheq), 291

Languages

abc, 254, 264–269
grap, 24–25
META, 25, 53–83
MfP, 269–274
pic, 22–23, 24, 26, 68, 150
PL/VPL, 364–370

Packages

2up, 433, 504
aldratex, 17, 504
aliphat, 210, 219, 504
amstex, 207, 504
array, 326, 504
avant, 353, 354, 504
axodraw, 228, 504
bakoma, 342, 504
bar, 17, 78, 504
basker, 354, 504
bdfchess, 284–285, 288, 504
bembo, 354, 504
bg, 299–302, 504
bookman, 353, 354, 504
boxes METAPOST, 68, 69, 72,
505
bridge, 303, 505
carom, 210, 214, 505
cchess, 293, 294, 505
ccycle, 210, 218, 505
chancery, 353, 354, 505
charter, 354, 505

chemstr, 210, 505
chemsym, 206, 207, 208, 209, 505
cheq, 288, 505
chess, 278–282, 285, 288, 289, 292,
294, 295, 505
chesskey, 285, 505
circ, 243–249, 505
color, 7, 27, 82, 100, 153, 311,
317–326, 348, 424, 464, 505
colordvi, 317, 505
colortab, 326, 505
colortbl, 323, 326–338, 505
crosswd, 306–308, 505
curves, 17, 505
cwpuzzle, 309, 505
dcolumn, 326, 505
draftcopy, 430, 505
dratex, 5, 17, 505
ecitree, 126, 505
eepic, 18, 20, 211, 212, 506
epic, 17, 18, 20, 208, 211, 212, 506
epsfig, 50, 506
fancybox, 342, 506
Feynarts, 228
feynman, 228, 506
feynmf, 84, 228–239, 506
feynmp, 84, 229, 230, 239, 506
foitex, 317, 338, 506
fontdoc, 487, 506
fontinst, 383, 387, 393–404, 410,
412, 487–493, 506
garamond, 354, 506
go, 295–298, 507
graph METAPOST, 72, 74, 75, 78,
83, 507
graphics, 7, 9, 27–32, 34, 39–45,
48–50, 51, 64, 99, 153, 231, 424,
500, 503, 507, 509, 510
graphicx, 27–29, 32–51, 98, 101,
461, 507
hcycle, 210, 218, 507
helvet, 353, 354, 507
hetarom, 210, 216, 217, 220, 507
hetaromh, 210, 216, 220, 507
hhline, 326, 329, 336, 418, 507
ifthen, 92, 198, 324, 325, 507
inputenc, 362, 507
keyval, 38, 51, 507

lcircuit, 18, 507
locant, 210, 507
longtable, 208, 326, 329, 507
lowcycle, 210, 215, 507
lscap, 51, 507
lucbr, 354, 507
m-chemic, 221–228, 507
mathptm, 68, 353, 354, 356, 357,
401, 508
mfpic, 25, 54, 84–92, 250, 508
mftoeps METAFONT, 93, 508
mtimes, 354, 508
multido, 139, 508
nassflow, 18, 508
newcent, 353, 354, 508
nimbus, 354, 508
MusikTeX, xxii–xxiv, 253, 254–264,
265, 267, 269, 273, 274, 508
MusicTeX, 254, 263, 269, 274, 275,
508
SliTeX, 338
Xy-pic, xxii, xxiii, 4, 5, 10, 19, 72,
167–204, 510
X_YTeX, 208–221, 510
PicTeX, 5, 14–16, 26, 222, 508
palatino, 353, 354, 508
pict2e, 17, 508
pifont, 323, 353, 356, 508
ppchtext, 221–228, 507, 509
PSfrag, 26, 460–462, 509
PSNFSS, 353–356, 359–362, 381,
385, 394, 504–508, 509, 510
pspicture, 17, 509
pst-3d, 96, 166, 509
pst-char, 96, 154, 509
pst-coil, 96, 120, 154, 160, 509
pst-eps, 96, 509
pst-fill, 96, 509
pst-grad, 96, 100, 105, 154, 157,
509
pst-node, 96, 159, 509
pst-plot, 96, 165, 509
pst-text, 96, 154, 509
pst-tree, 96, 163, 509
pstcol, 97, 100, 153, 154, 318, 342,
344, 509
pstpoly, 145, 509

PSTricks, xxii, xxiii, 4, 5, 11, 19, 20,
81, 95–166, 222, 250, 318, 326,
342, 435, 506, 508, 509
pstricks, 97, 153, 154, 509
rboxes METAPOST, 69, 509
rotating, 44, 49, 50, 509
semcolor, 342, 509
seminar, 153, 338–346, 433, 504,
509, 510
slideseq, 343, 510
sprite, 12, 13, 510
TeX-PS, 432, 510
texpic, 23
times, 353, 354, 359, 510
timing, 239–243, 510
trees, 127, 510
trig, 51, 510
utopia, 354, 510
Programs
abc2mtex, 254, 255, 264–269, 270,
504
Acrobat Distiller, 415, 455, 504
Acrobat Reader, 442, 504
Adobe Acrobat, 25, 372, 375, 406,
453, 504
Adobe File Utilities, 20, 504
Adobe Illustrator, 1, 4, 25, 68, 93,
94, 347, 442, 452, 504
Adobe Photoshop, 347, 504
Adobe Type Manager, 372, 375,
379, 380, 381, 388, 389, 406,
414, 447, 453, 504
afm2tfm, 291, 387, 390–393, 504
ArInfo, 25, 504
aurora, 349, 504
AutoCAD, 25, 83, 94, 152, 504
bit2spr, 13, 504
bitpxl, 8, 504
bm2font, 8, 466, 467, 504
chem, 24, 505
ChemDraw, 25, 505
CMacTeX, 168, 505
Corel Draw, 1, 93, 94, 347, 505
dpic, 250
dvi2ps, 29, 414, 505
dviaw, 29, 505
dvi copy, 360, 505
dvi drv, 415, 505

dvilaser/PS, 29, 437
 dvipdf, 29
 dvips, xxiii, 11, 19, 28, 29, 41, 65,
 68, 154, 288, 291, 317, 318, 320,
 342, 348, 349, 354, 361, 362,
 369, 370, 386, 387, 390, 396,
 408, 409, 414, 415–433,
 455–458, 460, 463, 468, 469,
 505
 dvipsone, 11, 19, 28, 29, 318, 414,
 505
 dvitomp, 66, 505
 dvitops, 29, 414, 505
 dviwin, 11, 29, 505
 dviwindo, 29, 388, 414, 453, 505
 Edmetrics, 388, 506
 emTeX, 28, 29, 505, 506, 510
 epsffit, 437, 442, 458, 506
 epstool, 457, 506
 Excel, 25, 506
 extractres, 437, 441, 506
 Feynarts, 506
 fig2mf, 83, 506
 fixdlsrps, 437, 506
 fixfmps, 437, 506
 fixmacps, 437, 506
 fixpsdtps, 437, 506
 fixsppps, 437, 506
 fixscribeps, 437, 506
 fixtpps, 437, 506
 fixwfwps, 437, 506
 fixwpps, 437, 506
 fixwwps, 437, 506
 flow, 18, 19, 506
 fontinst, 404
 FrameMaker, 437, 452, 506
 FreeHand, 1, 506
 getafm, 437, 506
 gftodvi, 363, 506
 gftopk, 20, 230, 363, 364, 506
 gftopxl, 364, 506
 gftype, 363, 506
 Ghostscript, xxiii, 375, 379, 408,
 431, 442–453, 455, 457–460,
 494, 506, 508, 509
 ghostview, 449, 450, 451, 507
 gnuplot, 20, 21, 83, 462, 507

gpic, 18, 22, 250, 251, 507
 grap, 24, 72, 507
 gsftopk, 418, 450, 453, 455, 507
 GSview, 449, 450, 457, 507
 gunzip, 41
 gzip, 500
 Hijaak, 20, 507
 hp2xx, 20, 507
 hptomf, 20, 507
 idraw, 1, 462
 Image Alchemy, 20, 507
 includeres, 437, 441, 507
 Internet Explorer, 498, 507
 latex2html, 457, 507
 lpr, 449, 507
 m4, 250, 251, 507
 Mac GS Viewer, 450, 507
 MacPaint, 3, 4
 makeindex, 84, 507
 makeinstance, 408, 507
 makempx, 66, 507
 MakeTeXPK, 415, 455, 507
 MakeTeXTFM, 410, 508
 Maple, 1, 508
 Mathematica, 1, 25, 508
 MATLAB, 1, 462, 508
 Microsoft Word, 437
 midi2tex, 254, 255, 274, 275, 508
 mmafm, 408, 508
 mpp, 273, 508
 mptotex, 66, 508
 music, 24, 508
 musixflx, 260, 263, 264, 508
 Netscape, 498, 508
 nfssfont.tex, 289, 508
 OzTeX, 29, 414, 508
 pbmtopk, 8, 508
 pctex32, 29, 508
 pctexhp, 29, 508
 pctexps, 29, 318, 508
 pctexwin, 29, 508
 pdf2ps, 453, 508
 pic, 20, 22–24, 184, 250–252, 508
 pie-chart.sh, 134, 508
 pktogf, 364, 508
 pktype, 364, 508
 pltotf, 365, 509

PMX, 255, 509
 pnmccrop, 457, 509
 ppmtogif, 457, 509
 METAFONT, 66
 ps2ascii, 452, 509
 ps2epsi, 452, 509
 ps2pdf, 453, 509
 ps2pk, 418, 453–455, 509
 ps_view, 450, 509
 psbook, 437, 439, 509
 psmerge, 437, 440, 509
 psnup, 435, 437, 438, 439, 509
 psprint, 29, 437, 509
 psresize, 437, 440, 509
 psselect, 437, 440, 509
 pstops, 436, 437, 509
 pstotext, 452, 509
 psutils, 413, 433–442, 458, 506,
 507, 509, 510
 pxltopk, 364, 509
 Scientific Word, 29, 375, 509
 Scribe, 437
 Separator, 347
 showchar, 437, 510
 SPSS, 25, 510
 t1utils, 372, 510
 teTeX, 455, 497, 510
 TeXcad, 6, 18, 510
 Textures, 19, 28, 29, 318, 348, 388,
 413–415, 453, 510
 tftopl, 365, 367, 400, 510
 tpic, 11, 18, 22, 250, 251, 415, 424,
 427, 464–466
 Tpscript, 437
 Transcript, 437
 troff, 22, 66–68, 437, 510
 TrueTeX, 29, 375, 453, 510
 vftovp, 367, 510
 vptovf, 367, 390, 393, 400, 510
 VTeX, 375, 414, 510
 Web2c, 385, 410, 497, 507, 508, 510
 Windows Write, 437
 WordPerfect, 437
 xdvi, 11, 28, 510
 xfig, 1, 6, 25, 83, 462, 510

All Commands

! notation (Xy-pic), 172, 173, 185,
 186, 189
 " notation (Xy-pic), 189
 " notation (abc), 268
 " . . . " notation (Xy-pic), 170
 " . . . " notation (pic), 22, 23
 ' notation (MusikTeX), 257
 ' notation (MP), 269, 270, 272, 273
 ' notation (Xy-pic), 180, 182, 189
 ' notation (abc), 266
 ' ' notation (MusikTeX), 257
 ' ' ' notation (MusikTeX), 257
 (– notation (pstricks), 103
 (–) notation (pstricks), 103
 (. . .) notation (Xy-pic), 170
 (. . .) notation (MP), 271, 272, 273
 (. . .) notation (Xy-pic), 181, 182
 (. . .) notation (abc), 268
 (2 notation (abc), 267
 (3 notation (abc), 267
 (4 notation (abc), 267
 * notation (Xy-pic), 168–170, 171,
 172, 173, 175–177, 181, 185, 198
 * notation (afm2tfm), 392
 * notation (chess), 278, 279, 282
 * notation (cwpuzzle), 309
 * notation (pstricks), 106
 ** notation (Xy-pic), 170, 171, 172,
 175–177, 193
 **– notation (pstricks), 103
 – notation (pstricks), 103
 *– notation (pstricks), 103
 – notation (pstricks), 103
 + notation (MP), 272
 + notation (Xy-pic), 168, 171, 173,
 175
 + notation (chess), 282
 + notation (dvips), 422
 + notation (m-chemic), 225
 + notation (pstricks), 106
 ++ notation (META), 55
 ++ notation (Xy-pic), 171, 173, 182
 +–+ notation (META), 55
 += notation (Xy-pic), 173
 , notation (MP), 270
 , notation (Xy-pic), 169
 , notation (abc), 266
 – notation (MP), 272
 – notation (Xy-pic), 173, 180, 184
 – notation (abc), 267, 268, 269
 – notation (chess), 282
 – notation (m-chemic), 225
 – notation (pstricks), 103, 122
 \– (circ), 246
 –) notation (pstricks), 103
 –* notation (pstricks), 103
 –** notation (pstricks), 103
 –– notation (META), 56
 –– notation (Xy-pic), 173
 –< notation (pstricks), 103
 –<< notation (pstricks), 103
 –> notation (pstricks), 103, 104, 118,
 122, 123, 131
 –>> notation (pstricks), 103
 –>| notation (pstricks), 103
 –>|* notation (pstricks), 103
 –>|* notation (pstricks), 103
 –J notation (pstricks), 103
 –c notation (pstricks), 103
 –cc notation (pstricks), 103
 –o notation (pstricks), 103
 –oo notation (pstricks), 103
 –| notation (pstricks), 103
 –|* notation (pstricks), 103
 . notation (MusikTeX), 259
 . notation (abc), 268
 \. (circ), 245
 . . notation (META), 55, 56
 .PE notation (pic), 22, 250
 .PS notation (pic), 22, 250
 .c notation (pic), 23
 .n notation (pic), 23
 .ne notation (pic), 23
 .notdef (PostScript), 291, 392
 .nw notation (pic), 23
 .se notation (pic), 23
 .sw notation (pic), 23
 / notation (MP), 272
 / notation (abc), 267, 268
 \/, 364
 / . . . / notation (Xy-pic), 172
 / ^ . . . / notation (Xy-pic), 173
 / _ . . . / notation (Xy-pic), 172
 / d . . . / notation (Xy-pic), 182
 / 1 . . . / notation (Xy-pic), 183, 184
 / r . . . / notation (Xy-pic), 170, 183
 / u . . . / notation (Xy-pic), 182
 : notation (Xy-pic), 170, 184
 : notation (chess), 282
 :: notation (abc), 266
 : | notation (MP), 270
 : | notation (abc), 266
 : | : notation (MP), 270
 : | | : notation (MP), 270
 ; notation (Xy-pic), 170, 177, 180
 ; notation (pic), 23
 < notation (MusikTeX), 257
 < notation (Xy-pic), 171, 199, 200
 < notation (abc), 267, 268
 < notation (dvips), 428
 < notation (pst-node), 114, 123
 < notation (pstricks), 122
 <– notation (pstricks), 103, 123, 131
 <–> notation (pstricks), 103, 104
 < . . . > notation (Xy-pic), 169
 << notation (abc), 267
 <<– notation (pstricks), 103, 122
 <<–> notation (pstricks), 103
 <<<< notation (Xy-pic), 182
 = notation (MusikTeX), 257
 = notation (Xy-pic), 170, 173, 179,
 189
 = notation (abc), 267
 = notation (dvips), 416

=: notation (afm2tfm), 392
 =: | notation (afm2tfm), 392
 =: |> notation (afm2tfm), 392
 > notation (MusikTeX), 257
 > notation (Xy-pic), 171, 199, 200
 > notation (abc), 267, 268
 > notation (afm2tfm), 392
 > notation (pst-node), 114, 122, 123
 >- notation (pstricks), 103
 >-< notation (pstricks), 103
 >> notation (abc), 267
 >>- notation (pstricks), 103
 >>-<< notation (pstricks), 103
 ? notation (Xy-pic), 171, 175, 176
 ?! notation (Xy-pic), 171
 ?< notation (Xy-pic), 171
 ?>>> notation (Xy-pic), 171
 @ notation (Xy-pic), 172, 178
 @ notation (dvips), 422
 @*[F] notation (Xy-pic), 183, 184
 @*[O] notation (Xy-pic), 183, 182
 @*[r] notation (Xy-pic), 183, 182
 @H notation (Xy-pic), 183, 184
 @M notation (Xy-pic), 183
 @R notation (Xy-pic), 183
 @W notation (Xy-pic), 183, 184
 @[*] notation (Xy-pic), 172, 173
 @{+} notation (Xy-pic), 172, 173, 176
 @{-} notation (Xy-pic), 170, 171, 195, 196
 @{-} notation (Xy-pic), 170, 171, 185, 187, 193, 194
 @{.} notation (Xy-pic), 170, 171, 193, 194
 @{<<} notation (Xy-pic), 171
 @{==} notation (Xy-pic), 176
 @{=} notation (Xy-pic), 170, 171
 @{>} notation (Xy-pic), 171
 @{o} notation (Xy-pic), 172
 @{x} notation (Xy-pic), 172
 @{...} notation (Xy-pic), 179, 203
 [- notation (pstricks), 103
 [-] notation (pstricks), 103

[...] notation (M_P), 271, 272, 273
 [...] notation (abc), 268
 [...] / notation (M_P), 272
 [1 notation (abc), 266
 [2 notation (abc), 266
 [F] notation (Xy-pic), 168, 169, 171–174, 178, 179, 182, 183, 185, 195
 [^...] notation (M_P), 272
 [o] notation (Xy-pic), 171, 173, 175, 179, 182, 185, 194
 [v...] notation (M_P), 272
 [l notation (abc), 266
 #copies (PostScript), 420
 % notation (dvips), 420, 421
 %b notation (dvips), 421
 %d notation (dvips), 421
 %f notation (dvips), 421
 %m notation (dvips), 421
 %p notation (dvips), 421
 & notation (MusikTeX), 256, 261
 & notation (Xy-pic), 168, 175, 181, 184
 \<c>cross (Xy-pic), 199
 \<c>twist (Xy-pic), 199
 \<cc>compositemap (Xy-pic), 188
 \<cc>lowertwo cell (Xy-pic), 188
 \<cc>two cell (Xy-pic), 188
 \<cc>uppertwo cell (Xy-pic), 188
 ~ notation (MusikTeX), 257, 258
 ~ notation (M_P), 271, 273
 ~ notation (Xy-pic), 178, 180, 189, 197, 201
 ~ notation (abc), 267, 268
 ~ notation (chemsym), 207
 ~ notation (pst-node), 114, 123
 ~ notation (abc), 267
 ~ notation (Xy-pic), 191, 202, 204
 ~* notation (Xy-pic), 176, 191, 192, 194, 202, 203
 ~** notation (Xy-pic), 176
 ~: notation (Xy-pic), 192, 193, 194, 202, 203
 ~< notation (Xy-pic), 191, 193, 194

<< notation (Xy-pic), 191, 192
 <> notation (Xy-pic), 191, 192–194
 ~= notation (Xy-pic), 191, 194, 195, 203
 ~> notation (Xy-pic), 191, 192, 194, 202, 203
 ~>< notation (Xy-pic), 191, 192
 ~>> notation (Xy-pic), 191, 192
 \ (Xy-pic), 168, 173, 181
 \ notation (abc), 266, 268
 { notation (M_P), 273
 {...} notation (M_P), 271, 272
 {...} notation (abc), 268
 {} notation (afm2tfm), 392
 {} notation (cwpuzzle), 309
 } notation (M_P), 273
 _ notation (MusikTeX), 257, 258
 _ notation (M_P), 271, 272
 _ notation (Xy-pic), 178, 180, 189, 201
 _ notation (abc), 267
 _ notation (chemsym), 207
 _ notation (pst-node), 114, 123
 _ notation (abc), 267
 ' notation (MusikTeX), 257
 ' notation (M_P), 269
 ' notation (Xy-pic), 180, 182, 188, 189
 ' notation (dvips), 41, 420
 | notation (MusikTeX), 256, 261
 | notation (M_P), 270
 | notation (Xy-pic), 180, 199, 200
 | notation (abc), 265, 266, 267, 268
 | notation (afm2tfm), 392
 | notation (chess), 278, 280, 282
 | notation (pstricks), 106
 | notation (pstricks), 103
 |* notation (pstricks), 103
 |*- notation (pstricks), 103
 |- notation (pstricks), 103
 |-> notation (pstricks), 106
 |- | notation (pstricks), 103
 |. notation (M_P), 270–273
 |: notation (abc), 266

|<*- notation (pstricks), 103
 |<*->|* notation (pstricks), 103
 |<- notation (pstricks), 103
 |<->| notation (pstricks), 103
 |= notation (afm2tfm), 392
 |=> notation (afm2tfm), 392
 |=: notation (afm2tfm), 392
 |=: |> notation (afm2tfm), 392
 |] notation (abc), 265, 266, 267, 268
 || notation (M_P), 270
 || notation (abc), 266
 || notation (afm2tfm), 392
 ||| notation (M_P), 270
 0 notation (Xy-pic), 170, 178
 1 notation (Xy-pic), 178
 2 notation (Xy-pic), 178
 3 notation (Xy-pic), 178
 \A (M_P), 270
 \A (circ), 244, 247
 abs (META), 57
 \ACROSS text (crosswd), 308
 \adamantane (ccycle), 218
 \add (fontinst), 493
 ADJ notation (m-chemic), 225, 226
 \allabreve (MusikTeX), 257
 \alltt environment (alltt), 12
 \alto (M_P), 269, 272
 \altoclef (MusikTeX), 257
 angle (META), 55
 angle key (graphicx), 33, 34, 36, 37
 angle key (pst-node), 161
 angleA key (pst-node), 118, 123, 125, 128, 129, 132, 162
 angleB key (pst-node), 114, 117, 118, 119, 124, 125, 128, 129, 131, 132, 162
 \anthracenev (carom), 214
 \ar (Xy-pic), 168, 172, 178, 179–183, 185, 189, 190, 195–198
 \arc (mfpic), 86
 arc (pic), 22
 arcangle key (pst-node), 121, 161

arcsep key (pstricks), 158
 arcsepA key (pstricks), 158
 arcsepB key (pstricks), 158
 arm key (pst-node), 162
 armA key (pst-node), 125, 162
 armB key (pst-node), 129, 131, 162
 \arpeggio (MusikTeX), 257
 array environment, 9, 49, 326
 \arrayrulecolor (colorbl), 329, 332, 333, 334, 336, 337
 \arrayrulewidth length, 329
 \arrow (mfpic), 86, 88, 91
 arrow (pic), 22
 arrowinset key (pstricks), 158
 arrowlength key (pstricks), 104, 158
 arrows key (pstricks), 100, 123, 124, 131, 132, 157
 arrowscale key (pstricks), 158
 arrowsize key (pstricks), 158
 \ascender (fontinst), 489
 asterisk notation (pstricks), 106
 \at (circ), 246
 atan (pic), 22
 \AtBeginDvi, 422
 \atpin (circ), 247, 248
 augment (graph), 77, 78–80
 autogrid (graph), 75, 76, 79, 81–83
 \axes (mfpic), 86, 87, 90
 axesstyle key (pst-plot), 134, 166
 \axisheadlen length (mfpic), 86, 87
 B notation (m-chemic), 223, 225
 \backturn (MusikTeX), 257
 \bar (MusikTeX), 256, 259–261
 \bass (MusikTeX), 261
 \bass (M_P), 270, 273
 \bassclef (MusikTeX), 257
 \Battery (private), 150
 bb key (graphicx), 32, 33, 34
 bbd key (pst-tree), 164
 \bbetter (chess), 282, 283
 bbh key (pst-tree), 164
 bbl key (pst-tree), 164
 bblx key (graphicx), 32
 bblly key (graphicx), 32
 bbox (METAPOST), 65, 66, 79, 81
 bbr key (pst-tree), 164
 bburr key (graphicx), 32
 bbury key (graphicx), 32
 \bclosed (mfpic), 86, 88
 \bdecisive (chess), 282
 beginchar (META), 60, 63
 beginfig (METAPOST), 63
 begingraph (graph), 74
 \benzofuranv (hetarom), 217
 \benzofuranevi (hetarom), 217
 \benzoxazolev (hetarom), 217
 \benzoxazolevi (hetarom), 217
 \Bes (M_P), 271–273
 bes notation (M_P), 270
 \betteris (chess), 282, 283
 \bicychep (ccycle), 218
 \bicychepv (ccycle), 218
 \bid (private), 305, 306
 bidding environment (bridge), 303, 305
 \bigboard (bg), 300
 bind (PostScript), 437, 445
 \Black (chess), 279, 281
 \black (go), 296, 297, 298
 black (METAPOST), 64
 black notation (color), 318, 319
 \blackbar (bg), 299, 300
 \blackcube (bg), 300
 \blackonmove (bg), 300
 \blackpoint (bg), 299
 BlendDesignMap (PostScript), 406
 BlendDesignPositions (PostScript), 406
 \BLens (circ), 246, 248
 blue (METAPOST), 64
 blue notation (color), 318, 319
 \board (chess), 278, 279, 292
 \boardcaption (bg), 299, 300, 301
 boolean (META), 54
 bop-hook (PostScript), 420, 429, 431

border key (pstricks), 157
bordercolor key (pstricks), 157
\bornane (ccycle), 218
bot notation (METAPOST), 64
\botaccent (fontinst), 398
\boundarychar (fontinst), 488
BoundingBox (PostScript), 28, 30, 33, 40, 41, 63, 456
box (pic), 22, 23
\boxit (MusiXTeX), 257
boxit (boxes), 69, 70–72
boxjoin (boxes), 69, 70–73
boxsep key (pstricks), 158
boxsize key (pst-node), 162
\Bpara (private), 49, 50
bpath (METAPOST), 70, 71, 73
\bracket (MusiXTeX), 257
bracketlength key (pstricks), 158
\BSplit (circ), 246, 248
btex (METAPOST), 64, 65, 66, 67, 72, 74, 75, 78, 80
\btwnfcn (mfpic), 89
buildcycle (METAPOST), 81
\bundle (circ), 246
\bupperband (chess), 282, 283
\bye (fontinst), 393
\bzdrh (carom), 211, 213, 214
\bzdrv (carom), 211, 212, 214
C notation (Xy-pic), 172
C notation (m-chemic), 225
\C (circ), 244
C- notation (pstricks), 103
c- notation (pstricks), 103
C-C notation (pstricks), 103
c-c notation (pstricks), 103
C: notation (abc), 266
\ca (MusiXTeX), 258, 259, 260
\caesura (MusiXTeX), 257
\Cam (circ), 246, 248
\capheight (fontinst), 489
CARBON notation (m-chemic), 222, 223
\cardmessage (bdfchess), 286, 287

CB notation (m-chemic), 222
\cbclosed (mfpic), 88
\cbreath (MusiXTeX), 257
\cc (circ), 245, 247, 248
cc- notation (pstricks), 103
cc-cc notation (pstricks), 103
\cca (MusiXTeX), 258, 259, 260
\cccc1 (MusiXTeX), 257, 259
\ccccc (MusiXTeX), 257, 259
\cccc1 (MusiXTeX), 257, 259
\cccu (MusiXTeX), 257, 258
\cc1 (MusiXTeX), 257, 258, 259, 260
\ccu (MusiXTeX), 257, 258, 259, 260
\cdf1 (MusiXTeX), 257
\cdsh (MusiXTeX), 257
\Cel (circ), 244
\centerto (circ), 247
\cf1 (MusiXTeX), 257
\CH (chemsym), 207
\chair (ccycle), 218
CHARACTER (PL/VPL), 367, 368
CharStrings (PostScript), 370
CHECKSUM (PL/VPL), 368
\chemical (m-chemic), 222, 223, 224–227, 228
\CheqBKing (cheq), 288
\Chess (chess), 285, 289, 292
chess environment (chess), 282
\chl (MusiXTeX), 257
\chu (MusiXTeX), 257
CIEBasedA (PostScript), 445
CIEBasedABC (PostScript), 445
\cinnolinev (hetarom), 217
\cinnolinevi (hetarom), 217
\circle (mfpic), 86
circle (pic), 22, 23
\circleit (MusiXTeX), 257
circleit (boxes), 69, 71–73
\circlenode (pst-node), 116, 160, 162, 163
circmargin (boxes), 69, 73
\circuit (private), 186, 187

circuit environment (circ), 245, 247, 248
\cl (MusiXTeX), 257, 258
\clef (Mfp), 269, 272, 273
clip (META), 67
clip key (graphicx), 32, 33, 34
\closegraphsfile (mfpic), 85
\club (private), 302, 304–306
\clubsuit, 302
\Clue (cwpuzzle), 310
\clue (crossword), 306, 307–309
cmym color model (color), 317
\cna (MusiXTeX), 257
\Cnode (pst-node), 116, 159, 162, 163
\cnode (pst-node), 116, 159, 163
\cnodeput (pst-node), 159
\codingscheme (fontinst), 488
coilarm key (pst-coil), 120, 159
coilarmA key (pst-coil), 159
coilarmB key (pst-coil), 159
coilaspect key (pst-coil), 159
coilheight key (pst-coil), 159
coilinc key (pst-coil), 159
coilwidth key (pst-coil), 120, 159
\color (color), 318, 319, 320, 321, 329, 331, 332
color (METAPOST), 64, 73
\colorbox (color), 322, 324, 325, 330, 331, 333, 335, 427
colorimage (PostScript), 349
colsep key (pst-node), 122, 123, 125, 163
\columncolor (colortbl), 326, 327, 328, 333–337
\columnwidth length, 38
command key (graphicx), 33, 34
COMMENT (PL/VPL), 365, 367
\comment (fontinst), 398
connect environment (mfpic), 85, 88, 89
\connection (circ), 247
\conttimingcounter (timing), 240
ConvertDesignVector (PostScript), 406, 409
\COOH (chemsym), 207

coords environment (mfpic), 92
copy (grap), 24
cornersize key (pstricks), 157
cos (pic), 22
cosd (META), 55
\cput (pstricks), 102, 159
\cql (MusiXTeX), 257
\cqu (MusiXTeX), 257
\crdexta (private), 305
\crdim (bridge), 303, 304
\croplattice (Xy-pic), 198
crossword environment
(crossword), 306, 307, 308, 310
crossword* environment
(crossword), 308, 309
\crv (Xy-pic), 175, 176
CRZ notation (m-chemic), 227
\cscglyph (fontinst), 398
\csh (MusiXTeX), 257
\cu (MusiXTeX), 257, 258, 259, 260
curl (META), 56
currentpicture (META), 66
curvature key (pstricks), 158
\curve (mfpic), 86, 92
\curveto (pstricks), 145
cutafter (METAPOST), 70, 71–73
cutbefore (METAPOST), 70, 71–73
\Cvar (circ), 244
cyan notation (color), 318, 319
cycle (META), 56, 57, 77–80
\cyclic (mfpic), 86
\cyclobutane (lowcycle), 216
\cyclohexaneh (carom), 213, 214
\cyclohexanev (carom), 212, 213, 214
\cyclopentaneh (lowcycle), 215
\cyclopentanev (lowcycle), 215, 216
\cyclopentanevi (lowcycle), 215
\cyclopropane (lowcycle), 216
D notation (Xy-pic), 172

\D (circ), 244
D: notation (abc), 266
\darkershade (mfpic), 88
dash key (pstricks), 157
\dashed (circ), 246
dashed (METAPOST), 72, 73, 74, 78
dashed (pic), 23
\dashlen length (mfpic), 87, 88
\dashlineset (mfpic), 88
\dashspace length (mfpic), 87, 88
\dataplot (pst-plot), 132, 134, 165
DB notation (m-chemic), 225
\Dcap (circ), 244
\dcqu (MusiXTeX), 257
\decaheterov (hetarom), 217
\decalineth (carom), 214
\decalineth (carom), 214
\declareencoding (fontinst), 493
\DeclareFixedFont, 112, 113
\DeclareFontEncoding, 404
\DeclareFontFamily, 359
\DeclareFontshape, 359
\DeclareFontsubstitution, 404
\DeclareGraphicsExtensions
(graphics), 39, 40
\DeclareGraphicsRule
(graphics), 34, 39, 40, 41
\DeclareMathSymbol, 404
\declaresize (fontinst), 493
\DeclareSymbolFont, 404
\DeclareSymbolFontAlphabet, 404
defaultdr (boxes), 69
defaultdr (boxes), 69
defaultfont (METAPOST), 64, 65, 73, 79, 81
defaultscale (METAPOST), 64, 66, 72, 73, 79, 81–83
define (pic), 23
\definechemical (m-chemic), 224
\definecolor (color), 317, 319, 320, 322, 330, 331, 334, 337
\degrees (pstricks), 99, 121, 135

deletefile (PostScript), 445
\DEP (MusiXTeX), 257
\depth (fontinst), 492
\depth length, 44
depth key (graphicx), 33
\descender (fontinst), 489
DESIGNSIZE (PL/VPL), 366
\designsize (fontinst), 489
DeviceGray (PostScript), 445
DeviceRGB (PostScript), 445
\DFP (circ), 245
\dhqu (MusiXTeX), 257
\diam (private), 302, 305
diamond notation (pstricks), 106
diamond* notation (pstricks), 106
\diamondsuit, 302
\dianode (pst-node), 116, 160, 162, 163
dimen key (pstricks), 99, 100, 144, 156
\ding (pifont), 323
dir (META), 56, 70–73
direction (META), 56
\div (fontinst), 493
\dontindentwhite (bg), 302
\dontshowcube (bg), 300, 302
\dontshowmoves (bg), 302
\dontshownumbers (bg), 300
\doqu (MusiXTeX), 257
dotangle key (pstricks), 158
dotlabel (METAPOST), 64, 65
dotlabels (METAPOST), 65, 66
\dotlineset (mfpic), 88
\dotnode (pst-node), 117, 159, 162, 163
dotscale key (pstricks), 105, 158
dotsep key (pstricks), 157
dotsize key (pstricks), 144, 158
dotstyle key (pstricks), 105, 108, 133, 144, 158
\dotted (mfpic), 86, 88
\dottedline (epic), 211
doublecolor key (pstricks), 157

doubleline key (pstricks), 103, 108, 157
 \doublelerulesepcolor (colortbl), 329
 doublesep key (pstricks), 103, 157
 \Down (private), 150, 151
 down (pic), 22, 23
 \downbow (MusixTeX), 257
 \downbow (MfP), 271
 \DOWNtext (crosswd), 308
 \downtrio (MusixTeX), 257
 \dqu (MusixTeX), 257
 DR notation (m-chemic), 225
 draft key (graphicx), 33, 34
 \draw (mfpic), 88, 90
 draw (META), 55, 56, 57, 70, 74
 draw (mftoops), 93
 draw_C (mftoops), 93, 94
 drawarrow (METAPOST), 70, 71-73
 drawboxed (boxes), 69, 70-72
 drawboxes (boxes), 69, 71
 drawdblarrow (METAPOST), 70
 drawunboxed (boxes), 69, 70, 73
 \drop (Xy-pic), 197, 198
 \drumclef (MusixTeX), 257
 \ds (MusixTeX), 257, 259
 \dtetrastereo (aliphath), 219, 220
 \Dtext (circ), 248
 \Dtrigonal (aliphath), 219
 \dtrigonal (aliphath), 219
 \duevolte (MusixTeX), 257
 \duration (MfP), 270, 271-273
 Dx key (pst-plot), 166
 dx key (pst-plot), 166
 Dy key (pst-plot), 134, 166
 dy key (pst-plot), 134, 166
 E: notation (abc), 265, 267
 EB notation (m-chemic), 225
 edge key (pst-tree), 128, 132, 164
 \elemskip length (MusixTeX), 260, 263, 265, 267
 \elemskip length (MusixTeX), 275

\ellipse (Xy-pic), 187, 195, 196, 197
 \ellipse (mfpic), 87, 92
 ellipse (pic), 22
 embedangle key (pst-3d), 166
 emode key (pst-node), 162
 \emph, 364
 \empty, 298
 Encoding (PostScript), 371
 \encoding (fontinst), 487
 end (META), 63
 end-hook (PostScript), 429, 433
 endchar (META), 60, 63
 \endencoding (fontinst), 487
 \endextract (MusixTeX), 259, 261
 endfig (METAPOST), 63
 endfor (META), 55, 57
 endgraph (graph), 74
 \endinstallfonts (fontinst), 394, 400
 \endmetrics (fontinst), 398, 488
 endmulti (PostScript), 433
 \endpiece (MusixTeX), 259
 \endpscharclip (pst-char), 113, 155
 \endpsclip (pst-text), 100, 101
 \endpsTree (pst-tree), 163
 \endresetglyph (fontinst), 398
 \endsetglyph (fontinst), 397, 489
 \endsetslot (fontinst), 398, 400, 488
 \endsprite (sprite), 13
 \endvarchar (fontinst), 488
 \endxy (Xy-pic), 169, 180
 \enotes (MusixTeX), 256, 259-261
 \entrymodifiers (Xy-pic), 182
 enumerate environment, 339
 eop-hook (PostScript), 429
 EPS_mode_setup (mftoops), 93
 \equal (chess), 283
 EQUILIBRIUM notation (m-chemic), 223, 227
 ER notation (m-chemic), 225
 etex (METAPOST), 64, 65-67, 72, 74, 75, 78, 80

\ethanestereo (aliphath), 219, 220
 \ethylene (aliphath), 219
 \Ethylenev (aliphath), 219
 \ethylenev (aliphath), 219
 ext key (graphicx), 33, 34
 ExtendFont (PostScript), 370, 428
 \extrarowheight length (array), 327, 328
 \F (MfP), 272
 fansize key (pst-tree), 164
 \fboxrule length, 322
 \fboxrule length (color), 322
 \fboxsep length, 30, 322, 335
 \fboxsep length (color), 322, 335
 \fcolorbox (color), 322
 \fermatadown (MusixTeX), 257
 \Fermataup (MusixTeX), 257
 \fermataup (MusixTeX), 257
 \fff (circ), 245
 \fileplot (pst-plot), 132, 152, 165
 \fill (pstricks), 143, 145
 fill (META), 57, 70, 74
 fill (mftoops), 93
 fill_C (mftoops), 93, 94
 fillcolor key (pstricks), 104, 105, 107, 112, 113, 124, 134, 135, 142, 145, 156
 fillstyle key (pstricks), 104, 105, 107, 108, 112, 113, 124, 134, 142, 145, 157
 find_BB (mftoops), 93
 FIVE notation (m-chemic), 223
 \fiveheterov (hetarom), 216
 \fiverm, 16
 \fiveunitv (hetarom), 220
 \fiveunitvi (hetarom), 220
 fix_drawn_cmyk (mftoops), 94
 fix_fill_cmyk (mftoops), 94
 fix_line_width (mftoops), 94
 \fla (MusixTeX), 258
 \flageolett (MusixTeX), 257
 floor (META), 55
 \fmf (feynmf), 229, 232, 237-239

\fmfblob (feynmf), 234
 \fmfblobn (feynmf), 234
 \fmfbottom (feynmf), 232
 \fmfbottomn (feynmf), 232
 \fmfcmd (feynmf), 239
 \fmfcurved (feynmf), 232
 \fmfcyclen (feynmf), 232
 \fmfdot (feynmf), 229, 234, 237, 338
 \fmfdotn (feynmf), 234
 fmffile environment (feynmf), 230
 \fmffixed (feynmf), 237, 238
 \fmffreeze (feynmf), 234, 238, 239
 fmffgraph environment (feynmf), 236, 238
 fmffgraph* environment (feynmf), 229, 236, 237
 \fmfi (feynmf), 238, 239
 \fmfiequ (feynmf), 238
 \fmfipair (feynmf), 238
 \fmfipath (feynmf), 238
 \fmfiv (feynmf), 238, 239
 \fmflabel (feynmf), 237
 \fmfleft (feynmf), 229, 232, 238
 \fmfleftn (feynmf), 232, 236, 237
 \fmfn (feynmf), 232
 \fmfpn (feynmf), 234
 \fmfpoly (feynmf), 234
 \fmfrcyclen (feynmf), 232
 \fmfright (feynmf), 229, 232, 238
 \fmfrightn (feynmf), 232, 236, 237
 \fmfstraight (feynmf), 232
 \fmfsurround (feynmf), 232
 \fmftop (feynmf), 232
 \fmftopn (feynmf), 232
 \fmfv (feynmf), 234
 \fmfvn (feynmf), 234
 \fmfpoly (feynmf), 234
 \fnode (pst-node), 117, 159, 162, 164
 \font, 289, 292
 FONTAT (PL/VPL), 368
 FONTDIMEN (PL/VPL), 365
 \fontdimen, 365

\fontdimen(n) (fontinst), 488
 FONTDSIZE (PL/VPL), 368
 FontInfo (PostScript), 373
 \fontinst (fontinst), 492
 FONTNAME (PL/VPL), 368
 for (META), 55, 57
 format (graph), 75, 76
 FOUR notation (m-chemic), 223
 fourhetero (hetarom), 216
 frame (graph), 75, 76, 77, 79-82
 frame (grap), 24, 25
 framearc key (pstricks), 104, 105, 123, 125, 131, 157
 framesep key (pstricks), 124, 158
 \frm (Xy-pic), 172, 176, 177, 202
 \from (circ), 246
 from (pic), 23
 \fromafz (fontinst), 395, 400
 \frompin (circ), 247
 \fullboard (bg), 300
 fullcircle (META), 67, 81
 \fullincr (bg), 302
 FullName (PostScript), 371
 \function (mfpic), 89, 90
 \furanose (hcycle), 218
 game environment (bg), 299, 300, 301
 gameone environment (bdfchess), 284, 285-287
 gametwo environment (bdfchess), 284, 285-287
 gangle key (pstricks), 158
 \gclear (mfpic), 88, 90
 gdata (graph), 76, 77, 79, 81-83
 gdotlabel (graph), 74
 gdraw (graph), 72, 74, 76-78, 80-82
 gdrawarrow (graph), 74
 gdrawdblarrow (graph), 74
 \generalmeter (MusixTeX), 261
 \generalsignature (MusixTeX), 258, 261
 \geometricsskip scale (MusixTeX), 260

\gfill (mfpic), 86, 88, 90
 gfill (graph), 75-81, 83
 gis notation (MfP), 270
 GIVES notation (m-chemic), 227
 glabel (graph), 72, 74, 78-83
 \glyph (fontinst), 397, 489
 \glyphrule (fontinst), 490
 \glyphspecial (fontinst), 490
 \glyphwarning (fontinst), 490
 \GND (circ), 247
 \gofontsize (go), 296, 297, 298
 \gosign (private), 296
 \grace (MfP), 271, 272
 gradangle key (pst-grad), 105, 157
 gradbegin key (pst-grad), 105, 108, 157
 gradend key (pst-grad), 105, 108, 157
 gradlines key (pst-grad), 157
 gradmidpoint key (pst-grad), 157
 \graph (gpic), 251
 \graphicspath (graphics), 39
 gray color model (color), 317
 \grcl (MusixTeX), 257
 \grcu (MusixTeX), 257
 green (METAPOST), 64
 green notation (color), 318, 319
 \gregorianCclef (MusixTeX), 257
 \gregorianFclef (MusixTeX), 257
 \grestore (pstricks), 143, 145
 grid (graph), 75
 gridcolor key (pstricks), 158
 griddots key (pstricks), 159
 gridlabelcolor key (pstricks), 159
 gridlabels key (pstricks), 159
 gridwidth key (pstricks), 158
 \gsave (pstricks), 143, 145
 \H, 207
 \H (chemsym), 207
 \h (chemsym), 207
 H: notation (abc), 266
 \ha (MusixTeX), 258
 \halfboard (bg), 300

\halfinacr (bg), 302
 \hand (bridge), 303–306
 \hand (private), 302
 \hanthracenr (carom), 214
 \hashlen length (mfpic), 87
 \hatch (mfpic), 87, 88
 hatchangle key (pstricks), 105, 157
 hatchcolor key (pstricks), 105, 157
 hatchsep key (pstricks), 157
 hatchspace length (mfpic), 87, 88
 hatchwidth key (pstricks), 157
 \HBLens (circ), 246
 \hbox, 321
 \hcap (Xy-pic), 201
 \headleft (M_P), 271
 \headlen length (mfpic), 87, 88
 \headright (M_P), 271, 272, 273
 \headshape (mfpic), 88
 \heart (private), 302, 305, 306
 \heartsuit, 302
 \height (fontinst), 492
 \height length, 44
 \height (pic), 23
 height key (graphicx), 33, 35–37
 \hhline (hhline), 336
 hiresbb key (graphicx), 33
 HiResBoundingBox (PostScript), 28, 33
 \hl (Mus_XTeX), 257, 258
 \Hpause (Mus_XTeX), 257
 \hpause (Mus_XTeX), 257, 259
 \hpausep (Mus_XTeX), 257
 \hphenanthrenev (carom), 214
 \HR (private), 30, 43
 \hs (Mus_XTeX), 257
 \HSLens (circ), 246
 ht (grap), 25
 \htopin (circ), 246, 247
 \hu (Mus_XTeX), 257, 258, 259
 \HVLens (circ), 246
 \I (circ), 244
 i notation (pic), 22

\ibbu (Mus_XTeX), 262
 \ibl (Mus_XTeX), 261, 262
 \ibu (Mus_XTeX), 261, 262
 \ifcase, 151
 \iffalse (fontinst), 491, 492
 \ifiscommand (fontinst), 491
 \ifisdim (fontinst), 492
 \ifisglyph (fontinst), 492
 \ifisint (fontinst), 492
 \ifisstr (fontinst), 492
 \ifthenelse (ifthen), 198
 \iftrue (fontinst), 491, 492
 image (METAPOST), 79, 81
 \imidazolev (hetarom), 217
 \imidazolevi (hetarom), 217
 \includegraphics (graphics), 27, 29, 30, 31, 39–41, 342, 461
 \includegraphics (graphicx), 32, 34, 38
 \includegraphics* (graphics), 30
 \includegraphics* (graphicx), 32
 \indanehi (lowcycle), 216
 \indanev (lowcycle), 216
 \indentwhite (bg), 302
 \indolev (hetarom), 217
 \indolevi (hetarom), 217
 \indolizinev (hetarom), 217
 \indolizinevi (hetarom), 217
 \Inductor (private), 150
 infont (METAPOST), 79, 81
 \inidiagram (go), 296, 297
 \inifullldiagram (go), 296, 298
 init_numbers (graph), 75
 input (METAPOST), 68
 \inputetx (fontinst), 488
 \inputmtx (fontinst), 489
 \installfamily (fontinst), 394, 399, 401
 \installfont (fontinst), 394, 395, 396, 399, 400, 402
 \installfonts (fontinst), 394, 399
 \instrumentnumber (Mus_XTeX), 261

\int (fontinst), 397, 488, 492
 invis (pic), 23
 \islurd (Mus_XTeX), 262
 \isluru (Mus_XTeX), 261, 262
 \isobenzofuranv (hetarom), 217
 \isobenzofuranevi (hetarom), 217
 \isoindolev (hetarom), 210, 217
 \isoindolevi (hetarom), 217
 \isoquinolinev (hetarom), 217
 \isoquinolinevi (hetarom), 217
 \isoxazolev (hetarom), 217
 \isoxazolevi (hetarom), 217
 \italic (fontinst), 493
 \italicslant (fontinst), 489
 itemize environment, 339
 itick (graph), 75, 76
 \Ivar (circ), 244
 \JKMSFF (circ), 245
 \junction (circ), 245
 K: notation (abc), 265, 266
 keepaspectratio key (graphicx), 33, 35, 36, 38
 \kemtkn (chemsym), 207
 \kerning (fontinst), 397, 493
 \key (M_P), 270, 271–273
 \key (chesskey), 285
 \knotholesize (Xy-pic), 202, 203
 \kqu (Mus_XTeX), 257
 KRN (PL/VPL), 366
 L notation (Xy-pic), 172
 \L (circ), 244
 \l (Mus_XTeX), 257
 \l... (Mus_XTeX), 259
 L: notation (abc), 266, 267
 \La (circ), 244
 LABEL (PL/VPL), 366
 label (METAPOST), 64, 65, 71, 75
 label (grap), 24
 labelarrow (METAPOST), 72
 labeloffset (METAPOST), 64, 65
 labels key (pst-plot), 166

\labelsep length, 102
 labelsep key (pstricks), 123, 127, 158
 \labelstyle (Xy-pic), 189, 199–203
 landscape environment (lscape), 51
 \largeboard (cchess), 294
 \larw (timing), 243
 \Laser (circ), 246, 248
 \latinfamily (fontinst), 393, 394
 \latticeA (Xy-pic), 198
 \latticeB (Xy-pic), 198
 \latticebody (Xy-pic), 197, 198
 \latticeX (Xy-pic), 198
 \latticeY (Xy-pic), 198
 Lc notation (Xy-pic), 176
 \lc (fontinst), 399
 \lclig (fontinst), 399
 \lclosed (mfpic), 88
 \LEE (circ), 244
 \Left (private), 150
 left (pic), 23
 \leftrepeat (Mus_XTeX), 257
 \leifrightrepeat (Mus_XTeX), 257
 length (META), 55, 71, 73
 \letter (go), 296, 297, 298
 \letterspacing (fontinst), 488
 levelsep key (pst-tree), 126, 129, 131, 164
 lft notation (METAPOST), 64
 \lifthpause (Mus_XTeX), 257
 \liftpause (Mus_XTeX), 257
 liftpen key (pstricks), 142, 145
 LIG (PL/VPL), 366, 392
 \ligature (fontinst), 398
 \lightershade (mfpic), 88
 LIGKERN notation (afm2tfm), 392
 LIGTABLE (PL/VPL), 366, 367
 line (pic), 22, 23
 linearc key (pstricks), 103, 157
 linecolor key (pstricks), 98, 100, 104–109, 112, 113, 123, 128, 133, 134, 136, 144, 156

\lines (mfpic), 86, 87, 91
 linestyle key (pstricks), 101, 103, 104, 108, 110, 129, 132, 133, 142, 157
 \linethickness (timing), 243
 \lineto (pstricks), 145
 \linewidth length, 38
 linewidth key (pstricks), 98, 100, 103, 104, 106, 108, 112, 142, 144, 156, 157
 \listofslides (slidesec), 343
 \listplot (pst-plot), 132, 165
 llft notation (METAPOST), 64
 \lmoiety (chemstr), 212, 215
 \ln (circ), 245
 longtable environment (longtable), 326
 loopsize key (pst-node), 119, 162
 \lppz (Mus_XTeX), 257
 \lpz (Mus_XTeX), 257
 \lpzst (Mus_XTeX), 257
 lrt notation (METAPOST), 64
 \lsf (Mus_XTeX), 257
 \lsfz (Mus_XTeX), 257
 \lsqu (Mus_XTeX), 257
 \lst (Mus_XTeX), 257
 \ltrigona (aliphat), 219
 M: notation (abc), 265, 266
 magenta notation (color), 318, 319
 mainline environment (private), 282, 283
 makepen (META), 54
 \makeslideheading (seminar), 343
 MAP (PL/VPL), 368
 MAPFONT (PL/VPL), 368
 \mathbb (private), 404
 \mathrm, 206
 mcol key (pst-node), 162
 \meter (M_P), 272, 273
 \meterC (Mus_XTeX), 257
 \meterfrac (Mus_XTeX), 261
 \meterplus (Mus_XTeX), 257
 \metrics (fontinst), 396, 488

\metron (Mus_XTeX), 257
 mfpic environment (mfpic), 85, 91
 \mfpicunit length (mfpic), 85, 87
 \middlecube (bg), 299, 300
 \minimumkern (fontinst), 489
 \Mirror (circ), 248
 mnode key (pst-node), 123, 124, 162
 mnodesize key (pst-node), 163
 mode (META), 60
 mode_setup (META), 61
 \monowidth (fontinst), 489
 \Mordent (Mus_XTeX), 257
 \mordent (Mus_XTeX), 257
 MOV notation (m-chemic), 225
 \move (bg), 300, 301, 302
 \move (chess), 279, 280, 281, 284, 286
 move (pic), 23
 movecount counter (chess), 279, 281
 \moverel (circ), 246, 247
 MOVERIGHT (PL/VPL), 368
 \mover (fontinst), 397, 490
 \moveto (pstricks), 143
 \moveup (fontinst), 490
 \mrestore (pstricks), 143
 \msave (pstricks), 143
 \mul (fontinst), 493
 multi (PostScript), 433
 \multicolumn, 305
 \multido (multido), 121, 139, 146
 \multido (pstricks), 139
 \multips (pstricks), 139
 \multirput (pstricks), 139
 music environment (Mus_XTeX), 259, 260, 261
 name key (pst-node), 124, 125, 128, 162
 named color model (color), 317, 318, 320
 \naphdrh (carom), 214
 \naphdrv (carom), 214
 \naput (pst-node), 114, 123, 131, 161
 natheight key (graphicx), 33
 natwidth key (graphicx), 33

\ninput (pst-node), 114, 123, 131, 132, 161
 \ncangle (pst-node), 118, 119, 125, 128, 129, 130, 131, 160
 \ncangles (pst-node), 118, 119, 160
 \ncarc (pst-node), 114, 118, 160, 161
 \ncarcbox (pst-node), 162
 \ncbar (pst-node), 118, 132, 160
 \ncbox (pst-node), 162
 \nccircle (pst-node), 120, 160
 \nccoil (pst-coil), 120, 160
 \nccurve (pst-node), 114, 118, 123, 160, 162
 \ncdiag (pst-node), 118, 119, 160
 \ncdiagg (pst-node), 118, 119, 160
 \ncline (pst-node), 160
 \ncline (pst-node), 115, 116, 122, 123, 160
 \nclloop (pst-node), 118, 119, 160, 162
 \ncput (pst-node), 161
 \ncurv key (pst-node), 123, 162
 \ncurVA key (pst-node), 162
 \ncurVB key (pst-node), 162
 \nczigzag (pst-coil), 120, 161
 \needsfontinstversion (fontinst), 492
 \neg (fontinst), 493
 \newcolumn type (array), 327
 \newdir (Xy-pic), 170, 181, 182
 \newfont, 289
 \newgame (chess), 279, 280
 \newgraphescape (Xy-pic), 185, 186, 187
 NEWMAN notation (m-chemic), 223
 \newpagestyle (seminar), 344
 \newpath (pstricks), 143
 \newpsobject (pstricks), 140
 \newpsstyle (pstricks), 142
 \newsldframe (seminar), 344, 345
 \nextlarger (fontinst), 488
 \nextslot (fontinst), 398, 400, 488
 \nfet (circ), 244

\nl (circ), 247, 248
 nochess environment (chess), 282
 nodealign key (pst-node), 162
 nodesep key (pst-node), 161
 nodesep key (pst-tree), 126, 128, 130
 nodesepA key (pst-node), 161
 nodesepB key (pst-node), 161
 \nonaheterov (hetarom), 210, 217
 \normal (MfP), 271
 normal key (pst-3d), 136, 137, 166
 \normalboard (bg), 300, 301
 \normalboard (cchess), 294
 normaldeviate (META), 55
 NormalizedDesignVector (PostScript), 406, 409
 note environment (seminar), 341
 \NOTes (MusixTeX), 260
 \NOTes (MusixTeX), 260
 \Notes (MusixTeX), 256, 260, 261
 \notes (MusixTeX), 256, 260, 261
 \noteskip length (MusixTeX), 260
 \noteslides (seminar), 341
 \NOTesp (MusixTeX), 260
 \NOTesp (MusixTeX), 260
 \Notesp (MusixTeX), 260
 \notesp (MusixTeX), 260
 \nprn (circ), 244, 247
 npos key (pst-node), 131, 162
 \nput (pst-node), 161
 \NRFSS (circ), 245
 nrot key (pst-node), 132, 162
 nullpen (META), 54
 numeric (META), 54
 \nv (circ), 245
 \nvms (circ), 244
 \O, 207
 \O (chemsym), 207
 o notation (pstricks), 106
 o- notation (pstricks), 103
 o-o notation (pstricks), 103
 O: notation (abc), 266

\oa (circ), 248
 \objectmargin length (Xy-pic), 191
 \objectstyle (Xy-pic), 189, 192, 194, 199, 202, 203
 \octfindown (MusixTeX), 257
 \octfinup (MusixTeX), 257
 OFF notation (m-chemic), 227
 offset key (pst-node), 123, 161
 offsetA key (pst-tree), 131
 \OH (chemsym), 207
 \oldGclef (MusixTeX), 257
 \OM (circ), 248
 \omit (Xy-pic), 188, 189, 190
 ONE notation (m-chemic), 223, 227
 \onlyslides (seminar), 341
 \OO (chemsym), 207
 oo- notation (pstricks), 103
 oo-oo notation (pstricks), 103
 \opengraphsfile (mfpic), 85
 oplus notation (pstricks), 106
 orient key (graphicx), 39
 origin (META), 76, 77
 origin key (graphicx), 33, 46, 47
 origin key (pstricks), 156
 otick (graph), 75, 76, 82
 otimes notation (pstricks), 106
 OUT notation (METAPOST), 74
 \ovalnode (pst-node), 116, 160, 162, 164
 Ox key (pst-plot), 166
 \oxazolev (hetarom), 217
 \oxazolevi (hetarom), 217
 \oxqu (MusixTeX), 257
 Oy key (pst-plot), 166
 \P, 207
 \pagecolor (color), 321, 345
 \pagestyle (seminar), 340
 pair (META), 54, 57, 64
 \paperheight length, 343
 \paperwidth length, 343
 \parabola (pstricks), 105, 106, 154
 \parafcn (mfpic), 89, 92
 \parametricplot (pst-plot), 165

\parbox, 42, 44, 45
 path (META), 54, 56, 57
 \PAUSE (MusixTeX), 257
 \PAUSE (MusixTeX), 257
 \pause (MusixTeX), 257, 259
 \pausep (MusixTeX), 257
 \pcangle (pst-node), 163
 \pcarc (pst-node), 121, 163
 \pcbar (pst-node), 163
 \pccoil (pst-coil), 163
 \pccurve (pst-node), 163
 \pcdiag (pst-node), 163
 \pcline (pst-node), 163
 \pcloop (pst-node), 163
 \pczigzag (pst-coil), 163
 pdfmark (PostScript), 415
 \PED (MusixTeX), 257
 \pen (mfpic), 86, 88, 90
 pen (META), 54
 pencircle (META), 54, 57, 73, 78
 pensquare (META), 82
 pentagon notation (pstricks), 106
 pentagon* notation (pstricks), 106
 \pfet (circ), 244
 \phenanthrenev (carom), 214
 Piatolist environment (pifont), 358
 pic (boxes), 69, 71, 73
 pickup (META), 57, 73, 78, 82
 picture (META), 54, 66, 67
 picture environment, 10, 17, 18, 22, 208, 220, 222, 228, 236, 239
 \piece (cchess), 293, 294, 295
 \Pifill (pifont), 357
 \Piline (pifont), 357
 Pilist environment (pifont), 358
 \Pinhole (circ), 246, 248
 \Pisymbol (pifont), 357, 359
 plot (graph), 74
 plotpoints key (pst-plot), 133, 166
 plotstyle key (pst-plot), 133, 134, 166

\plrfcn (mfpic), 89
 \plrregion (mfpic), 89
 PLUS notation (m-chemic), 227
 \ply (chess), 279, 280, 281, 284
 \PM (circ), 246
 \pnode (pst-node), 115, 117, 159, 162, 164
 \pnp (circ), 244
 \point (mfpic), 87
 point (META), 71, 72, 73
 pointfilled boolean (mfpic), 87
 \pointsizelength (mfpic), 87
 \Polar (circ), 246, 248
 \polygon (mfpic), 87
 \polylines (mfpic), 87
 POP (PL/VPL), 368
 \pop (fontinst), 491
 \POS (Xy-pic), 180, 183, 185, 187
 \pos (go), 296, 297, 298
 position environment (bg), 299, 300, 302
 position environment (cchess), 293, 294, 295
 position environment (chess), 279, 281
 \postcard (bdfchess), 285, 286, 287
 \postcardaddress (bdfchess), 285, 286, 287
 \postmove (bdfchess), 284, 286, 287
 \postply (bdfchess), 284, 286, 287
 \PP (chemsym), 207
 \Pr, 207
 \pr (chemsym), 207
 \printboard (bg), 300, 301
 \printlandscape (seminar), 342
 \ProcessVector (private), 152
 prologues (METAPOST), 67, 68
 \psaddtolength (pstricks), 98
 \psarc (pstricks), 105, 106, 109, 142, 154, 158
 \psarcn (pstricks), 109, 154
 \psaxes (pst-plot), 133, 134, 165
 \psbezier (pstricks), 106, 155
 \psboxfill (pst-fill), 96
 \psccurve (pstricks), 155
 \pscharclip (pst-char), 113, 155, 156
 \pscharclip (pst-text), 154
 \pscharpath (pst-char), 112, 154, 155
 \pscicle (pstricks), 104, 110, 155, 156
 \pscirclebox (pstricks), 102, 108, 121, 155, 160
 \psclip (pst-text), 100, 101
 \pscoil (pst-coil), 155
 \pscoil (pst-coil), 154, 155
 \pscurve (pstricks), 105, 109, 145, 155
 \pscustom (pstricks), 142
 \psdblframebox (pstricks), 108, 155
 \psdiabox (pstricks), 108, 155, 160
 \psdiamond (pstricks), 101, 104, 155
 \psdots (pstricks), 105, 155, 159
 \psecurve (pstricks), 155
 \psedge (pst-tree), 127, 128, 129, 131
 \psellipse (pstricks), 105, 155
 \psfrag (PSfrag), 460, 461
 \psframe (pstricks), 100, 104, 155, 159
 \psframebox (pstricks), 101, 107, 108, 117, 124, 155
 \psgrid (pstricks), 100, 136, 155
 \psline (pstricks), 98, 99, 101, 102, 103, 142, 155, 156
 psmatrix environment (pst-node), 121, 122-125
 \psovalbox (pstricks), 108, 155, 160
 \pspicture environment (pstricks), 97, 100, 102, 155
 \psplot (pst-plot), 133, 165
 \pspolygon (pstricks), 104, 156
 \psset (pstricks), 98, 99-101
 \pssetlength (pstricks), 98, 152
 \psshadow (pst-3d), 136, 166
 \psshadowbox (pstricks), 107, 156

\psspan (pst-node), 122
 \pstdriver (pstricks), 154
 \psttextpath (pst-text), 109, 112, 154, 156
 \pstheader (pstricks), 154
 \pstilt (pst-3d), 136, 166
 \pstree (pst-tree), 163
 \pstree (pst-tree), 125, 126–132, 163
 \pstriangle (pstricks), 104, 156
 \pstribox (pstricks), 108, 156, 160
 \pstrotate (pstricks), 154
 \PSTtoEPS (pst-eps), 132, 152
 \pstunit (pstricks), 154
 \pstVerb (pstricks), 154
 \pstverb (pstricks), 154
 \pstverbscale (pstricks), 154
 \pswedge (pstricks), 105, 135, 156
 \psxlabel (pst-plot), 165
 \psylabel (pst-plot), 165
 \pszigzag (pstricks), 152, 156
 \pt (MusikTeX), 259
 \pteridine (hetarom), 217
 \pteridinevi (hetarom), 217
 \purine (hetarom), 210, 217
 \purinevi (hetarom), 217
 PUSH (PL/VPL), 368
 \push (fontinst), 491
 Puzzle environment (cwpuzzle), 309
 PuzzleClues environment (cwpuzzle), 310
 \PuzzleLetters (cwpuzzle), 310
 \PuzzleNumbers (cwpuzzle), 310
 \PuzzleSolution (cwpuzzle), 310
 \PuzzleUnsolved (cwpuzzle), 310
 PuzzleWords environment (cwpuzzle), 310
 \pvmos (circ), 244
 \pyranose (hcycle), 218
 \pyrazine (hetarom), 214, 217
 \pyrazole (hetarom), 217
 \pyrazolevi (hetarom), 217

\pyridazine (hetarom), 217
 \pyridazinevi (hetarom), 217
 \pyridine (hetarom), 217, 220
 \pyridinevi (hetarom), 217
 \pyrimidine (hetarom), 217
 \pyrimidinevi (hetarom), 217
 \pyrrole (hetarom), 217
 \pyrrolevi (hetarom), 217
 \Q (circ), 244
 \qa (MusikTeX), 258, 259, 260
 \qb (MusikTeX), 261, 262
 \qdisk (pstricks), 156
 \ql (MusikTeX), 257, 258, 261, 262
 \qline (pstricks), 156
 \qp (MusikTeX), 257, 259
 \qq (MusikTeX), 257
 \qs (MusikTeX), 257
 \qu (MusikTeX), 257, 258, 259–262
 \quinazoline (hetarom), 217
 \quinazolinevi (hetarom), 217
 \quinoline (hetarom), 217
 \quinolinevi (hetarom), 217
 \quinoxaline (hetarom), 217
 quit (PostScript), 443
 quote environment, 38
 \qupp (MusikTeX), 257
 R notation (Xy-pic), 172
 R notation (m-chem), 223, 225
 \R (circ), 244, 247
 r notation ($\mathbb{R}$), 271, 272
 \r... (MusikTeX), 259
 \radians (pstricks), 99
 radius key (pst-node), 116, 127
 rand (pic), 22
 \rarw (timing), 243
 \rawboard (bg), 300, 301
 rboxit (rboxes), 69
 rbracketlength key (pstricks), 158
 \rcurveto (pstricks), 145
 \Re, 207
 \re (chemsym), 207
 read key (graphicx), 33, 34, 39

\readdata (pst-plot), 132, 134, 165
 \receiver (bdfchess), 285, 286, 287
 \rect (mfpic), 87
 rect (pic), 23
 \red (METAPOST), 64
 \red notation (color), 318, 319
 \ReEncodeFont (PostScript), 291, 370, 428
 \reencodefont (fontinst), 395, 399, 400
 ref key (pst-node), 124, 128, 162
 \reflectbox (graphics), 42
 \reflectedabout (META), 66
 \relax (fontinst), 487, 488
 \renamefile (PostScript), 445
 \renewfont (private), 289
 \resetcommand (fontinst), 492
 \resetdepth (fontinst), 398, 491
 \resetdim (fontinst), 492
 \resetglyph (fontinst), 398, 489, 491
 \resetheight (fontinst), 491
 \resetint (fontinst), 492
 \resetitalic (fontinst), 491
 \resetstr (fontinst), 492
 \resetwidth (fontinst), 491
 \Resistor (private), 150
 \resizebox (graphics), 31, 34, 43, 44, 99
 \resizebox* (graphics), 43, 44
 Resolution (PostScript), 426
 \restore (Xy-pic), 183, 184, 185, 187, 202
 \reverse (mfpic), 88
 \reverseallabreve (MusikTeX), 257
 \reverseC (MusikTeX), 257
 \rgb color model (color), 317
 \rh (MusikTeX), 259
 \Right (private), 150
 \right (pic), 22, 23
 \rightrepeat (MusikTeX), 257
 \rlineto (pstricks), 145
 \rmdefault, 354, 359, 403

\rmoiety (chemstr), 212
 \Rnode (pst-node), 115, 159, 162, 163
 \rnode (pst-node), 114, 115, 117–119, 159, 162, 164
 \roqu (MusikTeX), 257
 \ROT notation (m-chem), 225, 226
 \rotate (pstricks), 143
 \rotatebox (graphics), 27, 41, 44, 130
 \rotatebox (graphicx), 33, 34, 39, 46, 47
 \rotated (META), 56, 57, 67, 78–81
 \rotatedabout (META), 66
 \rotatedaround (META), 57
 \rotatepath (mfpic), 89
 \rowcolor (colortbl), 328, 334, 336, 337
 \rowsep key (pst-node), 122, 123, 125, 163
 \rput (pstricks), 101, 102, 110, 112, 114, 116, 139, 148
 \rq (MusikTeX), 261
 \rrtcell (Xy-pic), 189, 190
 \rsqu (MusikTeX), 257
 \rt notation (METAPOST), 64
 \rtrigonal (aliphat), 219
 \rtwocell (Xy-pic), 188, 189
 \runit key (pstricks), 99, 156
 \Rvar (circ), 244
 \RZ notation (m-chem), 223, 224, 225
 S notation (m-chem), 225
 \S, 207
 \S (circ), 244
 s notation ($\mathbb{S}$), 271, 272
 S: notation (abc), 266
 \save (Xy-pic), 183, 184, 185, 187, 202
 \savedata (pst-plot), 132, 165
 \SB environment (chemsym), 207
 \SB notation (m-chem), 225
 \Sb environment (amstex), 207
 \sbox, 321
 \scale (fontinst), 493

\scale (pstricks), 143
 \scale (PostScript), 431
 \scale key (graphicx), 33, 34, 35
 \scalebox (graphics), 31, 34, 42
 \scalebox (pstricks), 153
 \scaled (META), 57, 66, 67, 73, 78, 79, 81, 82
 \scalefont (fontinst), 395
 \scantokens (METAPOST), 77, 78–83
 \sclosed (mfpic), 88
 \scplain (semcolor), 346
 \ScrL (circ), 246, 248
 \ScrR (circ), 246
 \ScrTL (circ), 246
 \ScrTR (circ), 246
 \sDEP (MusikTeX), 257
 \SDict (PostScript), 426
 \sector (mfpic), 87
 \segno (MusikTeX), 257
 \selectdevice (PostScript), 448
 \sender (bdfchess), 285, 286, 287
 \seppawns (chess), 282, 283
 \set_BB (mfptops), 93
 \setclef (MusikTeX), 261
 \setcommand (fontinst), 492
 \setcoords (graph), 76
 \setdim (fontinst), 492
 \setglyph (fontinst), 397, 489
 \setgray (PostScript), 345
 \setint (fontinst), 396, 488, 492
 \setkern (fontinst), 489
 \setkeys (graphicx), 38, 47
 \setkeys (keyval), 38
 \setleftkerning (fontinst), 489
 \setlefttrighkerning (fontinst), 397
 \setpagedevice (PostScript), 431, 458
 \setrange (graph), 76, 78, 79, 82, 83
 \setrawglyph (fontinst), 489
 \setrender (mfpic), 85
 \setrightkerning (fontinst), 489

\setsplot (fontinst), 398, 400, 488
 \setstaffs (MusikTeX), 261
 \setstr (fontinst), 492
 \SetSymbolFont, 404
 \setupchemical (m-chem), 222, 227
 \setvolta (MusikTeX), 257
 \setvoltage (MusikTeX), 257
 \sfdefault, 354
 \sh (MusikTeX), 258
 \shade (mfpic), 86, 87, 89
 \shadespace length (mfpic), 87–89
 \shadow key (pstricks), 108, 112, 157
 \shadowangle key (pstricks), 112, 157
 \shadowcolor key (pstricks), 157
 \shadowsize key (pstricks), 157
 \Shake (MusikTeX), 257
 \shake (MusikTeX), 257
 \ShakeL (MusikTeX), 257
 \Shakene (MusikTeX), 257
 \Shakenw (MusikTeX), 257
 \Shakesw (MusikTeX), 257
 \shift (circ), 246, 247, 248
 \shifted (META), 57, 66
 \shipout, 433
 \shortput key (pst-node), 114, 122, 162
 \showbbox key (pst-tree), 132, 164
 \showboard (chess), 279, 281, 284, 286, 292
 \showboardwithnotation (bdfchess), 284
 \showcube (bg), 299, 300
 \showdiagram (go), 297, 298
 \showfulldiagram (go), 297
 \showgrid (private), 98, 142
 \showinversboard (bdfchess), 284
 \showmoves (bg), 302
 \shownumbers (bg), 299, 300
 \showorigin key (pst-plot), 166
 \showpage (PostScript), 433, 442, 443

showpoints key (pstricks), 103, 105, 106, 156
 sidewaysfigure environment (rotating), 50
 sidewaysstable environment (rotating), 50
 sin (pic), 22
 sind (META), 55
 SIX notation (m-chemic), 223
 \sixheterov (hetarom), 213, 216
 \sixunitv (hetarom), 220
 \sk (MusixTeX), 260
 \skiplevel (pst-tree), 164
 \skiplevels (pst-tree), 164
 SlantFont (PostScript), 428
 \slantfont (fontinst), 395
 \SLens (circ), 246, 248
 \slide (MusixTeX), 257
 slide environment (seminar), 339, 341
 slide* environment (seminar), 339
 \slidechapter (private), 344
 \SlideColors (private), 345
 \Slidecontents (slideseq), 343
 \slideframe (seminar), 339
 \SlideFront (private), 344
 \slideheading (seminar), 343
 \slideheight length (sem-page), 343
 \slideleftmargin (sem-page), 343
 \slideplacement (seminar), 341
 \sliderightmargin (sem-page), 343
 \slidewidth length (sem-page), 343
 \sline (timing), 241, 243
 \SloppyCurves (Xy-pic), 176
 \small (MfP), 271
 \smallaltoclef (MusixTeX), 257
 \smallbassclef (MusixTeX), 257
 \smallboard (bg), 299, 300
 \smallboard (chess), 294, 295
 \smallmusicsize (MusixTeX), 261
 \smalltrebleclef (MusixTeX), 257
 SPACE (PL/VPL), 365
 SPACE notation (m-chemic), 227
 \spade (private), 302, 304–306
 \spadesuit, 302
 SPECIAL (PL/VPL), 369
 \special, 6, 7, 9, 10, 11, 17–20, 22, 23, 26, 41, 228, 250, 251, 294, 367, 420, 422, 424–426, 427, 463, 464, 465, 490
 \special (fontinst), 490
 \SpecialCoor (pstricks), 99, 121, 135
 \sPED (MusixTeX), 257
 spline (pic), 22
 \splinetolerance (Xy-pic), 176
 \sprite (sprite), 12, 13
 sqrt (META), 55
 sqrt (pic), 22
 \squ (MusixTeX), 257
 \square (aliphath), 219
 \square (go), 296, 298
 square notation (pstricks), 106
 square* notation (pstricks), 106
 SR notation (m-chemic), 225
 \SS (chemsym), 207
 StandardEncoding (PostScript), 371
 start-hook (PostScript), 429, 433
 \startchemical (m-chemic), 222, 223, 224, 226, 227
 \startextract (MusixTeX), 259, 261
 \startpiece (MusixTeX), 259
 \stemdown (MfP), 271
 \stemno (MfP), 271
 \stemup (MfP), 271, 272
 step (META), 57
 \steroid (carom), 214, 215
 \steroidchain (carom), 214
 STOP (PL/VPL), 366
 \stopchemical (m-chemic), 222, 223, 224, 226, 227
 string (META), 54

\stroke (pstricks), 143
 style key (pstricks), 142
 SUB notation (m-chemic), 225, 226
 \sub (fontinst), 493
 subgridcolor key (pstricks), 159
 subgriddiv key (pstricks), 100, 159
 subgriddots key (pstricks), 159
 subgridwidth key (pstricks), 159
 \substitutenoisy (fontinst), 395, 399
 \substitutesilent (fontinst), 395, 399
 swapaxes key (pstricks), 156
 \Switch (private), 150
 \symbol, 296, 298
 \symbol (go), 296
 systemdict (PostScript), 443, 444
 T: notation (abc), 266
 tabbing environment, 293, 305
 tabular environment, 9, 49, 304, 306, 326, 329, 339, 410
 tabular* environment, 326
 \takecube (bg), 301
 \taput (pst-node), 114, 127, 161
 TB notation (m-chemic), 225
 tbarsize key (pstricks), 158
 \tbl (MusixTeX), 261, 262
 \tbput (pst-node), 114, 127, 161
 \tbu (MusixTeX), 261, 262
 \TC (pst-tree), 125, 126, 131, 163
 \Tc (pst-tree), 163
 \tcaption (mfpic), 90, 91
 \Tcircle (pst-tree), 131, 163
 \Tdia (pst-tree), 163
 \Tdot (pst-tree), 127, 163
 \tenor (MfP), 269
 tension (META), 56, 71–73
 \tetrahedral (aliphath), 219
 \tetralineh (carom), 214
 \tetralinev (carom), 214
 \tetrastereo (aliphath), 219, 220
 TeXBase1Encoding (PostScript), 361, 369, 388, 428

\textbf, 360
 \textblack (go), 297, 298
 \textcolor (color), 318, 319, 322
 \textheight length, 422
 \textit, 364
 \textmove (bg), 301
 \textpiece (chess), 293
 \textwhite (go), 298
 \textwidth length, 422
 \Tf (pst-tree), 164
 \Tfan (pst-tree), 164
 \tgqu (MusixTeX), 257
 \thechapterheading, 344
 thelabel (METAPOST), 65, 66, 67
 \then (fontinst), 491, 492
 then (pic), 22
 thislevelsep key (pst-tree), 164
 thistreefit key (pst-tree), 164
 thistreenodesize key (pst-tree), 164
 thistreesep key (pst-tree), 164
 \thput (pst-node), 161
 THREE notation (m-chemic), 223
 \Threedput (pst-3d), 136, 137, 138, 166
 \threehetero (hetarom), 213, 216
 thru (grap), 24
 ticks (grap), 24
 ticks key (pst-plot), 166
 ticksize key (pst-plot), 166
 tickstyle key (pst-plot), 134, 166
 \til (timing), 240
 \timadjust (timing), 243
 \timescalefactor (timing), 243
 timing environment (timing), 239, 240, 241
 \timingcounter (timing), 240
 \tin (timing), 240, 241
 \tiny (MfP), 271
 \tlabel (mfpic), 89, 91
 \tlput (pst-node), 114, 130, 161
 \Tn (pst-tree), 131, 163

tndepth key (pst-tree), 164
 tnheight key (pst-tree), 164
 \tnote (timing), 240, 241
 tnpos key (pst-tree), 127, 164
 tnsep key (pst-tree), 164
 to (pic), 23
 \togglenumbers (bg), 300
 top notation (METAPOST), 64
 \topaccent (fontinst), 398
 \totalheight length, 44
 totalheight key (graphicx), 33, 37, 38
 \Toval (pst-tree), 164
 \Tp (pst-tree), 131, 132, 164
 tpos key (pst-node), 162
 \TR (pst-tree), 126, 128, 163
 \Tr (pst-tree), 129, 131, 164
 \tr (MfP), 271
 transform (META), 54
 \transformfont (fontinst), 395, 399, 400
 \translate (pstricks), 143
 translate (PostScript), 431
 \treble (MusixTeX), 261
 \treble (MfP), 269, 272
 \trebleclef (MusixTeX), 257
 treefit key (pst-tree), 164
 treeflip key (pst-tree), 164
 treemode key (pst-tree), 127–129, 131, 165
 treenodes key (pst-tree), 165
 treesep key (pst-tree), 165
 \triangle (go), 296, 297, 298
 triangle notation (pstricks), 106
 triangle* notation (pstricks), 106
 \triazinev (hetarom), 217
 \triazinevi (hetarom), 217
 \Trille (MusixTeX), 257
 \trille (MusixTeX), 257
 trim key (graphicx), 33, 34
 \trinode (pst-node), 116, 117, 160, 162, 164
 \trput (pst-node), 114, 122, 130, 161

true (META), 57
 Tshadowangle key (pst-3d), 136, 166
 Tshadowcolor key (pst-3d), 166
 Tshadowsize key (pst-3d), 136, 166
 \tslur (MusixTeX), 261, 262
 \Tspace (pst-tree), 163
 \ttdefault, 354
 \Ttri (pst-tree), 164
 \turn (MusixTeX), 257
 \turn (mfpic), 90, 92
 \turnradius (Xy-pic), 184
 \turtle (mfpic), 87
 \tvput (pst-node), 161
 \txt (Xy-pic), 173
 type key (graphicx), 33, 34, 41
 U notation (Xy-pic), 172
 \U (circ), 244
 ulft notation (METAPOST), 64
 \underlinethickness (fontinst), 489
 \unfakable (fontinst), 397, 401
 unfill (METAPOST), 57, 79, 81
 uniformdeviate (META), 55
 unit key (pstricks), 99, 100, 136, 144, 156
 \unitlength length, 236, 239
 units key (graphicx), 46, 47
 \unsetcommand (fontinst), 492
 \unsetdim (fontinst), 492
 \unsetglyph (fontinst), 489
 \unsetint (fontinst), 492
 \unsetstr (fontinst), 492
 until (META), 57
 \Up (private), 150
 \upbow (MusixTeX), 257
 \upbow (MfP), 271
 \uppz (MusixTeX), 257
 upto (META), 57
 \uptrio (MusixTeX), 257
 \uput (pstricks), 102, 158, 435
 \upz (MusixTeX), 257
 \upzst (MusixTeX), 257

- urt notation (METAPOST), 64
- \UseAllTwoCells (Xy-pic), 188, 189
- \UseCompositeMaps (Xy-pic), 188
- \usedas (fontinst), 488
- \UseHalfTwoCells (Xy-pic), 188
- userdict (PostScript), 418, 425, 429, 443
- \UseTwoCells (Xy-pic), 188, 190
- \usf (MusixTeX), 257
- \usfz (MusixTeX), 257
- \ust (MusixTeX), 257
- \Utrigonal (aliphat), 219
- \utrigoal (aliphat), 219
- \Uvar (circ), 244
- \V (circ), 244
- \varbot (fontinst), 488
- \varchar (fontinst), 488
- vardef (META), 71, 72
- variation environment (private), 282, 283
- \varmid (fontinst), 488
- \varrep (fontinst), 488
- \vartop (fontinst), 488
- \vcap (Xy-pic), 202
- verbatim environment, 12
- verbatimtex (METAPOST), 66, 67
- viewpoint key (pst-3d), 136, 166
- viewport key (graphicx), 33, 34
- \VLens (circ), 246
- \vloop (Xy-pic), 203, 204
- \vover (Xy-pic), 199, 201, 203
- VTITLE (PL/VPL), 368
- \vtopin (circ), 246, 247
- \wbetter (chess), 282, 283
- \wdecisive (chess), 282, 283
- \wh (MusixTeX), 257, 258, 259
- whatever (META), 76, 78, 82
- \whiledo (ifthen), 113, 324, 325
- \whiledo (mfpic), 92
- \White (chess), 279, 281
- \white (go), 296, 297, 298
- white (METAPOST), 64
- white notation (color), 318, 319
- \whitebar (bg), 300
- \whitecube (bg), 300
- \Whitefalse (chess), 279
- \whiteonmove (bg), 299, 300, 301
- \whitepoint (bg), 299
- \Whitetrue (chess), 279, 281
- \whp (MusixTeX), 257
- wid (grap), 25
- \width (fontinst), 491, 492
- \widthlength, 44
- width (pic), 23
- width key (graphicx), 33, 35–38
- \wire (circ), 246
- \withattack (chess), 282, 283
- withcolor (METAPOST), 66, 67, 73, 74, 75–79, 81, 83
- withdots (METAPOST), 78
- withpen (META), 74
- \Word (cwpuzzle), 310
- write_postamble (mftoepts), 93
- write_preamble (mftoepts), 93
- \wupperhand (chess), 282, 283
- \wwire (circ), 246
- x key (graphicx), 46, 47
- x notation (pstricks), 106
- X: notation (abc), 265, 266
- xbbd key (pst-tree), 129, 165
- xbbh key (pst-tree), 165
- xbb1 key (pst-tree), 132, 165
- xbb2 key (pst-tree), 130, 132, 165
- \xheight (fontinst), 489
- \xmarks (mfpic), 87
- xpart (META), 54
- \xpletstyle (M_P), 272
- \xqu (MusixTeX), 257
- \xscalefont (fontinst), 395
- \xtwoCell (Xy-pic), 188, 189, 190
- \xunderv (Xy-pic), 202, 203, 204
- xunit key (pstricks), 99, 132–134, 156
- \xy (Xy-pic), 169
- xy environment (Xy-pic), 169, 173, 191
- \xybox (Xy-pic), 192, 197, 198
- \xygraph (Xy-pic), 184, 185–187, 201–203
- \xylattice (Xy-pic), 197
- \xymatrix (Xy-pic), 168, 181, 182, 183, 188–190
- \xyoption (Xy-pic), 168
- \xypolygon (Xy-pic), 190, 191–194, 202
- \xypolynome (Xy-pic), 194
- \xypolynode (Xy-pic), 192, 194, 202, 203
- \xypolynum (Xy-pic), 192
- y key (graphicx), 46, 47
- yellow notation (color), 318, 319
- \ymarks (mfpic), 87
- ypart (META), 54
- \yqu (MusixTeX), 257
- yscaled (META), 57
- \yscalefont (fontinst), 395
- yunit key (pstricks), 99, 132–134, 156
- Z notation (m-chemic), 225
- z notation (abc), 266, 268
- \z . . . (MusixTeX), 259
- ZO notation (m-chemic), 225, 227
- \zbreve (MusixTeX), 257
- \ZD (circ), 244
- \zh (MusixTeX), 259
- \zlonga (MusixTeX), 257
- \zmaxima (MusixTeX), 257
- \znotes (MusixTeX), 260
- \zq (MusixTeX), 261
- \zqb (MusixTeX), 261
- \zw (MusixTeX), 259
- \zwq (MusixTeX), 257

TeX Live UNIX TeX on a CD-ROM

The international TeX Users Group (TUG), in collaboration with the TeX users groups of the UK, France, Germany and the Netherlands and many individual members of the TeX community, have produced a CD-ROM containing a ready-to-run release of TeX (Thomas Esser's teTeX based on Karl Berry's Web2c), with most of the macro packages, and many of the tools and fonts, available on the Comprehensive TeX Archive (CTAN). They are arranged according to the TeX Directory Structure (TDS) standard, complete with documentation and source. The CD-ROM—called TeX Live—includes executables for most popular UNIX systems, including Linux, Solaris, AIX, Irix, HP-UX etc., along with ready-to-install packages for DOS, Windows, and Macintosh.

The TeX Live CD-ROM contains all the packages and tools described in *The L^ATeX Companion* and *The L^ATeX Graphics Companion*.

You can purchase this CD-ROM from TUG and all other TeX user groups.

Full technical details, prices, and ordering information can be found on the following Web site:

<http://www.tug.org/tex-live.html>